

# Customizable Domain-Specific Computing

**Jason Cong**

University of California

**Glenn Reinman and Alex Bui**

University of California

**Vivek Sarkar**

Rice University

To meet computing needs and overcome power density limitations, the computing industry has entered the era of parallelization. However, highly parallel, general-purpose computing systems face serious challenges in terms of performance, energy, heat dissipation, space, and cost. We believe that there is significant opportunity to look beyond parallelization and focus on domain-specific customization to bring significant power-performance efficiency improvement.

■ **TO MEET EVER-INCREASING** computing needs and overcome power density limitations, the computing community has halted simple processor frequency scaling and entered the era of parallelization, with tens to hundreds of computing cores integrated in a single processor, and hundreds to thousands of computing servers in a warehouse-scale data center. Such highly parallel, general-purpose computing systems, however, still face serious challenges in terms of performance, power, heat dissipation, space, and cost. We believe that there is significant opportunity to look beyond parallelization and focus on domain-specific customization to bring orders-of-magnitude power/performance efficiency improvement to important application domains. We are motivated in this endeavor by the following three observations.

First, each user or enterprise typically has a high computing demand only in one or a few selected application domains (e.g., graphics for game developers, circuit simulation for IC designers, financial analytics for investment banks), while its other computing needs (e.g., email, word processing, web browsing) can be easily satisfied by available computing technologies. Therefore, it is possible to develop a customizable computing platform in which computing engines and interconnects are specialized for that particular application domain, gaining significant improvements in

power/performance efficiency over that available with a general-purpose architecture.

Second, the performance gap between a totally customized solution (using an ASIC) and a general-purpose solution can be very large. For example,

Schaumont and Verbauwhede presented a case study of the 128-bit key AES (Advanced Encryption Standard) algorithm.<sup>1</sup> In that case study, an ASIC implementation in 0.18- $\mu\text{m}$  CMOS achieved 3.86 Gbits/second at 350 mW, while the same algorithm coded in Java and executed on an embedded Sparc processor yielded 450 bits/second at 120 mW. This difference implies a power/performance efficiency gap (measured in Gbits/second/W) of a factor of roughly 3 million. (Other implementation alternatives were also studied in the same paper, including the use of FPGAs and StrongARM processors.)

Third, it is extremely costly and impractical to implement each application in ASICs—the non-recurring engineering cost of an ASIC design in current 45-nm CMOS technology is more than \$50,000,000, and the design cycle can easily exceed a year. There is a strong need for a novel architecture platform that can be efficiently customized to a wide range of applications in a domain or a set of domains to bridge the huge power/performance gap between ASICs and general-purpose processors. To a large extent, the human brain is a good example of such a customizable platform. Moore's-law-like scaling has long stopped in neuroanatomy—the number of neurons and their speed have not increased in tens of thousands of years. Instead, most efficiency comes from specialization. If we look back through

history, we see that more advanced societies usually had a higher degree of specialization, yet all achieved with the same platform, the human brain. Today, just as we train people to be specialists such as doctors, lawyers, and engineers, the most effective way to cope with rapidly growing computational complexity is through specialization of customizable computing platforms for each application domain.

## Rationale

Given these observations, it is important to develop a general (and reusable) methodology for designing highly efficient customizable architecture platforms for domain-specific computing and the associated compilation tools and runtime management environment to support such a methodology. To pursue this idea, we formed the Center for Domain-Specific Computing (CDSC), primarily funded by the National Science Foundation with an award from the 2009 Expeditions in Computing Program. CDSC consists of 12 faculty members from multiple disciplines, including computer science and engineering, electrical engineering, medicine, and applied mathematics, from four universities: University of California, Los Angeles (the lead institution), Rice, UC Santa Barbara, and Ohio State.

The goal of the CDSC project is to demonstrate that achieving an orders-of-magnitude computing efficiency improvement for domain-specific applications is possible, and, moreover, that we can obtain such an improvement through a high degree of automation and reuse, so that our platform and methodology can be deployed in a wide range of application domains.

Although the basic concept of customizable architecture was introduced in the 1960s by Gerald Estrin,<sup>2</sup> early successes in customizable computing were demonstrated in the 1990s, where certain computation-intensive kernels were manually mapped to FPGAs for acceleration, achieving significant speedup. Examples include the DECPeRLe-1 system developed at DEC-PRL, the GARP project at UC Berkeley, and commercial efforts by Cray and SRC Computers (please refer to the recent survey by Hauck and DeHon for details;<sup>3</sup> more examples of related work can be found in various proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines).

These early efforts, however, faced several limitations, such as a communication bottleneck between the host CPU and FPGAs; customization that was limited to FPGAs with little or no integration of the latest multicore architectures; a lack of high-performance reconfigurable interconnect structures; scalability limitations; and a restricted programming environment that often required manual coding in hardware design languages or extensive rewriting of existing software for the FPGA implementation.

