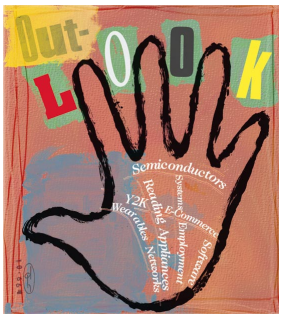


Computer Systems Research

The Pressure Is On

As applications become more demanding, computer systems research must not only redefine traditional roles but also unite diverse disciplines in a common goal: To make quantum leaps toward next-generation systems.

Krishna Kavi
University of Alabama at Huntsville

James C. Browne
University of Texas

Anand Tripathi
University of Minnesota

That computing and communication systems are becoming increasingly interdependent is evident in almost every aspect of society. Applications of these integrated systems are also spreading. As this trend continues, it will force the computing community not only to develop revolutionary systems but also to redefine "computer system" and the roles of traditional research disciplines, such as operating systems, architectures, compilers, languages, and networking. This awesome responsibility falls squarely on computer systems researchers—those who have traditionally developed the infrastructure on which applications are built.

In the Workshop on New Challenges and Directions for Systems Research, sponsored by the US National Science Foundation (http://www.cs.utexas.edu/users/new_directions), experts from a range of disciplines met to crystallize this new perspective and explore new paradigms for systems research and development. The emphasis was on finding quantum leaps, not evolutionary steps, to next-generation systems. The main product of the workshop was a list of research challenges, directions, and recommendations for the near and distant future. In this article, we look at three of those challenges, describe what is being done to meet them, and review remaining obstacles:

- *How to develop high-confidence systems with predictable properties at a predictable cost.* How do we use component-based design techniques to develop future applications, such as telemedicine or the design of new pharmaceuticals?
- *How to develop global-scale systems.* A key issue is how to specify and implement critical issues such as quality of service (QoS), security, and robustness. Application programming interfaces (APIs) for global-scale applications must be quite different from current APIs. For example, they must make it easier to accurately predict a system's properties from the properties of its components.
- *How to make architectures dynamic and adaptive.* Architecture that is liquid—reorganizes the execution engine as an application progresses—will be mandatory if we are to exploit the tremendous advances in hardware technologies. Dynamic architectures will blur the boundaries between hardware and software. The challenge is to look beyond reconfigurable logic and use runtime information across hardware and software boundaries. This will open doors to interoperation among applications using different execution paradigms, memory-consistency models, and network protocols.

As is characteristic of systems research topics in general, these challenges and their solutions are extremely interdependent. We also include "Outlook on" sidebars throughout the article that provide more specific evaluations of key challenges in architecture, networking, and software systems. The "Outlook on Applications" sidebar contains some of the applications identified in an interim report from the President's Information Technology Advisory Committee as possibilities for the next decade.

DEVELOPING ROBUST, PREDICTABLE SYSTEMS

Software systems are growing ever larger and more complex. End-to-end behavior often depends on tens or even hundreds of independently developed subsystems. At the same time, computer systems are assuming increasing responsibility for controlling and managing physical and logical processes—from telephone calls to safety-critical applications. Having designers attempt to engineer very complex systems without an adequate base of fundamental knowledge is proving disastrous: Software systems are failing at increasing rates. Products continue to be unacceptable because of software flaws. Both functional and performance deficiencies are becoming more commonplace.

In short, we cannot continue to use current system-development paradigms.

0018-9162/99/$10.00 © 1999 IEEE

Challenges

The techniques, methods, and processes for developing robust systems must meet four basic requirements. Finding an effective way to develop systems that meet these requirements is a challenge across all disciplines.

- *Composability*. We must be able to compose systems from components of multiple scales and from multiple semantic domains. As a start, we need effective models of composability, particularly ones that meet the requirements of dynamic structures, timing deadlines, and fault tolerance.
- *Dynamic, adaptable systems*. We must be able to build systems that can adapt to changing requirements and changing implementation environments on time scales from microseconds to years. This will let us more effectively manage resources, upgrade components, and tolerate failures.
- *Risk management*. Early in development, we must be able to reliably predict the system properties that will result from a given design. At present, we do not know a complex system's properties until the system is mostly developed. Because this lack of predictability presents an unacceptable risk, many potentially useful complex systems are never built.
- *Security and availability*. We must be able to build systems that are secure and available enough to make computers the primary means of critical personal and commercial transactions.

Directions and recommendations

Research programs must enhance our ability as a nation to develop processes that produce high-confidence software systems with predictable properties at predictable cost. One effective strategy for focusing research is to develop demonstration prototypes of Grand Challenge systems. An example is a transaction-oriented information management system based on Internet 2 that has known security and performance properties at design time and evolves gracefully over a specified set of changes in requirements and execution environment.

Intuitively it may seem odd to speak of "demonstration prototype" and "known security and performance properties" in one scenario. However, in this context, a *demonstration prototype* is a system that shows how appropriate conceptual foundations, appropriate methodologies, and adequate tool implementations can yield a product with predictable properties for predictable cost. This contrasts with, say, a rapid prototype, which focuses on the ability of the design to realize a given set of functions.

