On the Integration of Computer Architecture and Parallel Programming Tools into Computer Curricula*

José A. B. Fortes†, Nirav H. Kapadia†, Rudolf Eigenmann†,
Renato J. Figueiredo†, Valerie Taylor††, Alok Choudhary††
Luis Vidal‡ and Jan-Jo Chen‡

†School of ECE, Purdue University
††Department of EECS, Northwestern University

‡Dept. of Mathematics, Statistics and Computer Science
Chicago State University

Abstract

Tools for computer architecture design and parallel programming have become essential to practicing computer architects and software developers in industry. There is significant demand for designers who can develop, use and modify them. More importantly, design and simulation tools capture fundamental engineering concepts that should be mastered by computer professionals. This paper describes an approach to the integration of tools for computer architecture simulation, performance prediction, program optimization and application characterization into computer science and engineering curricula. The approach is based on a unique, operational distributed infrastructure - the Purdue University Network Computing Hubs (PUNCH). The PUNCH infrastructure lets users with different computing platforms run a broad range of tools via standard WWW browsers. PUNCH also allows universities to share course-development efforts, tool expertise and resources while preserving the appearance of a centralized point of access to all tools installed in PUNCH. The expected outcomes of this project are: (1) generations of computer architecture and software designers who are capable of understanding and using state-of-the-art, industrially-relevant tools and (2) an infrastructure model for inter-university cooperation in curriculum improvement and resource sharing that can be extended/replicated across other institutions interested in incorporating computer design tools in their curricula.

I. Introduction

Computer-based design, simulation and evaluation tools are essential to practicing computer engineers and scientists involved in building, programming or evaluating computer systems. This is a consequence of the continued growth in the complexity of these systems and their component chips, which are expected to contain up to a billion transistors by the time the

*This work was partially funded by the National Science Foundation under grant EIA-9872516, and by an academic reinvestment grant from Purdue University.

computer engineering class of 1999 graduates. More than ever, not only do computer engineers need to know novel and advanced architecture concepts, but they must also master the tools needed to develop and validate them. To quote [6], "Simulation and tracing tools help in the analysis, design and tuning of both hardware and software systems. Simulators can execute code for hardware that does not yet exist, can provide access to internal state that may be invisible on real hardware, can give deterministic execution in the face of races, and can produce "stress test" situations that are hard to produce in real hardware. Tracing tools can provide detailed information about the behavior of a program [...]. That, in turn, provides feedback that is used to improve the design and implementation of everything from compilers to applications." It is the premise of this paper that it is essential for computer engineers of the next millennium to understand computer-based tool capabilities, their applicability to design problems, the setup of simulation/evaluation experiments and fundamental techniques for developing their own tools.

This paper describes an approach to the integration of computer architecture tools into computer curricula that does not require new courses to achieve the stated goals. Instead it significantly enriches existing courses with content and experiments not present or possible in existing curricula and their supporting infrastructures. It enables instructors and students to "experience" examples and homework assignments that show how appropriate tools can be used for design and analysis in the context of each topic. The choice, order and depth of coverage of tools are customizable to fit the needs and preferences of each course and instructor.

Three institutions with programs in Computer Science and/or Computer Engineering are involved in a project recently funded by the National Science Foundation to implement and evaluate the approach described in this paper: Chicago State University, Northwestern University and Purdue University. The implementation is based on a distributed computing infrastructure - the Purdue University Network Computing Hubs [12] (PUNCH). The infrastructure allows installed tools to be readily used without requiring time-consuming search, installation, custom set-ups and other support activities. In addition, immediate experimentation and access to documentation enable users to quickly assess the suitability of the tools for the desired design activity.

This paper is organized as follows. Section II lists some of the tools of interest to computer professionals and the challenges faced in their incorporation into existing curricula. Section III discusses ongoing activities at the author's institutions that are aimed at addressing those challenges. Section IV describes succinctly the WWW-computing infrastructure that enables the approach being pursued by the three institutions (as well as their collaborative efforts). Section V presents concluding remarks.