Our research on domain-specific computing has addressed these challenges by providing

- a wide range of customizable computing elements, from coarse-grained customizable cores to fine-grained field-programmable circuit fabrics;
- customizable and scalable high-performance interconnects based on the RF-interconnect (RF-I) or optical interconnect technologies;
- highly automated compilation tools and runtime management systems to enable rapid development and deployment of domain-specific computing systems, and
- a general, reusable methodology for retargeting to different application domains.

To better understand the potential of customizable domain-specific computing, we have analyzed the medical image processing domain, which has a high computing demand and is of great importance to healthcare.

## A case study: medical image processing domain

Imaging is now a routine clinical tool in the diagnosis and treatment of most medical problems, but many advances in this field have been constrained to the research environment due to a lack of computational power. Several medical image processing algorithms are infeasible for real-time clinical use; and objective, automated quantitative methods that can enhance detection and evaluation are not widely used. Power and cost-efficient high-performance computation in this domain can have a significant impact on health care in terms of preventive medicine (e.g., virtual colonoscopy for colorectal cancer screening), diagnostic procedures (e.g., automatic quantification of tumor volume), and therapeutic procedures (e.g., presurgical decision-making and monitoring and analysis during surgery).

**Table 1. Algorithms in medical imaging pipeline.**

Step	Computation kernel	Communication scheme	Representative algorithm
Reconstruction	Dense and sparse linear algebra, optimization methods	Iterative; local or global communication	Compressive sensing
Restoration	Sparse linear algebra, structured grid, optimization methods	Noniterative; highly parallel; local and global communication	Total variational algorithms
Registration	Dense linear algebra, optimization methods	Parallel, global communication	Optical flow/fluid registration
Segmentation	Dense linear algebra, spectral methods, MapReduce	Local communication	Level set methods
Analysis	Sparse linear algebra, n-body methods, graphical models	Local communication	Navier-Stokes equations

Medical imaging consists of a processing pipeline, typically with the following steps:

1. *Image reconstruction*: Computes a series of images from physical sensor data (such as x-rays or magnetic pulse sequences).
2. *Image restoration*: Removes noise and image artifacts from the image, such as those caused by environmental conditions or patient movement.
3. *Registration*: Orients a given image to a reference image (e.g., an image of a healthy person or an earlier image of the same individual).
4. *Segmentation*: Identifies and extracts regions of interest in the image (e.g., a tumor).
5. *Analysis*: Performs many kinds of feature analysis, such as measuring the size of a tumor, computing its growth rate (based on past segmentation results), etc.

Table 1 lists some typical algorithms used in these steps and their computation and communication patterns. Algorithms can vary considerably from one step to another, requiring different architecture support for the best efficiency.

In our preliminary studies, we looked at the possibility of using graphics processing units (GPUs) and FPGAs for acceleration. For example, for a biharmonic registration algorithm, the GPU provided a  $93\times$  speedup, while the FPGA (Virtex-4 LX100) provided an  $11\times$  speedup (both measured against a Xenon 2-GHz processor). However, for a 3D median denoising filter algorithm, the GPU provided a  $70\times$  speedup, while the FPGA achieved close to a  $1,000\times$  speedup (due to bit-level parallel operations).

Clearly, even in this rather narrow domain, no single homogeneous architecture can perform well on all these applications. This example underscores the need for customization: an ability to adapt architectures to match an application's computation and communication requirements.

### Overall approach

To realize the order-of-magnitude power/performance efficiency improvement via customization, yet still leverage economy of scale, we are developing a *customizable heterogeneous platform* (CHP), consisting of a heterogeneous set of adaptive computational resources connected by high-bandwidth, low-power, nontraditional, reconfigurable interconnects. Specifically, the CHP includes

- integration of customizable cores and coprocessors that will enable power-efficient performance tuned to the specific needs of an application domain; and
- reconfigurable high-bandwidth and low-latency on- and off-chip interconnects, such as RF-interconnects and optical interconnects, which can be customized to specific applications.

Figure 1 shows an example CHP configuration, with a set of fixed cores, customizable cores, programmable fabric, and a set of distributed cache banks. The design uses a reconfigurable off-chip optical bus to supply the bandwidth necessary to feed these components and a reconfigurable on-chip RF-I bus for high-bandwidth, low-latency communication between them. Figure 1 is a single point in a large design space; the key

consideration in selecting a design from this space is flexibility.