Research must also span disciplines. Narrow, single-discipline research programs will not help us attain the required knowledge and technology base. The problem does not fall solely to software engineering, for example. First of all, "engineering" implies that we are applying technologies we know to be effective. That is not the case here. Second, problems such as composability and risk management are common

Defining a Research Agenda

The goal of the Workshop on New Challenges and Directions for Systems Research (held July 31 to August 1, 1997, in St. Louis) was to identify significant new approaches to systems research—research into the infrastructure for developing computer and communication systems applications. A primary theme was integrated approaches to significant problems that span traditional research disciplines such as operating systems, architecture, compilers, networking, multimedia, real-time systems, system security, and system evaluation.

The research directions have five critical goals:

- Offer a quantum leap to the next generation of computer and communication systems.
- Examine the role of Grand Challenge problems in systems research. Grand Challenge problems are characterized by an extraordinary requirement for computing resources, the use of multidisciplinary approaches to solutions, and a potential to significantly affect how we use computing systems.
- Identify challenges and directions for building synergistic relationships among historically disjoint computing systems research areas.
- Create new paradigms for computer and communication systems that address Grand Challenge problems.
- Define infrastructure requirements that will enable research based on synergistic approaches to Grand Challenge problems.

The results of the workshop are documented at (http://www.cs.utexas.edu/users/new_directions). Although the workshop was held over a year ago, a recent interim report from the President's Information Technology Advisory Committee (http://www.hpcc.gov/ac/interim/) identified similar requirements in areas that overlap systems research, including the need for component-based software development and a scalable information infrastructure.

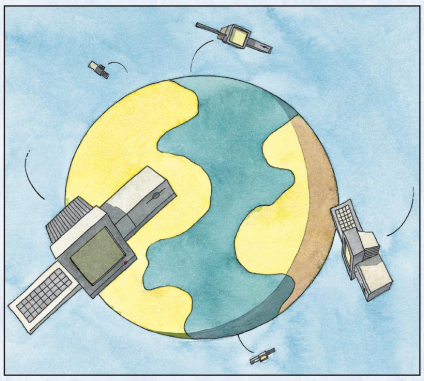
The research recommendations that came from the workshop will directly address these critical areas of concern. For example, one of the workshop's recommendations is to adopt a multidisciplinary approach in defining new application programming interfaces. This will aid in the component-based development of software systems with predictable behavior. Other recommendations include developing models to predict the performance of dynamic and adaptive systems, and establishing a research infrastructure to support research on global-scale applications. Both these recommendations address the PITAC report's requirement to develop a scalable information infrastructure.

Outlook on Applications: Reshaping Relationships Around the World

The extraordinary rise in power, connectivity, content, and flexibility of information technologies is so fundamental that it is dramatically reshaping relationships among people and organizations. It is also quickly transforming our processes of discovery, learning, exploration, cooperation, and communication.

We can now study vastly more complex systems and begin to rapidly advance our understanding of learning and intelligent behavior in living and engineered systems.

Our current challenge is to realize the full potential of these new resources and institutional transformations. The recent President's Information Technology Advisory Committee Interim Report (http://www.hpcc.gov/ac/interim) lists a range of application visions:



One billion people worldwide can access the Internet simultaneously.

- One billion people worldwide can access the Internet simultaneously and engage in real-time electronic meetings, download the daily news, conduct financial transactions, or talk to friends and relatives around the world.
- Any individual can participate in online education programs regardless of geographic location, age, physical limitation, or personal schedule.
- Telemedicine applications are commonplace. Specialists use videoconferencing and telesensing to interview and even examine patients who may be hundreds of miles away.
- Complex products and structures can be designed via computer simulations that accurately represent the physical properties of the system being built. Designers, manufacturers, builders, suppliers, and users participate in the design.
- Research is conducted in virtual laboratories, in which scientists and engineers routinely perform their work without regard to physical location. They interact with colleagues, access instrumentation, share data and computational resources, and access information in digital libraries.
- Government services and information are easily accessible to citizens, regardless of their physical location, level of computer literacy, or physical capacity.

In addition to these visions is an often ignored application: the intelligent integration of home computing and entertainment devices. An example is a home where the household electronics recognize the residents by their profiles (TV viewing habits, room-temperature settings, and so on) and automatically alter settings to meet the residents' needs.

wherever effective processes are critical to system development, regardless of discipline.

Research must also be based on sound formal models of systems and development processes. Researchers outside the traditional software engineering and security communities must become involved in developing new programming paradigms, compilers, software processes, and tools that are amenable to formal and rigorous analysis. Effective research programs will couple formal methods with knowledge not only from experts in many disciplines, but also from tool and system developers. The focus must be on scalability and relationships across domains. We can expect major advances from such a program. We could, for example, design highly secure and reliable systems, since we would be able to verify security and reliability by integrating analyses and testing.