II. Tools for computer architecture design and parallel programming

Tools for the design, evaluation and programming of high-performance processors and their components have been and continue to be developed in many universities and industrial laboratories. Tables 1 and 3 show a representative subset of such tools for designing and programming uniprocessors and multiprocessors. Tools for other computer engineering areas are equally important (e.g. performance programming, interconnection network design and I/O modeling). Together, these tools and their underlying implementation techniques comprise the body of knowledge that the authors are integrating into mainstream computer education curricula.

Tool	Purpose	Host Arch.	Input	Output	Related course topics
ATOM† [26]	Uniprocessor Simulation, Tool Building	DEC Alpha	C source (analyzer) object (application)	Text, program-dependent	**Instruction set, pipeline, memory hierarchy**
AUGMINT [21]	x86 Multiprocessor Simulation	x86 Solaris, NT,Linux	C with ANL macros	Text	CISC instr. set, distrib. shared mem., cache coherence
Chameleon [18]	VLIW compiler and simulator	Power, PowerPC	C	VLIW assembly, text traces	**VLIW, superscalar, branch prediction, ILP**
DLXTools: SPIM, SuperDLX, etc [10]	Uniprocessor Simulation, Education	MIPS,Unix, MS-Windows	C, DLX assembly, ASCII traces	Text and/or GUI	**Instr. set, pipeline, mem. hierarchy**, vector, **superscalar, ILP**
HPAM_Sim [5]	Heterogeneous Multiprocessor Simulation	Solaris	C/Fortran with message-passing calls	Statistics, Text	Message-passing, distributed memory, heterogeneous computing
MINT [27]	Multiprocessor Simulation	Sparc, MIPS	C source (analyzer), MIPS object code (application)	Text, program-dependent	Distrib. shared mem., coherence, consistency
Netsim [11]	Network Simulation, Tool Building	Unix	C	Text, program-dependent	Network topologies, routing, protocols
Pantheon [28]	Storage systems I/O simulation	Unix	Tcl, C++	text statistics	I/O, storage systems, RAID
LSU Proteus [3]	Multiprocessor Simulation	Sparc Solaris	C w/ ANL macros, config. (Text/GUI)	Text trace, GUI	Distrib. shared mem., coherence, networks
RSIM [22]	Multiprocessor Simulation	SGI, Sparc V9 Solaris, Convex	object code	Text Statistics	**Speculation, branch pred.**, consistency, **dynamic sched.**
Shade† [6]	Uniprocessor Simulation, Tool Building	Sparc Solaris	C source (analyzer) Sparc object code (application)	Text, program-dependent	**Instruction set, pipeline, memory hierarchy**
SimICS† [20]	Uni/Multiproc. Simulation	Sparc Solaris	C source (analyzer) Sparc object code (application)	Text, program-dependent	**Memory hierarchy**, O/S, multitasking
SimOS [25]	Uni/Multiproc. Simulation	MIPS, Solaris (with cross-compilation)	Tcl script (analyzer), MIPS object code (application)	Text; Tcl/ Perl and/or GUI	**Memory hierarchy**, O/S, multitasking, I/O
SimpleScalar [2]	Uniprocessor Simulation	x86, RS6000, Sparc, Alpha, PA-RISC	C, Fortran	Text, program-dependent	**Instr. set, memory hierarchy, superscalar**, branch pred., speculation
TangoLite [7]	Multiprocessor Simulation	MIPS	C/Fortran with ANL macros	Text	Distrib. shared-memory, mem. hier., coherence
VTune	Performance Analysis	x86, MS-Windows	C/C++, Fortran	GUI, statistics	**CISC microarchitecture, superscalar, mem. hier.**
WARTS: QPT [16], Dinero-III, Tycho, WWT-II [19]	Cache/Program Profiling and Simulation, Uniprocessor/ Multiprocessor	Sparc Solaris NOWs, SMPs	C w/ PARMACS macros (WWT), object code (QPT), traces (Dinero, Tycho)	Text traces, statistics	**Instr. set, mem. hierarchy**, distrib. shared mem., coherence, consistency network interfaces