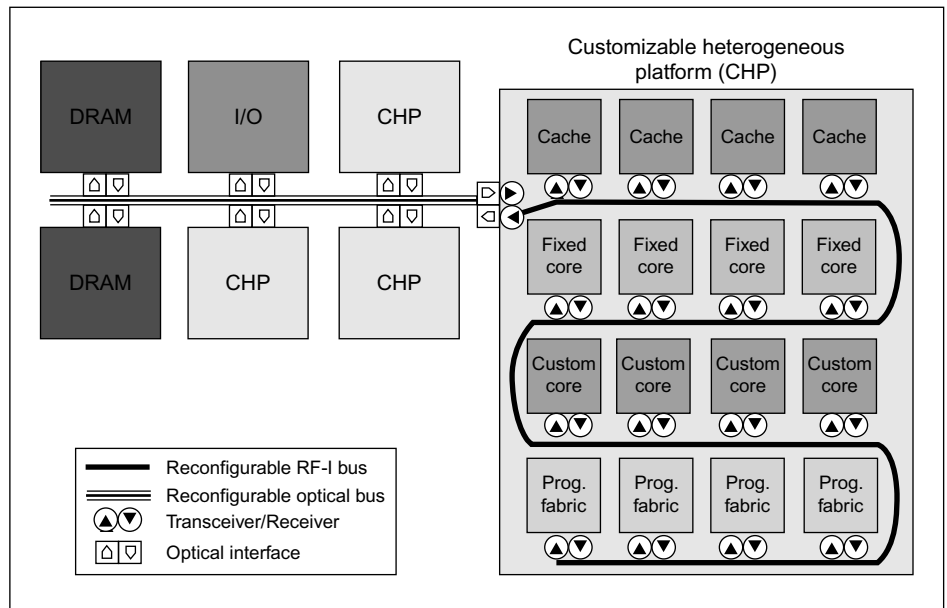
### Customizable computing engines

Three component types that exhibit different levels of customization and parallelism are considered in CHP designs.

- *Fixed cores* can vary dramatically in their energy efficiency, computational power, and area, but have limited reconfigurability: they can mainly make use of techniques like voltage or frequency scaling to adapt power/performance characteristics.

An example of this kind of architecture is the IBM Cell, with a general-purpose power processor element (PPE) core and the more numerous, but simpler, synergistic processor element (SPE) cores. In a recent study, we designed ParallAX,<sup>4</sup> a chip multiprocessor with heterogeneous fixed cores for physics-based animation. ParallAX combines a larger number of simple, fine-grained computing resources with a smaller number of more powerful, coarse-grained computing resources to handle a diverse physics workload, along with a flexible allocation scheme between coarse- and fine-grained cores. Kumar et al.<sup>5</sup> have demonstrated reductions in energy delay product as high as 6.35× for certain applications when dynamically switching between heterogeneous fixed cores.

- *Customizable cores* provide coarse-grained adaptation to application demand, offering a number of discrete, tunable options that can be set, with flexibility somewhere between FPGAs and fixed cores. It is possible to design cores with a rich set of tunable characteristics, such as register file sizes, cache sizes, datapath bit width, operating frequency, and supply voltages. Lee and Brooks<sup>6</sup> found that for a single customized core, up to a 5.3× improvement in efficiency can be achieved through intelligent adaptation (but the most useful set of tunable architectural features varied from application to application). We used the



**Figure 1. Example platform for domain-specific computing.**

- *Programmable fabrics* provide maximal flexibility by implementing custom instructions and specialized coprocessing engines to offload computation or accelerate core performance. They can implement complex operations with completely customizable architecture, in terms of the number of computing units, the types of computing units, the level of pipeline stages, and so on. Implementation of such customized circuits on programmable fabrics can be achieved efficiently with automatic C-to-FPGA compilation.<sup>7</sup>

In addition to these customizable components, designers can include dedicated function blocks (accelerators) that are frequently used in the given application domain, such as discrete cosine transform (DCT) and fast Fourier transform (FFT) functions for medical image processing. We identify custom instructions for customizable cores, customized coprocessors in programmable fabric, and dedicated functionality using both top-down and

bottom-up design approaches. In the top-down approach, we rely on domain-specific extensions to specify such complex functions. Based on the usage frequency and potential power/performance efficiency improvement of commonly used complex functions, designers can decide whether they should implement the functions by custom instructions, coprocessors, or dedicated logic. In the bottom-up approach, we analyze the underlying computational graphs for making such decisions. For example, we have developed techniques based on efficient cut enumeration in data-flow graphs (DFGs) for identifying custom instruction candidates and efficient data-mining techniques for identifying approximate program patterns for either custom instructions or dedicated functions.

The CDSC will also explore more sophisticated forms of software-directed resource transformation, including composing multiple simpler cores into a more complex virtual core type for executing sequential code, via core-spilling<sup>8</sup> or core fusion<sup>9</sup>, for instance.