DEVELOPING GLOBAL-SCALE SYSTEMS

The development of global-scale systems is a natural outgrowth of the many applications that integrate computers and communications. QoS requirements, which relate to both timing requirements and richness of service (such as video quality in telepresence) will be critical in many applications. Resource-management functions must also broaden: For an application in wide-area networks, such functions must span several administrative organizations. Finally, there must be support for mobile users and resources.

Challenges

An important challenge in this area is how to develop mechanisms that let resources be accessed uniformly and seamlessly across LANs, intranets, and the Internet. The sidebar "Outlook on Networks" describes some of the key issues in this challenge, including the development of a global uniform naming scheme and name-resolution infrastructure. To build such a facility, researchers must address scalability, robustness, and security, as well as more traditional issues such as replication, caching, and fault tolerance. To satisfy the mobility requirement, naming schemes must be location independent so that clients can access mobile resources without identifying their locations. Such mechanisms would apply to all types of resources.

Another challenge is how to manage resources and cope with end-to-end latencies. Many future network-centric applications, such as ubiquitous, high-quality telepresence, will impose stringent QoS requirements. To meet such demands, resource management must be dynamic and scalable: It must meet real-time constraints and deal with different administrative authorities.

Resource management strategies for global systems must also address variations in workload and resources, how to deal with failures, and how to tune the system given its runtime behavior. Another challenge is how to cope with increased loads. In some

Outlook on Networks: Experimentation to Harness the Power of the Internet

The current infrastructure of wide-area network research must change if we are to develop applications that exploit all the communication power of the Internet. Current WANs do not deal with congestion effectively, offer poor support for quality of service (QoS), and have low reliability. There is also no clear and rigorous definition of the semantics of shared state.

The community needs simple programming models with clear and powerful semantics and abstractions for developing Internet-based applications with adequate support for QoS, availability, fault tolerance, security, and protection requirements.[1]

WAN research must also address design scalability, how to cope with diversity and change, and performance, reliability, and security. Major challenges are how to develop network control software and self-configuring network components, how to better use hardware technology, how to simplify the evolution of networks, and how to directly support distributed applications in a way that will maximize network resources and capabilities. To meet these challenges, there must be open platforms for systems research so that experimenters can configure both hardware and software components quickly and easily.[2]

Naming

One prerequisite for these goals is to develop more effective naming in global-scale systems. The primary function of a name service is to map a given resource name to its current address. Recognizing the brittleness of URL-based names[3] when mobile resources are involved, an Internet Engineering Task Force working group is exploring the use of Uniform Resource Names (URNs).[4] URNs will be globally unique and provide resource naming regardless of location. They will also be persistent and scalable (in the context of the resources' life span), transcribable, and human-readable. These functional requirements will become essential if we are to seamlessly integrate and access new resources and applications on global information networks.

Both URL and URN belong to a broader class of names, Uniform Resource Identifiers (URIs).[5] Currently, DNS (domain name system) names are primarily location dependent, and the DNS service by itself cannot support URNs because it would have to maintain a potentially very large number. In all other respects, however, the DNS service is the best candidate foundation for building a URN service. One effort to use DNS to implement a URN service is a NAPTR (short for naming authority pointer), which is a new type of DNS resource record. A NAPTR provides rules for mapping parts of URIs to domain names. By changing the mapping rules, designers can change the host that is contacted to resolve a URI. The result will be a more graceful handling of URLs over long periods.

NAPTRs form the foundation for one of potentially many new proposals for handling URNs.[6] An IETF working group has identified a general framework that will let different URN systems work together, thus making it easier to seamlessly integrate applications based on different URN schemes.

Mobile agents can migrate autonomously to perform computations on behalf of their owners.

Mobile agents

Another issue, and one very much a part of current agendas, is the creation and use of mobile agents—active objects that can migrate autonomously from node to node to perform computations on behalf of their owners, either users or programs.[7] Traditionally, distributed applications have relied on the client-server paradigm in which client and server processes communicate either through message passing or remote procedure calls. The RPC model is usually synchronous—the client suspends itself after sending a request to the server, waiting for the results of the call.

The mobile-agent paradigm has three main advantages over RPC and message passing.[8] First, moving processing functions close to where the information is stored reduces the amount of communication between client and server. With mobile agents, the user sends the agent only once (with its initial parameters) to the remote server. It then interacts with the server locally without using the network; after it completes its task, it migrates back to its home site with the results. Thus, instead of doing a simple keyword-based Web search, the user would send an agent to perform a more intelligent search and filtering, which he has customized to satisfy his current needs.

Second, the mobile agent paradigm increases the degree of asynchrony between client and server. While its agents execute on various servers, the client need not remain connected to the network. This feature is especially useful for mobile computers, which are often switched off to save on power and network-connection charges.