Table 1: Simulation/Tracing tools (†indicates that only object code is available). *Analyzer* refers to the simulator engine provided by the user in tools such as Shade; *Application* refers to the actual workload to be simulated. Some tools provide intermediate textual outputs that may be post-processed by a GUI visualization tool. All topics shown in the rightmost column are covered in graduate courses and can benefit from integration with one or more tools. Typically, topics in boldface are also covered in undergraduate courses.

Some tools have been specifically developed for educational purposes, i.e. with the goal of illustrating computer architecture concepts that are easily visualized through simulation. While leveraging the availability of such tools, the curricula improvements discussed in this

Tool Name / Laboratory	ATOM	AUGMINT	Chamel.	DLX	HPAMSim	MINT	Netsim	Pantheo	Proteus	RSIM	Shade	SimICS	SimOS	SScalar	TangoLi	VTune	WARTS
Multiprocessor Simulation		√			√	√			√	√		√	√		√		√
Uniprocessor Simulation	√		√	√							√	√	√	√			
Cache Simulation	√			√							√	√		√			√
I/O Simulation								√					√				
Network Simulation							√		√	√							
Instruction Set	√			√							√			√		√	√
Instruction Level Parallelism			√	√						√				√		√	
Education	√	√		√					√	√	√		√	√		√	√

Table 2: A possible assignment of tools to virtual laboratories

Tool	**Purpose**	**Host Arch.**	**Input**	**Output**	**Related course topics**
CIAT [1]	Application Analysis Tool	Unix platforms	Program and data files	Data files	Application, benchmark characterization
National Compiler Infrastructure [8]	A Family of Compilers	Unix, x86 platforms	Fortran, C, C++, Java	C, Machine code	**Compiler case studies, internal representations**
Max/P	Detects maximum degree of parallelism	Unix platforms	Fortran	ASCII report	Parallel programming, determining achievable limits
Pablo [24]	Performance analysis, visualization tool	Unix, Windows/NT	Fortran, C, SDDF files	Interactive display	Performance models, characteristics of applications
Paradyn [17]	Performance analysis tool	Unix, Windows/NT	Source programs, object code	Interactive display	Performance models, characteristics of applications
Polaris [4]	Parallelizing compiler	Unix	Fortran, C	Fortran	**Compiler case studies, transformations, internal representations**
UrsaMinor [23]	Performance optimization tool	Java	Source programs, database, SDDF	Interactive, WWW, database	Parallel programming methodology, relationship of compilers and performance analysis

Table 3: Compilers and tools in the Parallel Programming Lab. (Course topics in bold face indicate undergraduate curriculum.)

paper have different nature and goals. The objective is to introduce tools and the underlying techniques as objects of study and use. Using an analogy, the goal is to integrate into computer engineering curricula the equivalent of Matlab for digital signal processing and Mentor Graphics for VLSI design. While these packages certainly provide means of illustrating basic concepts, they also introduce students to fundamentals and skills needed to carry out advanced designs in the respective disciplines.

Given the desirability of exposing students to the nature, inner workings and use of the above mentioned tools, it is important to find ways of effectively incorporating them into existing computer engineering curricula. However, with few exceptions, many of the above tools have strong dependencies on resources, technical environment and curriculum of the institution where they were developed. In addition, a large number of tools are intended for use by tool experts who have an inherent understanding of their inner workings and the necessary skills to quickly learn how to use them. This reality presents several requirements or obstacles to the incorporation of a tool into the curriculum:

- the existence of platform(s) to run the tool,
- the need to install software (including but not limited to the tool itself),
- tool code modifications required by the local computing environment (e.g. due to OS versions and computing center policies),
- accessing, collecting and distributing the appropriate documentation and possibly source code (for both users and "developers"),
- user interface needs for targeted educational uses and
- the need to mitigate the often "unfriendly" nature of leading-edge prototype tools.