#### Customizable interconnects

In addition to customizable computing engines, our CHP architecture will provide low-latency, high-bandwidth, and reconfigurable interconnects for data sharing between cores, coprocessors, cache banks, and memory banks. It will accommodate the communication requirements of a particular application (or even different phases of the same application). We are first considering adapting conventional interconnects to an application domain's demands (e.g., using express-virtual channels<sup>10</sup>). Later efforts by the CDSC will explore the use of novel interconnect technologies, in particular, RF-interconnect.<sup>11</sup> RF-I offers significant advantages: higher bandwidth, lower latency, better power efficiency, and compatibility with existing CMOS technology, making it particularly attractive for on-chip communication. (Optical interconnect has similar advantages but requires modification of existing CMOS technologies, presently making it difficult for on-chip integration. But our CHP is amendable to optical interconnects.)

The most significant advantage of these interconnects, in the context of customized computing, is their ability to provide an application-specific interconnect topology. Such reconfiguration can be achieved by selectively allocating RF-I bandwidth between different components on-chip. Multiple

bandwidth channels can coexist on the shared RF-I waveguide for simultaneous communication. In application phases that stream data from memory to a distributed set of processing resources or to coprocessors, we can set up dedicated communication channels by selective frequency assignment. Or, in phases where synchronization occurs at shared data boundaries, we can set up communication channels between logically related processing units. RF-I is also amenable to multicast and can therefore be configured to accelerate broadcast-based cache coherence or synchronization primitives.<sup>11</sup>

#### Application modeling and software design

There is a natural tension between hardware-first and software-first approaches to building domain-specific systems. In the former, designers start with a representative workload of existing domain applications and use their characteristics to create a customized CHP that, in turn, drives the design of a domain-specific language and compiler extensions. The hardware-first approach represents the usual practice of “software playing second fiddle to hardware,” exemplified by recent multicore processors such as the Cell, for which investigation of general-purpose programming models and compiler innovations started only after the hardware design was completed.

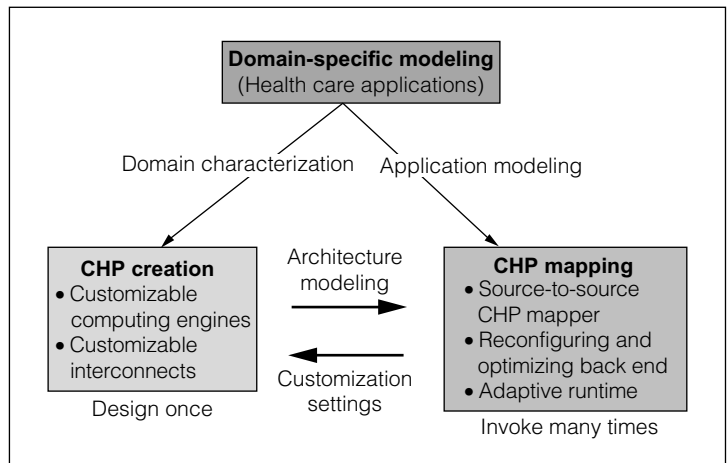
In the software-first approach, designers use the workload characteristics to drive the definition of a domain-specific programming language and compiler extensions that, in turn, drive the CHP's creation. The software-first approach has been pursued less often and has the danger of leading to special-purpose hardware with applicability to narrow application domains. Because there are well-known limitations in both approaches, we instead use the following three-stage approach to carefully balance software and hardware considerations in the spirit of software-hardware codesign:

1. *Domain-specific modeling.* In this stage, we create a representative set of executable application models, using *domain-specific language extensions* and a *domain-specific coordination graph* notation, both designed to be accessible to domain (programming) experts. We can use these models to reveal inherent high-level properties of the application domain, such as intrinsic parallelism and communication topologies. This domain-specific

modeling serves as an input to later CHP creation and CHP mapping stages.

2. *CHP creation.* Next, we use the application models to design and implement an optimized set of hardware resources (that is, the CHP) for a particular application domain (or a set of related domains). The CHP creation determines how many cores, how much cache, which custom instructions, how much customization and reconfiguration, and what sorts of mapping transformations are useful for customization. The decision is driven by a set of representative applications with domain-specific modeling specifying their computation requirements.
3. *CHP mapping.* Given a set of application models and an optimized CHP for a given domain, the third problem is CHP mapping. This develops domain-specific compilation and runtime systems that enable optimized mappings of the applications in the domain to the heterogeneous resources of the CHP. This stage also determines the configuration and transformation settings that we should select for configurable CHP resources in different program phases. For example, we might determine that a sequential part of an application should map to the most powerful out-of-order core in our CHP—and it may be that the only way to provide such a powerful core is with some core composition technique (i.e., a transformation that is presumed available to us). Or we may determine that a large number of simple in-order cores must be configured to communicate using a 3D mesh (e.g., for a volumetric image set decomposition).

