

CRISP: Center for Research in Intelligent Storage and Processing in Memory

Kevin Skadron

Dept. of Computer Science
University of Virginia
Charlottesville, VA, USA

José F. Martínez

Computer Systems Laboratory
Cornell University
Ithaca, NY, USA

Yuan Xie

Dept. of Electrical & Computer Engineering
University of California at Santa Barbara
Santa Barbara, CA, USA

Steven Swanson

Dept. of Computer Science & Engineering
University of California, San Diego
La Jolla, CA, USA

Jignesh Patel

Dept. of Computer Science
University of Wisconsin
Madison, WI, USA

Abstract—The CRISP team will pursue research to enable processing to happen as close to data as possible and eliminate bottlenecks due to data motion. This will require innovations across the system stack. Ultimately, we seek to lower the effort barrier significantly for everyday programmers. The goal is to achieve highly portable, “bare-metal,” and understandable performance across a wide range of heterogeneous systems.

Keywords—processor, memory, storage, operating system

I. INTRODUCTION

In today’s Big Data era, the volume of data is exploding, with data coming from a growing variety of video, sensors, and informatics platforms. Users need support for both throughput-oriented and latency-sensitive tasks on vast quantities of archived data—all while optimizing for energy efficiency and other resource costs and providing reliability and security. Support for ever-bigger-data computation, in turn, enables new real-time capabilities and new degrees of insight, enabling novel applications and markets. Yet today’s systems are increasingly bottlenecked by data movement costs. This trend is compounded by the fact that data in memory is generally stored with little or no consideration of data content, usage patterns, or how data are grouped into higher-level objects. Many algorithms also exhibit poor temporal locality (because they only touch data items once) and poor spatial locality (because they exhibit irregular access patterns across vast data sizes), making caches inefficient. A further source of inefficiency is that the raw bandwidth at the edge of the data arrays in all current memory and storage technologies is 1-2 orders of magnitude greater than what can be transmitted off chip.

However, traditional semiconductor technology scaling is stalling. The long-standing contract between applications and hardware, namely that clock speeds roughly double with every new generation of hardware, ended about a decade ago, making it hard for application performance to scale “effortlessly.” As a result, specialized hardware is becoming prevalent. However,

specialization efforts so far have primarily focused on increased *processing* efficiency, even though the intrinsic performance of today’s memory and storage architectures are grossly underutilized, because processing and storage are designed independently and deployed as separate resources. This means the high bandwidth of the data arrays is throttled by narrow interfaces and high power to move data at high speeds. As a result, systems are increasingly bottlenecked by data movement costs and are no longer able to keep up with this data explosion.

II. RESEARCH CHALLENGES

Solving these challenges and enabling the next generation of data-intensive applications requires computing to be embedded in and around the data, creating *intelligent memory and storage* (IMS) architectures that do as much of the computing as possible as close to the bits as possible. The closer computation is to the memory cells, the closer it comes to the raw bandwidth available in the storage arrays. This proximity will simultaneously address data and computational challenges. By embedding computation with the data arrays, or at least within the memory and storage chips, we can achieve massive increases in parallelism, and the resulting energy efficiency may allow a much higher density of computing and storage devices in a given form factor. However, such IMS capabilities will require reconstructing the entire system stack, with new architectural and operating-system abstractions, new memory semantics, and new techniques for compiling and optimization, and dynamic yet efficient system software.

In addition to these architectural trends, innovation in device technology complicates the landscape even further. Emerging non-volatile technologies are already transforming the memory/storage landscape. Each of these upcoming technologies offers different trade-offs for embedding processing near the data, in the data arrays themselves, or even