The overhead associated with satisfying the above requirements is multiplied by the moderately large number of tools needed to cover the different subareas of computer architecture and parallel programming. It requires significant commitments of human and machine resources and it has been a major factor in slowing down the integration of tools into the curriculum.

The above mentioned obstacles and overheads can be addressed or minimized by using a unique web-based software infrastructure developed at Purdue with the purpose of providing universal access to simulation tools and necessary resources. The infrastructure - PUNCH - provides its users with the ability to access virtual laboratories of tools (topically organized as illustrated in tables 1, 2 and 3) where they can read educational materials (e.g. manuals, frequently-asked-questions and articles), examine examples and *run* any tool. For instructors and students, the overheads of locating, porting, learning a tool and finding resources to run it are minimized if not eliminated.

III. Curriculum development

Most, if not all, computer engineering curricula include several classes on computer architecture and system software. They use established textbooks, which provide a stable source of well-organized fundamental knowledge on the subject. Table 4 shows courses currently offered at the institutions participating in this effort. Typically, one or more courses of a computer engineering curriculum require students to carry out a design project that involves limited design and simulation of one or more parts of a simple computer (typically, a CPU and/or cache and memory system) and evaluate it with an appropriate workload. Undergraduate classes with relatively large enrollments often (re)use variants of a standard project which necessarily emphasizes a particular aspect of the class material. The overhead incurred in setting, maintaining and administering these large projects is nontrivial. It is often the case that simplifications of the design and ad-hoc techniques or tools are used to keep overheads at an acceptable level.

The project-only approach described above, while providing students with valuable insights into the operation of a simple processor, exposing them to the intricacies of microarchitecture design and providing an understanding of the software environments, suffers from several disadvantages:

Course #/ Title	Insti- tution	Level/ Frequency	#Students Per Offering/ Instructor	Elective/ Required	Brief Description of Innovation
EE365 Intro. Design of Dig. Computers	Purdue School of ECE	Junior Fall, Spring	80	Required	Integration of tools and subjects as per Table 1
EE565 Computer Architecture	Purdue School of ECE	Senior and Year 1 grad Fall, Spring	70	Elective for seniors	Integration of tools and subjects as per Table 1
EE468 Intro. to Compilers	Purdue School of ECE	Senior Fall	80	Required	Integration of tools and subjects as per Table 3
EE563 Parallel Programming	Purdue School of ECE	Senior and Year 1 grad Spring	15	Elective	Integration of tools and subjects as per Table 3
EE573 Compiler Systems	Purdue School of ECE	Senior and Year 1 grad Fall	15	Elective	Integration of tools and subjects as per Table 3
ECE C61 Computer Architecture	Northwestern Dept. of ECE	Ju./Se. and Year 1 grad Fall, Winter	30	Required	Integration of tools and subjects as per Table 1
ECE C62 Comp. Arch. (capstone)	Northwestern Dept. of ECE	Ju./Se. and Year 1 grad Winter	15	Required	Integration of tools and subjects as per Table 1
ECE D52 Advanced Comp.Arch.	Northwestern Dept. of ECE	Senior and Year 1 grad Spring	15	Elective	Integration of tools and subjects as per Table 1
ECE D55 Distributed Systems	Northwestern Dept. of ECE	Ju./Se. and Year 1 grad Fall	15	Elective	Integration of tools and subjects as per Table 1
Cptr 303 Computer Org. and Arch.	Chicago State Dept. of CS	Junior and Senior Fall,Spring	30	Required	Integration of tools and subjects as per Table 1
Cptr 370 Special Topics in Comp. Sci.	Chicago State Dept. of CS	Junior and Senior Fall	30	Elective	Integration of tools and subjects as per Table 1,3
Cptr 341 Modeling and Simulation	Chicago State Dept. of CS	Junior and Senior Fall,Spring	30	Elective	Integration of tools and subjects as per Table 1

Table 4: Courses targeted for curriculum innovation and tool integration. Academic year is divided into semesters at Purdue and Chicago State University, and into quarters at Northwestern University. Testing and selection of appropriate tools for each subject are part of the ongoing activities.