Figure 2 shows this three-stage approach. Supporting such an approach is difficult with current general-purpose programming models for heterogeneous hardware, such as Nvidia’s CUDA (compute unified device architecture) or the Cell SDK (software developer kit). These models force the programmer to exploit customizable hardware at the lowest possible levels of hand-partitioned code and explicit data transfers tied to specific hardware structures. Such frameworks result in significant rewrites when the application must be repartitioned for execution on different hardware configurations or platforms. These approaches also miss out on opportunities for hardware customization for different application phases. In contrast, we expect that our three-stage approach

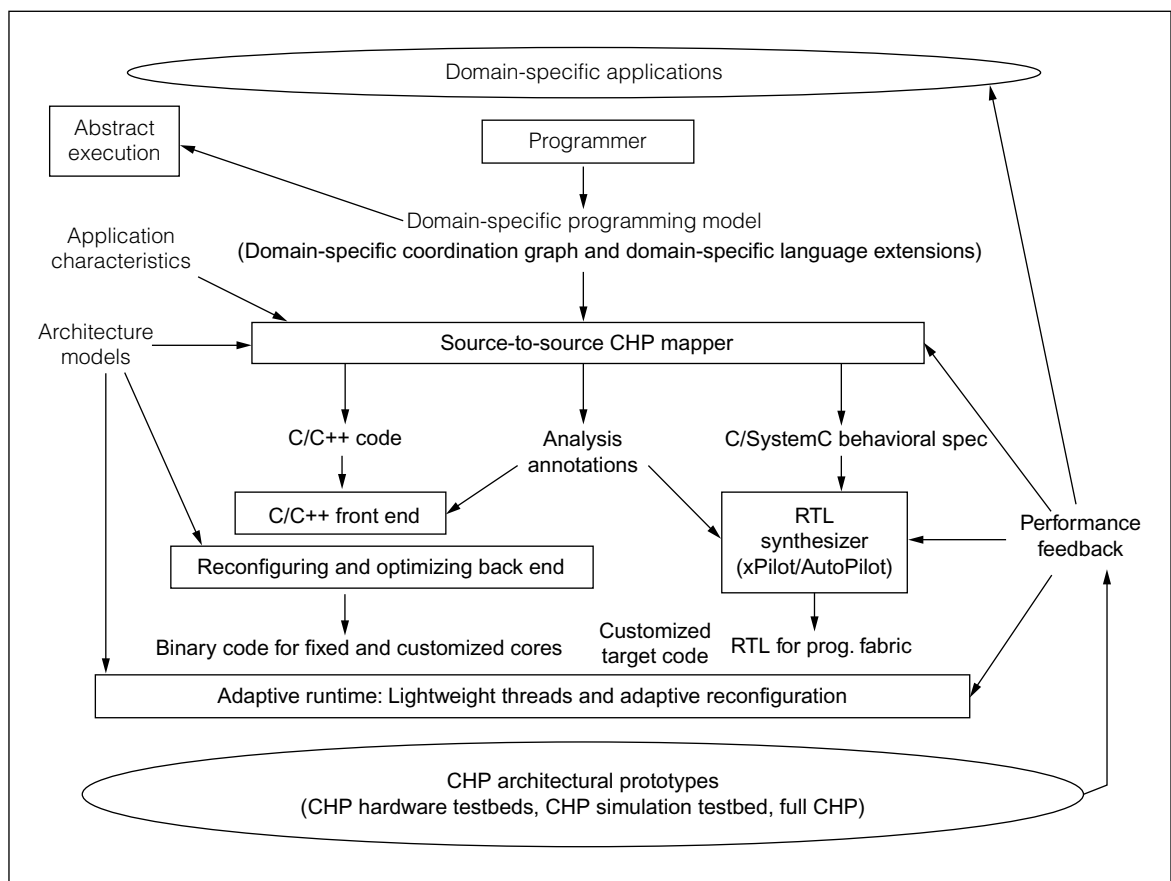


**Figure 2. The Center for Domain-Specific Computing approach to designing customizable architecture platforms involves three development stages: domain-specific modeling, customizable heterogeneous platform (CHP) creation, and CHP mapping.**

will balance software and hardware considerations to better expose opportunities for improvements in computing efficiency, while using a software approach that supports automation and reuse and that is accessible to domain experts.

Figure 3 outlines the structure of the software stack we are building to support mapping of a domain-specific application to a CHP. The domain experts start with a domain-specific programming model, which includes a domain-specific coordination graph that describes the inherent concurrency, dependency, and locality of the applications in the given domain. The model also includes domain-specific language extensions that can be used to easily describe, for example, the domain’s data types and computational patterns. Then, an automated mapping and synthesis flow performs source-to-source transformation and compilation to generate optimized binaries for heterogeneous multicore processors and the RTL code for the customized accelerators in the CHP. A lightweight runtime system supports runtime resource management and customization.