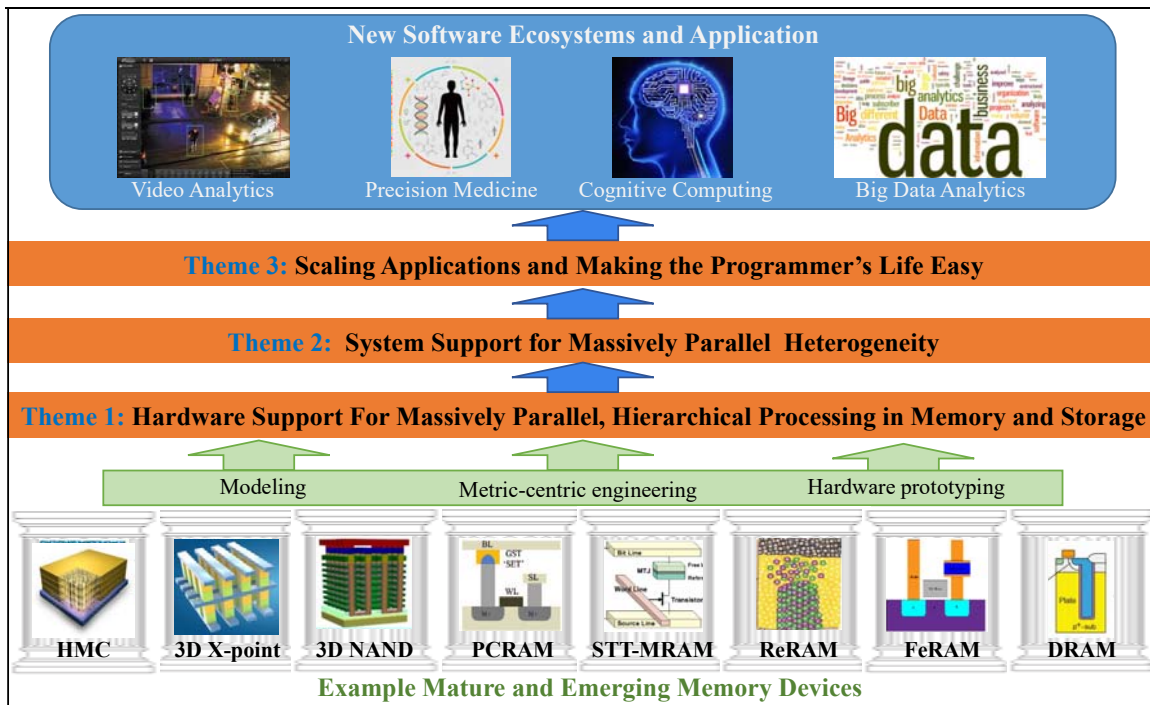


Fig. 1: Overview summarizing the CRISP research program’s application-driven development of intelligent storage and memory systems, incorporating effects of diverse device technologies.

directly in

the bit cells. These emerging technologies complicate the hardware design space. For all these reasons, programmability and computational efficiency must be addressed in an integrated fashion.

At the same time, we want to achieve high programmer productivity and code portability across diverse, heterogeneous architectures. Obtaining high performance and efficiency on data-intensive problems, or porting software to emerging hardware, should not require a “ninja” programmer. We must protect the user from this hardware and software ferment by providing a highly intuitive programming model that allows programmers to focus on *what* they want to do, instead of *how*, while nevertheless operating as close to the raw performance of the data arrays as possible. We must also provide programmers and system administrators with tools to reason about performance and efficiency, tune their algorithms, and navigate the Pareto-optimal frontier of performance, energy-efficiency, and precision. This will democratize high-performance, heterogeneous, data-intensive computing, to enhance productivity of the industrial and military workforce and enable an improved software ecosystem that opens new markets for computer systems.

III. APPROACH

In order to solve these challenges, the Center for Research on Intelligent Storage and Processing in Memory (CRISP) brings together a team of 20 faculty from eight leading research universities in a coordinated cross-stack research plan

spanning applications and programming frameworks,

system software support for IMS systems, and hardware. Fig. 1 presents a summary of our approach, comprising three main themes.

A. *Scaling Applications and Making the Programmer’s Life Easy*

CRISP is premised on application-driven innovation, in order to develop IMS capabilities in ways that transform real-world practice. Making IMS systems practical for everyday programmers will in turn require suitable programming frameworks.

1) *Application-driven innovation.* We will work with application experts across a set of diverse, data-intensive applications that will be commercially and socially important over the next 10-20 years. These include big-data analytics, video analytics, medical imaging, genomics, precision medicine, and cognitive computing. These are all data intensive, but exhibit a rich diversity of computation, data access, and communication patterns, and a need for real-time processing. With major improvements in throughput and latency, dramatic new capabilities and markets will emerge. By implementing our solutions in the context of real applications and real data, we are able to work through the details of how hardware and software features are best leveraged by applications, and drive innovative hardware and software features based on deep application insights. At the same time, this work will advance the state of the art in these application domains, enabling new capabilities and benefitting users, patients, etc.