Finally, agents potentially outperform RPC and message passing in real-time control applications. If the application uses RPCs to control a device, it may be difficult (if not impossible) to guarantee real-time deadlines because of variations in network communication delays. With agents, the application can send an agent to the device and control the device locally, which means better predictability. Agents can also monitor performance of computing nodes and network bandwidth and migrate to nodes that are more likely to complete tasks on time.

References

1. B. Liskov, keynote speech at *Workshop on New Challenges and Directions for Future Research*, 1997, http://www.cs.utexas.edu/users/new_directions.
2. J. Turner, keynote speech at *Workshop on New Challenges and Directions for Future Research*, 1997, http://www.cs.utexas.edu/users/new_directions.
3. T. Berners-Lee, L. Masinter, and M. McCahill, "RFC 1738: Uniform Resource Locators (URLs)," Dec. 1994; http://www.isi.edu/publications/.
4. K. Sollins, "Architectural Principles of Uniform Resource Name Resolution," Sept. 1997; http://www.isi.edu/publications/.
5. T. Berners-Lee, "RFC 1630: Uniform Resource Identifiers in WWW," June 1994; http://www.isi.edu/publications/.
6. R. Daniel, "RFC 2168: Resolution of Uniform Resource Identifiers using the Domain Name System," June 1997; http://www.isi.edu/publications/.
7. J. White, "Mobile Agents," General Magic Inc., Oct. 1995; http://www.genmagic.com/Telescript/.
8. D. Chess et al., "Itinerant Agents for Mobile Computing," *IEEE Personal Comm.*, Oct. 1995, pp. 34-59.

cases, systems may need to add nodes and hardware on the fly to meet application requirements. In other cases, applications must be notified of a resource loss so that they can modify their behavior.

A third major challenge is the creation of programming abstractions that make it easier to program and use large distributed systems. New techniques, languages, and paradigms have evolved in response to such needs—but have brought their own set of challenges. Perhaps the most promising new paradigm is the use of mobile agents. The sidebar "Outlook on Networks" describes the characteristics of this paradigm.

Electronic commerce and agent-based distributed computing have increased the importance of security and assurance in network-based systems. A host node must permit mobile agents originating from any site to execute in an open distributed system and still be able to protect itself from attacks by malicious agents. Similarly, agents themselves may need to be protected from the hosts they visit.[1]

Directions and recommendations

There are several key directions in this area.

Naming. To address the challenge of providing uniform access to resources, researchers are looking to replace URLs, which are location dependent, with Uniform Resource Names (URNs). The sidebar "Outlook on Networks" describes this new naming scheme. The goals of this effort are to design naming schemes and name-resolution protocols and develop an organization for the name service's database that uses partitioning, replication, and caching.

Application programming interfaces. APIs should make it possible to specify resource, latency, and QoS requirements. These requirements could then be translated to suitable resource-allocation and scheduling decisions (at the system's lower levels) or to negotiations for resources and QoS services. Researchers must develop instrumentation for monitoring and control, as well as financial models for accounting. Monitoring and adapting to changing conditions will be critical to the success of global systems. APIs must permit the exchange of information on many levels and aid in negotiation and adaptation.

Research must also cut across system levels (network, operating system, runtime system) and application disciplines in defining APIs that permit resource-management policies and the specification of QoS, security, and reliability requirements. Cross-disciplinary research must also be the watchword in designing systems that adapt to changing resources and application requirements.

Data and service replication. Replication is both a

Three Decades of Computing Systems Research: Characteristics and Key Developments.

	1970s	1980	1985
Chip densities	< 1,000	< 100,000	< 500,000
Clock rates	1 MHz	10 MHz	<50 MHz
Main memory	A few Kbytes	< 1 Mbyte	< 32 Mbytes
Architecture	Bit parallel (4 bits) Microprogrammed Sequential execution	Bit parallel (8 bits) Microprogrammed and pipelined	Bit parallel (16 bits) RISC, pipelined Cache memories
Compilers	C, Pascal p4 Dataflow analysis	GNU retargetable compilers Vectorization Programming environments	ML,Tcl/Tk Interprocedural analysis
Operating systems	Unix Time sharing	Remote procedure calls (RPC) Distributed file systems	Microkernels (Mach, Chorus) X Windows Name file server
Networking	ARPAnet FTP, Telenet Ethernet and LANs	Internet TCP/IP	Domain name system Network management protocol

benefit and a disadvantage. On the one hand, it increases both robustness and scalability; on the other, it complicates the management of distributed state sharing, security, and fault tolerance. Trust relationships and security policies in global systems can be quite complex, and security breaches may pose high risks. Risk management and intrusion detection and prevention will become key concerns. The major focus in security should be on end-system security, not just on middleware or application-level security. Language designs and compiler support must address security requirements. AI techniques for pattern recognition can be used to detect unauthorized access. An important direction in this area is the creation of languages and interfaces that define security policies and a means to analyze those policies.

Research infrastructure. We clearly need an infrastructure to support research in global-scale applications. The infrastructure must integrate diverse research domains, each of which now has a distinct approach and policy. At the network-resource-management level, research often requires dedicated research facilities that experimenters may alter or reconfigure as desired. On the other hand, research in higher level abstractions and services requires a stable underlying network. Some experiments require installing software components on extreme numbers of nodes and subsequent access to those components.