Although we are targeting medical imaging applications as a demonstration vehicle for our project, the three-stage approach and the hardware-software design techniques proposed for each stage are general and reusable for domain-specific computing and applicable to other domains. Such a reusable design methodology and software mapping flow will significantly lower the cost of creating new



**Figure 3. Overview of compilation and runtime system structure. The domain experts build a domain-specific programming model; then an automated mapping and synthesis flow performs source-to-source transformation and compilation to generate optimized binaries for multicore processors and RTL code. A lightweight runtime system supports runtime resource management and customization.**

domain-specific CHPs. In fact, a generic CHP could even be introduced for general-purpose computing. It would include a mix of general-purpose cores with different capabilities, field-programmable logic (via on-chip or 3D integration) for accelerators (instead of customized logic fixed at the design or fabrication time), and a customizable network-on-chip (NoC). In general, there will be a trade-off between the domain scope covered by a CHP and its energy efficiency.

### Progress and early results

As the first step of CHP creation, we have developed an elaborative simulation infrastructure based on Virtutech's Simics and the open-source General Execution-driven Multiprocessor Simulator (GEMS) originally developed by the University of Wisconsin, but with a number of significant extensions. These extensions include support of heterogeneous cores;

tightly coupled and loosely coupled accelerators; hybrid cache structures; new hierarchical memory coherence protocols; heterogeneity in NoCs; support for reconfigurable alternative interconnects; and accurate power, performance, and area models.

Using this simulation infrastructure, we are exploring and validating several novel architectural ideas and options as part of the CHP designs. For example, we designed a customizable hardware cache that supports

- dynamic allocation of a shared pool of cache blocks between a conventional cache and scratchpad memory (SPM),
- a flexible SPM mapping scheme that maps SPM blocks onto cache sets with adaptive reconfiguration at runtime, and
- software pipelining operations through SPM prefetching.

Our experimental results on SPEC2006 benchmarks and medical imaging applications show that this proposed scheme can achieve considerable performance improvement and energy reduction, compared with a hardware cache alone or a limited reconfigurable hybrid cache.

Another example is a hardware resource management scheme we developed for accelerator sharing. The scheme supports sharing and arbitration of multiple cores for a common set of accelerators and includes an efficient cache management scheme for accelerators to mitigate memory latency by overlapping data transfer with computation. Simulation results show significant improvements in both performance and energy efficiency compared with approaches using OS-based accelerator management. Our approach achieved an average 9× improvement in performance and a 32× improvement in energy efficiency, with minimal hardware overhead (less than 0.01% of chip area).<sup>12</sup>

For domain-specific modeling, the CDSC team has been collaborating with Intel on the Concurrent Collections (CnC) model, which forms the foundation of the domain-specific coordination graph. CnC is a declarative and implicitly parallel coordination language that supports flexible combinations of task and data parallelism while retaining determinism. CnC computations are built in sequential steps related by data and control dependence edges, which are represented by a CnC graph.

We developed a semantic model for CnC, called Featherweight CnC, which simplifies the full CnC language without reducing its power. The Featherweight CnC model was used to obtain the first known formal proof of the determinism property for the CnC language. We have also extended CnC by creating a model called CnC-CUDA that supports execution of steps on both CPUs and GPUs.<sup>13</sup> The CnC-CUDA extensions include the definition of multithreaded steps for execution on GPUs and automatic generation of data and control flow between CPU steps and GPU steps. Experimental results show that this approach can yield significant performance benefits with both GPU execution and hybrid CPU/GPU execution. Various medical imaging applications are now being modeled with CnC.

For CHP mapping, we selected the open-source Rose compiler infrastructure developed at Lawrence Livermore National Laboratory as the baseline infrastructure for the high-level mapper, and the

open-source LLVM (Low Level Virtual Machine) as the baseline infrastructure for the optimizing back end. An important development in our work is the creation of Habanero-C, a language, compiler, and runtime that integrates the following four orthogonal constructs with the C language to support task parallelism:

- the `async` and `finish` constructs for lightweight dynamic task creation and termination,<sup>14</sup>
- the `place` construct for locality control in task and data distributions,
- the `isolated` construct for mutual exclusion and isolation among tasks, and
- the `phasers` construct for collective and point-to-point synchronization and reduction.

Our CHP mapping infrastructure also interfaces with UCLA's C/C++ to RTL high-level synthesis tool xPilot<sup>7</sup> and its successor AutoPilot (from AutoESL, now part of Xilinx) for generation of various accelerators implemented with dedicated circuits or using the programmable fabric on the CHP.

**ALTHOUGH THE FINAL** CHP is intended to be a system-on-a-chip implementation, we have built an initial heterogeneous prototype for testing the software mapping flow. This prototype comprises one Intel Core i7 CPU, two Nvidia Tesla C1060s, and one Xilinx ML605 (Virtex6 LX240T) card. The motherboard of this system has four PCI-express slots. The Tesla computing cards and FPGA card are all integrated by the PCI-express subsystem. We have developed multiple implementation templates, corresponding device drivers, and basic APIs for direct memory access data transfer, computing kernel invocation, and synchronization. Currently, we are developing a CnC-based flow for partitioning the tasks in the medical image-processing pipeline to heterogeneous components, including processors, GPUs, and FPGAs. ■

## Acknowledgments

The Center for Domain-Specific Computing is funded by the National Science Foundation's Expedition in Computing Award CCF-0926127. Other CDSC faculty members are Denise Aberle, Richard Baraniuk, Frank Chang, Tim Cheng, Jens Palsberg, Miodrag Potkonjak, P. Sadayappan, and Luminita Vese. We greatly appreciate their participation in this



project. We also thank Yi Zou at UCLA for providing GPU and FPGA acceleration results on several medical imaging applications.