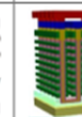


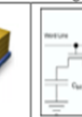

memory mechanism & device structure	ionic-electronic				electronic					
	resistive switching		spin-based		floating gate		charge-based			
										
acronym	ReRAM	CBRAM	PCRAM	FeRAM	STTRAM	3D NAND	2D NOR	3D DRAM	DRAM	SRAM
maturity	low	low	medium	medium	medium	high	very high	high	very high	very high
cell area	$\ll 4 F^2$ if 3D	$\ll 4 F^2$ if 3D	$4 F^2$ - $20 F^2$	$12 F^2$ - $22 F^2$	$6 F^2$ - $20 F^2$	$\ll 1 F^2$	$25 F^2$ - $40 F^2$	$< 1 F^2$	$6 F^2$	$< 150 F^2$
multi-bit	$\gg 1$	$\gg 1$	$\gg 1$	1	1	> 1	$\gg 1$	1	1	1
voltage	$< 3 V$	$< 3 V$	$< 3 V$	$< 1 V$	$< 2 V$	$< 20 V$	$< 12 V$	$< 1 V$	$< 1 V$	$< 1 V$
read time	$< 10 ns$	$< 10 ns$	$\sim 10 ns$	$< 20 ns$	$< 10 ns$	$50 - 100 \mu s$	$< 10 ns$	$< 50 ns$	$\sim 10 ns$	$< 1 ns$
P/E time	$< 10 ns$	$< 10 ns$	$\sim 50 ns$	$< 10 ns$	$< 10 ns$	$100 \mu s / 1 ms$	$10 \mu s / 100 ms$	$< 50 ns (?)$	$\sim 10 ns$	$< 1 ns$
retention	$> 10 yr$	$> 10 yr$	$> 10 yr$	$> 10 yr$	$> 10 yr$	$> 10 yr$	$> 10 yr$	$> 10 ms (?)$	$\sim 64 ms$	N/A
endurance	$> 1E8$	$> 1E12$	$> 1E8$	$> 1E12$	$> 1E15$	$> 1E3$	$> 1E5$	$> 1E16$	$> 1E16$	$> 1E16$
write energy / bit	$< 10 fj$	$< 10 fj$	$\sim 10 pJ$	$< 10 fj$	$\sim 0.1 pJ$	$< 10 fj (?)$	$\sim 10 pJ$	$> 50 fj (?)$	$\sim 10 fj$	$< fj$

Fig. 2. Key characteristics for emerging and conventional memory devices (adapted from Refs. [1,2]). Note that the reported write energy is specified for intrinsic memory cell operation and do not account for circuit level energy losses, which, e.g., are typically dominant in flash memories.

2) *Programming frameworks.* Intuitive programming requires raising the level of abstraction, with support from intelligent compiler and runtime environments, to achieve write-once, run-anywhere capability. We believe that the most effective approach is to provide high-level abstractions to operate on data sets and data streams instead of individual data elements. By providing common primitives, especially primitives that map naturally to an application domain, such as image analysis or genomics, programming becomes more intuitive. And by using declarative abstractions where possible, extracting what the programmer wants to do, instead of how, the system obtains rich information about how data and tasks are related, while maximizing flexibility to transform the software to run efficiently across widely varying architectures, without the need for painstaking manual porting to achieve the correct functionality and to achieve high performance.

A key component of our programming framework is recognizing that a single domain-specific language (DSL) is unlikely to be sufficiently “natural” across all application domains, and even within a single complex application, different parts of the application may need to speak different domain-specific languages to make it easier for the programmer to focus on the algorithmic specification. Our approach to support multiple DSLs intrinsically also has the desirable side-effect of cross-pollinating our research efforts so that the improvement in one DSL can broadly apply across multiple applications. Associated with our framework is also the separation of logical DSL optimization, and the mapping to a common directed acyclic graph of operators for all DSLs. This common mapping will allow us to build the run-time system once and use it multiple times across many DSLs and target architectures.

Ultimately, even more important than the specific abstractions/languages we develop, is a methodology for deriving new abstractions and languages as needed to support emerging applications, while leveraging a common mapping and optimization framework. This approach insulates the programmer from hardware details, enables a large software

base to use IMS, and ultimately benefits heterogeneous systems of all types.

B. Hardware Support for Massively Parallel Processing in Memory and Storage.

“Bare-metal” performance will be achieved by allowing computation to happen as close to the data as possible, and to leverage the full bit-level parallelism of the data arrays wherever possible. By “bare metal,” we mean to transmit to the application, with minimal loss, the full performance potential of the hardware. This requires novel hardware technologies for processing at different levels of the memory/storage hierarchy, and exploring what kind of mechanisms and abstractions this hardware should present to the software, with the goal of achieving performance and efficiency as close as possible to the inherent bit-level parallelism of the data arrays.