- instructors and students tend to focus on the topics needed to complete the project; other topics tend to be covered lightly and without exposure to associated tools,

- application and benchmarks characterization are often replaced by references to collected statistics of their behavior; while this is sufficient to provide justification of computer architecture concepts, it prevents student exposure to techniques and tools for gathering such statistics as well as to questions on how to interpret them,
- students are not exposed to simulation techniques and models that can be used to estimate the performance of a particular component of a computer without having to design (and typically emulate) an entire processor and
- last but not least, no insights are provided into the inner workings of tools for computer architecture and parallel programming and techniques for their validation; having access to such tools would enable students to understand how the outputs of tools are generated and would also provide students with some ability to modify or build tools for other purposes.

The curriculum innovation consists of including fundamentals and experiments in tool use and design into existing classes that cover computer architecture and parallel programming topics. The extent and emphasis of each experiment depend on the nature and level of the course. The goal is to expose students to tools and quantitative design and evaluation techniques along with the study of basic and advanced concepts of computer systems. In courses with or without a project, instructors can enrich student assignments with tool-using design and evaluation problems of appropriate complexity. Currently, implementing such a course is an undertaking that is very costly (in resources and time) for both instructors and students. The approach pursued by the authors is to develop materials, procedures and infrastructure that make it possible for instructors and students to succeed in such courses. PUNCH, the unique web-based infrastructure developed at Purdue University and accessible to participating institutions, provides much of the technology needed to enable the stated approach. Tools and associated educational materials are accessible and, in the case of tools, also *executable*. Instructors at different institutions are able to integrate individual tools into class assignments.

Curriculum enhancements

The computer systems courses offered at the participating institutions (shown in Table 4) collectively cover the same "classical" topics at depth commensurate with the level of the class. Therefore, the ensuing discussion is applicable to all institutions. The proposed enhancements can be divided into two classes: tool fundamentals and tool-based experiments. The first includes additional course content needed to cover technical aspects of tool and experiment design. The second corresponds to class assignments that require the use of tools for design and evaluation purposes.

Tool fundamentals

The correct use, design and engineering of tools subsumes the understanding of many fundamental concepts across several knowledge areas. These topics include probability, statistics, systems programming, software engineering and the general area of simulation. A sample of relevant topics in probability and statistics includes how to summarize data, data sampling, confidence intervals, regression models and queuing theory. Other concepts closely related to simulation include random-number generation, experiment design and useful probability distributions. Topics from other areas include discrete event simulation, tracing techniques, program-driven and execution-driven simulation, instrumentation techniques, concurrency issues (e.g. timing, coordination, synchronization), workload characterization and dynamic compilation techniques.

The coverage of all of the above-referred concepts in full depth could require one or more courses but another effective approach is possible. Many of these topics and their prerequisite knowledge are covered in other classes in contexts that are possibly unrelated to computer architecture and programming. For example, courses on probability theory and programming cover some of the topics mentioned above. By revisiting and extending these subjects in the targeted courses, students are able to link and apply previous concepts in computer architecture and programming classes. To best serve students at different levels and with diverse technical knowledge, educational self-contained modules can be developed to address individual topics. Thus, when covering a particular subject in the targeted courses, it becomes possible to introduce students to a related engineering tool *and* the module(s) needed to explain *why* and *how* that tool works. The written materials for each of the modules are made accessible to instructors and students through PUNCH. They can be incorporated into the courses at each institution in a manner that fits their specific curricula and instructor preferences.

Computer architecture curricula and tools

The content of computer architecture classes at the institutions involved in the project is typical of similar classes at many institutions. It includes the "standard" topics of datapath design, control design, pipelining, caches, memory organization, Input/Output and storage systems, networking and multiprocessors. The undergraduate and graduate texts ([9] and [10]) written by Hennessy and Patterson are currently used and typify the nature and level of the topics covered in these courses.