## References

1. P. Schaumont and I. Verbauwhede, "Domain-Specific Codesign for Embedded Security," *Computer*, vol. 36, no. 4, 2003, pp. 68-74.
2. G. Estrin, "Organization of Computer Systems—The Fixed Plus Variable Structure Computer," *Proc. Western Joint Computer Conf.*, AFIPS Press, 1960, pp. 33-40.
3. S. Hauk and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, Morgan Kaufmann, 2007.
4. T. Yeh et al., "ParallAX: An Architecture for Real-Time Physics," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA 07)*, ACM Press, 2007, pp. 232-243.
5. R. Kumar et al., "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO 36)*, IEEE CS Press, 2003, pp. 81-92.
6. B. Lee and D. Brooks, "Efficiency Trends and Limits from Comprehensive Microarchitectural Adaptivity," *Proc. 13th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS 08)*, ACM Press, 2008, pp. 36-47.
7. J. Cong et al., "Platform-Based Behavior-Level and System-Level Synthesis," *Proc. IEEE Int'l SOC Conf.*, IEEE Press, 2006, pp. 199-202.
8. J. Cong et al., "Accelerating Sequential Applications on CMPs Using Core Spilling," *IEEE Trans. Parallel and Distributed Systems*, vol. 18, no. 8, 2007, pp. 1094-1107.
9. E. Ipek et al., "Core Fusion: Accommodating Software Diversity in Chip Multiprocessors," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA 07)*, ACM Press, 2007, pp. 186-197.
10. A. Kumar et al., "Express Virtual Channels: Towards the Ideal Interconnection Fabric," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA 07)*, ACM Press, 2007, pp. 150-161.
11. M.F. Chang et al., "Power Reduction of CMP Communication Networks via RF-Interconnects," *Proc. 41st IEEE/ACM Ann. Int'l Symp. Microarchitecture (MICRO 41)*, IEEE CS Press, 2008, pp. 376-387.
12. J. Cong et al., "AXR-CMP: Architecture Support in Accelerator-Rich CMPs," *Proc. 2nd Workshop on SoC Architecture, Accelerators and Workloads (SAW-2)*, IEEE CS Press, 2011, in press.
13. M. Grossman et al., "CnC-CUDA: Declarative Programming for GPUs," *Proc. Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2010; to appear in Springer-Verlag LNCS 6548.
14. Y. Guo et al., "SLAW: A Scalable Locality-Aware Adaptive Work-Stealing Scheduler," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS 10)*, IEEE CS Press, 2010, pp. 1-10.


**Jason Cong** is Chancellor's Professor and director of the Center for Domain-Specific Computing (CDSC) at the University of California, Los Angeles (UCLA). His research interests include synthesis of VLSI circuits and systems, programmable systems, novel computer architectures, and nanosystems. He has a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a Fellow of the ACM and IEEE.

**Vivek Sarkar** is a professor of computer science, E.D. Butcher Chair in Engineering, at Rice University and the associate director of CDSC, where he leads the domain-specific modeling and CHP mapping thrusts. His research interests include multiple aspects of parallel software including programming languages, program analysis, compiler optimizations, and runtimes for parallel and high-performance computer systems. He has a PhD in computer science from Stanford University. He is an ACM Fellow.

**Glenn Reinman** is an associate professor in the Computer Science Department at UCLA and leads the customizable heterogeneous platform creation thrust in the CDSC. His research interests include microprocessor design, parallel programming, and virtual/augmented reality. He has a PhD in computer science from University of California, San Diego.

**Alex Bui** is an associate professor in the Department of Radiological Sciences at UCLA and leads the application driver thrust in CDSC. His research interests include distributed medical information systems, data modeling, medical data visualization, and applications of medical image processing. He has a PhD in computer science from UCLA in 2000. He is a member of IEEE.

Direct questions and comments about this article to Jason Cong, UCLA Computer Science Department, 4731J Boelter Hall, Los Angeles, CA 90095; cong@cs.ucla.edu.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

# IEEE computer society

**PURPOSE:** The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.

**MEMBERSHIP:** Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

**COMPUTER SOCIETY WEBSITE:** [www.computer.org](http://www.computer.org)

**OMBUDSMAN:** To check membership status or report a change of address, call the IEEE Member Services toll-free number, +1 800 678 4333 (US) or +1 732 981 0060 (international). Direct all other Computer Society-related questions—magazine delivery or unresolved complaints—to [help@computer.org](mailto:help@computer.org).