We will develop a modular but efficient processing hierarchy that provides intelligence within the memory/storage units and their interfaces. This includes processing in the data arrays, at the edges of the arrays, in the controllers, etc. At the same time, interfaces for accelerators will allow them to access the intrinsic throughput of the data arrays. We also envision chips with several layers of memory and/or storage, layered on top of, or packaged with, dedicated processing layers. These multi-chip IMS modules will then be combined via a scalable interconnect, to create nodes with high data capacity and high cross-section bandwidth. These, in turn, will scale from edge devices with a single IMS module to cloud systems with racks in which each node supports hundreds of terabytes of memory and petabytes of storage.

Our approach to developing new hardware mechanisms for adding intelligence into memory and storage will provide immediate benefits for existing technologies (e.g., DRAM, Flash, and Xpoint), but our solution will also be *technology-driven*, in order to leverage emerging technologies.

We will also evaluate how appropriate tailoring of emerging technology properties such as write latency and retention time can better support application needs. We will

have a focused effort on understanding both existing SRAM/DRAM/Flash memory and emerging NVM technology, e.g. those shown in Fig. 2—not just from the device-level understanding, but also from a utility point of view—that is, with an eye on the applications that might benefit from such technology. Also, importantly to the success of our project, we must understand how to model these new technologies, so that we can evaluate and quantify the impact on the architecture and higher up the system stack.

Effective IMS systems must also operate within acceptable power and thermal budgets, so these factors will be an integral part of the design space. Both static and dynamic thermal and power management will be considered.

By applying these principles across the system at various scales, and by developing solutions for diverse memory and storage technologies, we will create a set of building blocks for IMS systems, scaling from edge devices with a single IMS module to cloud systems with racks in which each node supports 100s of TB of memory and PBs of storage.

C. System Support for Massively Parallel Heterogeneity.

The system software sits in the middle: it is responsible for bridging the user-level software and the emerging architectures. This requires a high level of automation, while still giving users performance transparency and control over tradeoffs among performance, energy-efficiency, etc.

1) *Performance transparency.* Providing performance transparency addresses two key challenges in achieving “bare metal” performance. First, we plan to make algorithmic exploration easy, to identify the optimal combination of algorithm and mapping. Second, during such algorithm exploration and/or performance debugging, the application programmer needs help in understanding the root causes that may hinder performance. Such performance debugging is challenging today, and likely be even more challenging in the future with increasing complexity in the topology of storage and compute elements packaged in new architectural configurations. Thus, we plan to build introspection mechanisms in our run-time system, to help application programmers and system engineers understand why the application performed in a certain way. This understanding can then allow an application programmer to determine what change to make to their application logic.

Support for these capabilities will require appropriate hardware probes, semantically richer and more sophisticated than today's simple performance counters, with hardware plus runtime management mechanisms appropriate for our intelligent memory environment. We envision a hierarchy of hardware probes that are highly programmable; that can collect a wide array of data; and that possess the ability to preprocess the data they collect into higher-level knowledge—even conduct a modest degree of inference. The *system software* can then use these to guide automatic scheduling decisions or to provide reporting and diagnostic capabilities to the programmer.

2) *System software.* The system software is responsible for bringing computation and data together at the right place and time to achieve the user's needs, while brokering

resources among competing applications and requirements, and providing security, resilience, and performance transparency. This will require new abstractions and new system services.

For example, intelligent memory systems will be highly diverse, so the interfaces we devise for programming will need to represent computation and requirements in a way that is, to the extent possible, agnostic of the underlying hardware. We have lived with a fairly simple “address” based interface to data, whether it be to memory words or file/disk blocks. While this may have sufficed so far, such interfaces, although simple from a programmatic viewpoint, are woefully inadequate to exploit the rich computational capabilities within memory. Instead, the basic naming scheme will be object-oriented; if a new type of memory, processor, coprocessor, or interconnect appears on the market, our system will require the developer to provide implementations of basic operations and information about the device's behavior. Our system will use that information to integrate the new device into the system so that its capabilities are available via our programmer-visible interfaces.

Scheduling of computation and data motion become challenging in IMS systems, because a heterogeneous collection of computational elements will be embedded in memory system components of different nature and at multiple levels of the hierarchy, interconnected through a correspondingly heterogeneous fabric. This will result in an unconventionally hard-to-reconcile spectrum of computational capability, as well as data latency and bandwidth. Additional, dynamic inhomogeneity will result from transient spikes in the different parts of the system, and the corresponding ripple effects. Reliability-related inhomogeneity will also occur, from component failures over the lifetime of the system.