DYNAMIC AND ADAPTIVE ARCHITECTURES

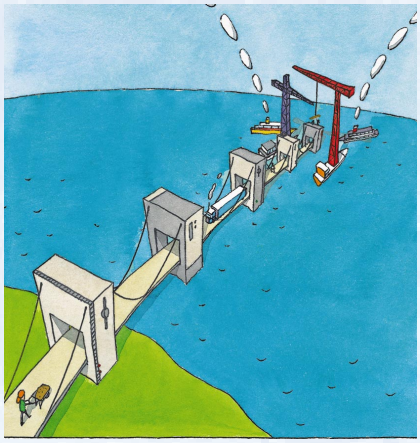
With the possibility of billion transistor chips that can run at gigahertz clock speeds, we are approaching a critical technological threshold that will revolutionize the organization and abstractions in current machines and systems. These changes present exciting challenges across all areas of systems architecture, and no one is quite sure where they will lead. Among the alternatives for processor architecture are trace processors, multithreaded processors, multiscalars, multiple processors on a chip, and superspeculative executions of instructions. The sidebar "Outlook on Systems Architecture" describes the characteristics of these choices. Future systems will rely on the pervasive use of *virtual machine* architecture, in which the instruction-level architecture is made "liquid" (instruction set, caches, and other data paths can be changed on the fly as an application progresses). Researchers must reexamine the traditional definitions of architecture and system design as technological advances open the door to designing much higher level systems.

Many forms of dynamic optimization are emerging at different design levels. In the past, these efforts

1990	1995	2000
1 million	10 million	1 billion
100 MHz	500 MHz	1 GHz
100 Mbytes	Gigabyte	100 Gbytes
Instruction parallel 32-bit architecture Branch prediction Out-of-order execution Multimedia	Instruction parallel 32- and 64-bit architecture Superscalar Speculative execution	Thread parallel 64-bit architecture Superscalar, trace processors, multithreaded architecture Multiscalars
C++, HPF/Fortran 90 IDL compilers Visual programming languages	Java/JVM, mobile code proof-carrying code	Web programming Robust compiler infrastructure
Distributed objects, CORBA	Mobile code Web security	Global scale OS
ATMs HTTP, Mbone, MIME	Web IPv6 Internet 2 Active networks	Next-Generation Internet Billion-node World Wide Web

Outlook on Systems Architecture: How to Use a Billion Transistors

The challenge is no longer how to produce a billion-transistor chip, but how to best use devices that contain such chips. The immediately obvious solution is to add more of something, such as on-chip (L1) cache memory, pipelines (providing a higher degree of superscalars), registers, hardware context, prefetch buffers, or on-chip DRAM. Less obvious design alternatives may better satisfy an important set of requirements, however: a high degree of instruction-level parallelism; large or multiple sets of register files; the ability to predict both conditional branches and instruction-generated data values, and the ability to overcome memory latencies.



The challenge is no longer how to produce a billion-transistor chip, but how to best use devices that contain such chips.

Trace processors

Like many alternatives, trace processors[1] attempt to exploit instruction parallelism by having multiple processing elements—each of which resembles a modern superscalar unit—on a single chip. Programs are expanded into traces and stored in trace caches. Traces contain a set of instructions along with predicted conditional branches. Multiple instructions from these traces are issued to the various processing elements. Data values may be predicted and inserted to eliminate data fetches.

Unlike conventional systems, trace processors will be able to predict and fetch the next set of traces, which may involve multiple branch predictions, and forward results to multiple processing elements. Physical registers comprising local and global registers will permit fast access without numerous ports. To exploit the parallelism provided by trace processors, designers will need large instruction windows so that they can construct traces with a high degree of parallelism.

Multiscalars

Multiscalar systems[2] are similar to trace processors in that they contain multiple processing elements on a chip. Unlike trace processors, however, each processing element executes a fine-grained task that is typically a basic block or a loop iteration. Because many processing elements reside on the same chip, communication among tasks is very efficient.

have focused to a large extent on instruction-processing problems, such as branch prediction. Dynamic optimization can also take the form of runtime mechanisms that observe execution behavior, predict future behavior, and use predictions to enhance performance, typically through speculation. Generalizations of these techniques will apply to other areas of architecture, including memory and I/O subsystems.

Today's caches use only limited aspects of memory behavior and speculate in a very limited way on the data dependencies in memory. Future memory systems should permit memory renaming, connect stores and loads, and increase the degree of speculation. More aggressive approaches may examine values being accessed from memory and take speculative actions. IRAM (intelligent RAM) could pave the way for such intelligent memories.

Challenges

There are three major challenges in this area.