**CHAPTERS:** Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

**AVAILABLE INFORMATION:** To obtain more information on any of the following, contact Customer Service at +1 714 821 8380 or +1 800 272 6657:

- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years practice and significant performance in five of those 10)

## PUBLICATIONS AND ACTIVITIES

**Computer:** The flagship publication of the IEEE Computer Society, *Computer*, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.

**Periodicals:** The society publishes 13 magazines, 18 transactions, and one letters. Refer to membership application or request information as noted above.

**Conference Proceedings & Books:** Conference Publishing Services publishes more than 175 titles every year. CS Press publishes books in partnership with John Wiley & Sons.

**Standards Working Groups:** More than 150 groups produce IEEE standards used throughout the world.

**Technical Committees:** TCs provide professional interaction in more than 45 technical areas and directly influence computer engineering conferences and publications.

**Conferences/Education:** The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.

**Certifications:** The society offers two software developer credentials. For more information, visit [www.computer.org](http://www.computer.org)/certification.

## NEXT BOARD MEETING

23–27 May 2011, Albuquerque, NM, USA

## EXECUTIVE COMMITTEE

**President:** Sorel Reisman\*

**President-Elect:** John W. Walz\*

**Past President:** James D. Isaak\*

**VP, Standards Activities:** Roger U. Fujii†

**Secretary:** Jon Rokne (2nd VP)\*

**VP, Educational Activities:** Elizabeth L. Burd\*

**VP, Member & Geographic Activities:** Rangachar Kasturi†

**VP, Publications:** David Alan Grier (1st VP)\*

**VP, Professional Activities:** Paul K. Joannou\*

**VP, Technical & Conference Activities:** Paul R. Croll†

**Treasurer:** James W. Moore, CSDP\*

**2011–2012 IEEE Division VIII Director:** Susan K. (Kathy) Land, CSDP†

**2010–2011 IEEE Division V Director:** Michael R. Williams†

**2011 IEEE Division Director V Director-Elect:** James W. Moore, CSDP\*

\*voting member of the Board of Governors †nonvoting member of the Board of Governors

## BOARD OF GOVERNORS

**Term Expiring 2011:** Elisa Bertino, Jose Castillo-Velázquez, George V. Cybenko, Ann DeMarle, David S. Ebert, Hironori Kasahara, Steven L. Tanimoto

**Term Expiring 2012:** Elizabeth L. Burd, Thomas M. Conte, Frank E. Ferrante, Jean-Luc Gaudiot, Paul K. Joannou, Luis Kun, James W. Moore

**Term Expiring 2013:** Pierre Bourque, Dennis J. Frailey, Atsuhiko Goto, André Ivanov, Dejan S. Milojcic, Jane Chu Prey, Charlene (Chuck) Walrad

## EXECUTIVE STAFF

**Executive Director:** Angela R. Burgess

**Associate Executive Director; Director, Governance:** Anne Marie Kelly

**Director, Finance & Accounting:** John Miller

**Director, Information Technology & Services:** Ray Kahn

**Director, Membership Development:** Violet S. Doan

**Director, Products & Services:** Evan Butterfield

**Director, Sales & Marketing:** Dick Price

## COMPUTER SOCIETY OFFICES

**Washington, D.C.:** 2001 L St., Ste. 700, Washington, D.C. 20036

**Phone:** +1 202 371 0101 • **Fax:** +1 202 728 9614

**Email:** [hq.ofc@computer.org](mailto:hq.ofc@computer.org)

**Los Alamitos:** 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314

**Phone:** +1 714 821 8380

**Email:** [help@computer.org](mailto:help@computer.org)

## MEMBERSHIP & PUBLICATION ORDERS

**Phone:** +1 800 272 6657 • **Fax:** +1 714 821 4641 • **Email:** [help@computer.org](mailto:help@computer.org)

**Asia/Pacific:** Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan

**Phone:** +81 3 3408 3118 • **Fax:** +81 3 3408 3553

**Email:** [tokyo.ofc@computer.org](mailto:tokyo.ofc@computer.org)

## IEEE OFFICERS

**President:** Moshe Kam

**President-Elect:** Gordon W. Day

**Past President:** Pedro A. Ray

**Secretary:** Roger D. Pollard

**Treasurer:** Harold L. Flescher

**President, Standards Association Board of Governors:** Steven M. Mills

**VP, Educational Activities:** Tariq S. Durrani

**VP, Membership & Geographic Activities:** Howard E. Michel

**VP, Publication Services & Products:** David A. Hodges

**VP, Technical Activities:** Donna L. Hudson

**IEEE Division V Director:** Michael R. Williams

**IEEE Division VIII Director:** Susan K. (Kathy) Land, CSDP

**President, IEEE-USA:** Ronald G. Jensen