This points to the need for a high-level description of compute tasks and a flexible, dynamic mechanism for mapping those computations onto the available network, compute, and memory resources. This requires an execution paradigm in which there is a runtime (late/dynamic) binding of computation. In this way, computation can be moved closer to data, and vice versa, or mapped to other heterogeneous computing resources, whenever it is deemed beneficial by the runtime system, and the appropriate optimized executable module will be available. The runtime can also optimally engage just-in-time optimization where appropriate to incorporate details only available at runtime.

One key idea to our vision of the runtime system is to think of replication as a first-class primitive, so that multiple copies of hot data and computation can be kept at different levels of the system. Updates to data can either be lazy or eager, depending on the required application semantics for consistency. Our “execution plan” optimizer will identify opportunities to transform computations from the programmer's specification into an execution plan that is specialized for the specific hardware resources at hand as well as the current location(s) of the data.

Finally, users should be able to achieve their desired operating region on the Pareto-optimal frontier of latency,

throughput, precision, and energy-efficiency. For example, users may be willing to sacrifice some precision to achieve higher performance. The system software must therefore support interfaces that abstract heterogeneous hardware and runtime decisions, dynamic optimization capabilities to navigate the Pareto-optimal frontier and automatically select the best set of computational and data-motion operations to optimize performance, using mechanisms that minimize runtime intervention, so that after the initial setup, applications run at bare-metal speeds.

IV. PUTTING IT ALL TOGETHER

Having outlined the work to be done in each of the three research themes, we now briefly describe how they will work in unison to achieve the desired goals of leveraging the heterogeneously intelligent memory hierarchy without requiring extensive programmer effort. On the one hand, applications will use a declarative style, to express the desired computations on the associated data, while minimizing constraints on how the system achieves this. On the other hand, the hardware will expose abilities to place and refer to the data, properties that it will adhere to (integrity, security, consistency, etc.) for this data, in addition to mechanisms for placing, performing and orchestrating computations on the data. The job of the system software then is to provide the translation mechanisms from the declarative expressions in the application down to the new mechanisms exposed by the hardware.

As a simple example, consider a Big Data application expressed using a relational DSL, where the goal is to filter a number of data items from two tables/DataFrames, which are then merged. A SQL-like specification for implementing this would call for a “Select” operation followed by a “Join” operation. Let us say the hardware offers multiple memory chips, each with comparator logic embedded with the memory data arrays. In addition, the hardware may also provide vector processing units at the controllers interfacing to these memory chips. The translation process will start with the high-level declarations, and the associated execution plan, which would describe how the computations need to be performed. For execution on IMS units, the compiler would determine which functionality is the best fit for the semantics of the Select and Join and generate the appropriate machine code. The scheduler would then map these onto the different compute/storage elements, starting from the coarsest level (say

the data arrays of the memory chip as a whole), and recursively going down onto finer-grained elements (individual comparators within each array). The Select operator would thus be implemented using the fine-grain comparators very close to the data elements. The Join itself may be performed by the coarser-granularity vector units at the memory controllers, which get the subsetted and streamed data in a somewhat synchronized fashion, again orchestrated by the scheduler.

V. CONCLUSIONS

CRISP brings together a team of high-impact PIs with a track record of cross-stack research, with the goal of rethinking the entire system stack in order to tightly integrate processing and data, and ultimately operate as close as possible to the “computational speed of light” defined by the inherent dataflow of the algorithm and the raw throughput of the memory and storage data arrays. At the same time, these goals will be realized in a programming framework that is designed to maximize programmer productivity and software portability.

VI. ACKNOWLEDGMENTS

We wrote this paper based substantially on our proposal submission to DARPA/SRC’s JUMP program, to which all 20 PIs contributed: Luis Ceze (U. Washington), Jason Cong (UCLA), Kevin Eliceiri (U. Wisconsin), Samira Khan (UVA), Rob Knight (UCSD), Jing Li (U. Wisconsin), José Martínez (Cornell), Vijay Narayanan (Penn State), Jignesh Patel (U. Wisconsin), Tajana Rosing (UCSD), Anand Sivasubramaniam (Penn State), Kevin Skadron (UVA), Mircea Stan (UVA), Dmitri Strukov (UCSB), Steven Swanson (UCSD), Yuan Xie (UCSB), Zhiru Zhang (Cornell), Jishen Zhao (UCSD), Yuanyuan Zhou (UCSD), and Song-Chun Zhu (UCLA).

The center website can be found at <http://www.crisp-center.org>.

REFERENCES

- [1] International Technology Roadmap for Semiconductors, 2012.
- [2] Yuan Xie. “Modeling, Architecture, and Applications for Emerging Memory Technologies.” *IEEE Design Test of Computers*, 28(1):44–51, Jan. 2011.