Application programming interfaces. APIs must allow the exchange of information across software and hardware boundaries. Research should push dynamic optimization techniques down into lower level design by allowing the hardware itself to be reconfigured on the basis of runtime observations. Likewise, we should push runtime information up across the hardware-software boundary by providing feedback to the application so that it can change the way it uses the machine. For example, a storage subsystem might collect information on the virtual memory's performance. The information in turn could cause the running program to reorganize data to improve performance.

Some existing systems use static profiling information[2] and dynamic profiling[3] to permit the exchange of information between compiler and runtime systems. However, APIs must go further, to allow information exchange not only between compilers and runtime systems but also between application code and compilers and between runtime systems and instruction-level architectures, memory subsystems, and instruction-issue hardware.

To support programmable hardware and network devices, APIs must provide the compiler with a view that includes not only the CPU, but also the storage and networking subsystems. Application designers could then load object code into these subsystems to coordinate and optimize their performance for that application. For example, a storage subsystem could offer several consistency models, and storage and network subsystems could have programmable tracing hardware. We could then load object code into these subsystems to coordinate and optimize their performance. To support reconfigurability and liquid architectures, a compiler might negotiate an instruction set to optimize a particular program's performance, or system designers might use field programmable gate arrays (FPGAs) to customize the instruction set.

Multiple-layer system evaluation. Sophisticated mechanisms that enhance a technology's performance often make it harder to predict the complete system's performance. An apparently insignificant program change frequently causes a large, unexpected performance vari-

Simultaneous multithreading

Multithreading aims to hide long latencies when accessing slower memories either by switching between multiple execution threads or by interleaving instructions from multiple threads. Simultaneous multithreading (SMT)[3] interleaves blocks of instructions from multiple threads. Because it combines parallelism both between threads (interthread) and within a single thread (intrathread), it can issue many instructions simultaneously to different functional units (or pipelines) in superscalar architectures. SMT architectures, like any multithreaded architecture, perform better in multiple hardware contexts (a separate register set, program counter, and status registers). Without multiple contexts, when a context must change from one thread to another, the processor must save the registers belonging to one thread and load the registers belonging to another—a procedure that can be prohibitively expensive.

Superspeculative architectures

Modern processors use aggressive branch-prediction techniques to eliminate pipeline stalls from conditional branch instructions. Superspeculative architectures[4] go beyond control speculation to use data speculation also. The basic concept is that most producer instructions generate highly predictable values, so consumer instructions can speculate their source operands and begin executing even before producer instructions complete their execution.

To mitigate the negative effects of slower memories, IRAM (intelligent RAM) designs[4] move at least some processing to DRAM chips. Typically, such chips limit their processing capability to avoid using large silicon areas for processor logic. One recent design demonstrated IRAM chips with a 1-Mbit DRAM and a CPU that performs only integer arithmetic on 8-, 16-, or 32-bit values.[5] IRAM chips can be used in conjunction with conventional CPU chips to enable single-instruction, multiple-data or vector processing for multimedia applications. Designers may also be able to delegate memory management and garbage collection to the IRAM chips and raise the data abstraction visible to CPUs.

References

1. J. Smith and S. Vajapeyam, "Trace Processing: Moving to Fourth Generation Microarchitecture," *Computer*, Sept. 1997, pp. 68-74.
2. G. Sohi, S. Breach, and T. Vijayakumar, "Multiscalar Processors," *Proc. 22nd Int'l Symp. Computer Architecture*, ACM Press, New York, 1995, pp. 414-425.
3. S. Eggers et al., "Simultaneous Multithreading: A Platform for Next-Generation Processors," *IEEE Micro*, Sept.-Oct. 1997, pp. 12-19.
4. M. Lipasti and J. Shen, "Superspeculative Microarchitectures for Beyond AD 2000," *Computer*, Sept. 1997, pp. 59-66.
5. Y. Nunomura, T. Shimizu, and O. Tomisawa, "M32R/D: Integrated DRAM and Microprocessor," *IEEE Micro*, Nov.-Dec. 1997, pp. 40-48.

ation. This uncertainty makes it difficult to design many subsystems, including compilers and operating systems. Researchers must investigate techniques that will qualitatively improve our ability to predict system performance. Models for performance prediction should be portable across architectures. There seems to be an important tension between making systems more adaptive and making them more predictable, although these goals need not be mutually exclusive.

Design of dynamic, adaptive, and scalable systems. Computer applications will continue to broaden in both scope and scale. Dynamic optimizations and adaptations should consider both dimensions. At one extreme are million-processor systems; at the other are very small scale systems, such as processing devices in appliances and disposable systems. Design principles for dynamic and adaptive architectures must consider the entire spectrum.

Directions and recommendations

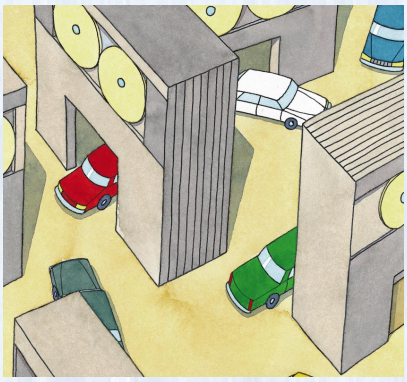
There are three major directions in this area. Some of these, like APIs and a research infrastructure, are recurring themes, but we include them to show how a particular research direction will address more than one challenge.

Application programming interfaces. New principles are needed to construct APIs. The API should not only permit the exchange of information across hardware and software layers, but also emphasize composability. At one level, we should be able to specify the desired behavior of each lower level subsystem. For example, a compiler should be able not only to program the CPU and cache parameters directly, but also to specify a desired memory-consistency model or to program network components and the I/O subsystem. Likewise, we should be able to specify the desired behavior of any higher level subsystem.

Research infrastructure. Researchers should validate these API design principles experimentally. This requires defining evaluation metrics, creating ways to implement the APIs, and designing experiments to evaluate the implementations. To meet these goals, an infrastructure must be created that will let researchers experimentally evaluate architectural alternatives and compiler and runtime interactions with the hardware architecture. To design APIs and frameworks, we will need approaches that cross many disciplines, including hardware architectures, compilers, runtime systems, networking, and operating systems. With the resulting frameworks, designers will be able to create composable, adaptable systems that fall anywhere on the very large to very small scale spectrum.

Predictive models and methods. We will need models and analytical methods that accurately predict the performance of dynamic and adaptive systems. Performance in this context implies not only execution speed, but also properties such as QoS, fault tolerance, availability, survivability, and security. The models and analytical methods should reflect the contribution of each change to overall system performance. A necessary first step to creating models and methods is to identify the interrelationships among system components and reflect them in the system models. The relationships may be based on empirical observations. The

Outlook on Software Systems: Facing the Challenge of Component Reuse and Development



Designers of many existing complex products—automobiles, airplanes, and even computers—compose off-the-shelf components from different manufacturers into complete systems.

The key research challenge for software system development is to develop a broadly applicable technology for component reuse and compositional development. Designers of many existing complex products—automobiles, airplanes, and even computers—compose off-the-shelf components from different manufacturers into complete systems. In compositional development, the developer defines specifications for both the functionality and interface of the desired components and then builds the product from existing components that meet those specifications. Component manufacturers in turn develop components with frequently requested specifications.

The software industry has not yet succeeded in adopting this concept on any broad scale. Large "components," operating systems, database systems, and so on, are widely reused. A few component libraries at the code level are widely and successfully used, but these are mostly in specialized domains such as graphical interfaces (Visual Basic, Visual C++, JavaBeans) and may not be suitable for large-scale systems. Systems that define interfaces (object request brokers), such as CORBA, COM/DCOM and their competitors, provide a framework for interfacing independently developed components.

But too many significant software systems are developed largely without the use of existing components. Efforts to build significant systems using commodity components have by and large been unsuccessful. This argues first that current approaches are inadequate and second that they are inadequate because the fundamental conceptual foundations for compositional development are inadequate. Indeed, the lack of effective methods for compositional development is a major cause of the so-called software crisis. More effective compositional development processes would produce more robust and efficient software systems and lower the cost and effort to develop them.

Object-oriented design methods provide some foundation for reuse at the code level.[1] Abstract type-oriented languages (such as Ada) or object-oriented languages (such as C++) provide features that let designers compose new types or objects from already defined ones. Object-oriented languages also commonly provide mechanisms that let designers extend the template interface. These features and mechanisms have been useful but do not appear to extend to levels of abstraction above procedural code. Composition must take place at analysis and design levels. Design patterns[2] and design databases represent needed research directions.

A small but growing community of researchers is developing concepts and methods for component reuse and compositional development. Among the major research directions are software frameworks and architectures,[3] which are toolkits for developing families of software systems. Research is also being done to develop interface and high-level specification languages that support composition from components, particularly those that are domain-specific. The number of conferences and workshops with a major focus on reuse and compositional development—the IEEE International Conference on Software Reuse, the Workshop on Institutionalization of Software Reuse, the IEEE International Conference on Software Engineering, ACM's Conference on Object-Oriented Systems, Languages, and Applications (OOPSLA), and the European Conference on Object Oriented Programming (ECOOP)—indicates the level of interest in research on compositional development and component reuse.

References

1. L.P. Deutsch, "Design Reuse and Frameworks in the Smalltalk-80 System," in *Software Reusability*, Vol. II, T.J. Biggerstaff and A Perlis, eds., ACM Press, New York, 1989.
2. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1994.
3. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, Englewood Cliffs, N.J., 1996.

models should account for the behaviors of both the application and its execution environment.

The next step should be to experimentally validate the models and analyses. A cross-disciplinary approach will help in evaluating systems as a whole and in identifying how each system component contributes to performance.

Global-scale systems. Current system design methods are suitable only for specific classes of systems (reconfigurable systems that use FPGAs or ASICs or shared-memory symmetric multiprocessors, for example). Research must extend these techniques to develop design approaches for both large- and small-scale systems. Integral to design methods is research on how to systematically map application behaviors onto execution environment behaviors. Research on the design of APIs and metrics should significantly influence design methods.

The three challenges we have examined are representative of the numerous challenges and directions that await the computing community. The workshop contains position papers and group findings that highlight many other challenges and directions in other areas of systems research. Also, some of the recommendations we have presented, although specific to systems research, are in line with a recent interim report from the President's Information Technology Advisory Committee (http://www.hpcc.gov/ac/interim/), which covers all areas of technology.

If we could distill one major thought from the workshop results, it would be this: *Systems research is facing an unprecedented challenge.* Systems developers are facing a major discontinuity in the scale and nature of both applications and execution environments. Applications are changing from transforming data to directly interacting with humans: They will use hardware and data that span wide-area, even global, networks of resources and involve interactions among users as well. Even the architecture of individual processors is uncertain.

Despite this uncertainty, we see three clear first steps to addressing the identified challenges.

Define a new paradigm for systems research. The traditional role of systems research is giving way to interdependent programs that focus on a range of disciplines. Defining an appropriate model for cross-disciplinary research that spans systems, systems integration, applications, and hardware architectures should be a high priority for the research community and funding agencies.

Attack problems common to all system development. Composability, risk management based on early evaluation, and adaptability are problems that plague the development of all systems and represent critical barriers to developing dynamic global-scale systems. Emphasizing innovative approaches to these common problems will benefit all subdisciplines of systems research and make it easier to develop applications.

Build a research infrastructure. Experimental research on large-scale distributed systems is almost intractable because of the cost and effort to deploy meaningful experiments. A research program to define and instantiate a family of infrastructures is a prerequisite to experimental research on large-scale distributed systems. ❖

References

1. W. Farmer, J. Guttman, and V. Swarup, "Security for Mobile Agents: Issues and Requirements," *Proc. 19th Nat'l Information Sys. Security Conf.*, National Inst. of Standards and Technology, Baltimore, Oct. 1996, pp. 591-597.
2. A. Rawsthorne and J. Souloglou, "Dynamite: A Framework for Dynamic Retargetable Binary Translations," Tech. Report UMCS 97-3-2, Univ. of Manchester, Manchester, UK, 1997.
3. X. Zhang et al., "System Support for Automatic Profiling and Optimization," *Proc. Symp. Operating Systems Principles*, ACM Press, New York, 1997, pp. 15-26.

Acknowledgments

The 1997 Workshop on New Challenges and Directions for Systems Research was supported in part by NSF grant CCR-9714873 and hosted by the University of Washington, St. Louis. We thank all the workshop participants and keynote speakers for their thought-provoking discussions. Keynote speakers were Jim Smith (University of Wisconsin), Butler Lampson (Microsoft Research), Barbara Liskov (Massachusetts Institute of Technology), and Jonathan Turner (University of Washington, St. Louis). Other participants were Tom Anderson, Frank Anger, B.R. Badrinath, Brian Bershad, Robert Blumoffe, James Browne, Pei Cao, John Carter, J. Chapin, Sid Chatterjee, Andrew Chien, Dave Culler, Ron Cytron, Mike Dahlin, Jack Dennis, Mike Foster, Helen Gill, Ken Goldman, Mark Hill, Susan Horwitz, Frans Kaashoek, Krishna Kavi, Jay Lapreau, Gary Nutt, David Oran, Calton Pu, Guru Purulkar, R. Rajkumar, Peter Reiher, Mendel Rosenblum, Vivek Sarkar, Mike Scott, Marc Shapiro, J.P. Singh, Peter Steenkiste, Mike Smith, Tatsuya Suda, Anand Tripathi, and Willy Zwaenepoel.

*Krishna Kavi is a professor and eminent scholar in the ECE department at the University of Alabama at Huntsville. His research interests are in computer systems architecture, performance measurement, and the formal modeling of concurrent systems. He edited two IEEE CS Press tutorials (*Real Time Systems: Abstractions, Languages, and Design Methodologies, *1992, and* Scheduling and Load-Balancing, *1995). He was an associate editor of* IEEE Transactions on Computers *from 1993 to 1997. He received a PhD in computer science and engineering from Southern Methodist University. He is a senior member of the IEEE and a member of the ACM.*

James C. Browne is a regents chair professor of computer science, and a professor of physics at the University of Texas. His research interests are in parallel processing and software systems, with a focus on programming, modeling, and operating systems. He received a PhD in chemistry from the University of Texas at Austin. He is a fellow of the ACM, British Computer Society, American Association for the Advancement of Science, and American Physical Society, and is a member of the IEEE Computer Society.

Anand Tripathi is an associate professor of computer science at the University of Minnesota. His research interests include distributed systems, fault-tolerant computing, and object-oriented programming. He received a PhD in electrical engineering from the University of Texas at Austin and is a member of the IEEE Computer Society and the ACM.

Contact Kavi at kavi@ece.uah.edu; Browne at browne@cs.utexas.edu; and Tripathi at tripathi@cs.umn.edu.