Based on previous research and educational activities an initial set of tools that meet the criteria and goals of this project has been selected. Tables 1 and 4 show how several tools are in correspondence with the traditional topics covered in computer architecture classes. However, part of the ongoing work is to evaluate and select the best available tool for a given topic through experience as well as interactions with other researchers and educators. Some of the tools are well known and find widespread use in industry and academia. Other tools for advanced topics have been recently developed but are stable enough for use in teaching undergraduate classes. Currently, the selection of tools reflects the dominance of SUN computer systems in the student-accessible computational facilities available to PUNCH.

Compiler and parallel programming curricula and tools

Existing curricula include courses introducing students to basic and advanced compiler topics. For example, at Purdue two courses are available to undergraduate students: "(EE468) Introduction to Compilers and Translation Engineering", and "(EE 573) Compilers & Translator Writing Systems". Currently, the first course introduces students to the design and construction of compilers and other translators. Topics include compilation goals, organization of a translator, grammars and languages, symbol tables, lexical analysis, syntax analysis (parsing), error handling, intermediate and final code generation, assemblers, interpreters, and an introduction to optimization/parallelization. Emphasis is on engineering, from scratch, a compiler or interpreter for a small programming language – typically a C or Pascal subset. Projects involve the stepwise implementation of such a system.

The second course presents the concepts needed to efficiently design and implement translators. Basic compiler/translation theory and technology are briefly reviewed, then the course focuses on software tools for the automatic construction of translators, as well as more complex concepts involving the construction of compiler symbol tables, etc. Each student constructs a simple lexical-recognizer, parser, and code-generator.

The current curriculum also offers the course "(EE563) Programming Parallel Machines". The course examines how to program parallel processing systems. Various parallel algorithms are presented to demonstrate different techniques for mapping tasks onto parallel machines. Parallel architectures to be considered are: SIMD (synchronous), MIMD (asynchronous), mixed-mode (SIMD/MIMD hybrid), shared-memory, and distributed memory architectures. There are several programming projects, one on each machine. The similarities and differences among the machines and their languages will be discussed. The curriculum at Northwestern University also offers the course "(ECE D55) Distributed Computing Systems." It examines the fundamentals, design issues and various paradigms of distributed systems including message passing, client-server and distributed shared memory. The course includes also programming assignments and projects.

Ongoing curriculum enhancements provide advanced research compilation tools that expose students to new compiler technology and let the students try out and experiment with working compilers. Tables 3 and 4 show an initial set of tools and how they are related to course topics.

IV. PUNCH, the enabling infrastructure for integration of tools into curricula

The proposed curricula innovations leverage software developed for the implementation of the Purdue University Network-Computing Hubs (PUNCH) [15, 13, 14]. PUNCH is a Perl-based infrastructure (with about 12,000 lines of code) that can be viewed as an operating-system for the world-wide web. It provides: 1) transparent and universal access to remote programs and resources, 2) access-control (privacy and security) and job-control (run, abort, and program-status) functionality in a multi-user, multi-process environment, and 3) support for logical (virtual) organization and management of resources. PUNCH allows users to: a) upload and manipulate input-files, b) run programs, and c) view and download output - all via standard WWW browsers.

Figure 1 shows a global view of PUNCH's organization. PUNCH consists of the front-end, called the Hub, and the back-end, called SCION for Scalable Infrastructure for On-demand Network computing. The Hub consists of a collection of WWW interfaces and associated logical groupings of tools (i.e. laboratories and discipline-specific hubs). SCION manages the access and control of distributed resources and tools to serve requests from the front-end (see Figure 1). A resource can be an arbitrary platform and can be added incrementally by specifying its architecture (make, model, OS, etc.) and starting a server on it. A new tool can be added by providing to PUNCH information about the location of the tool, input/output

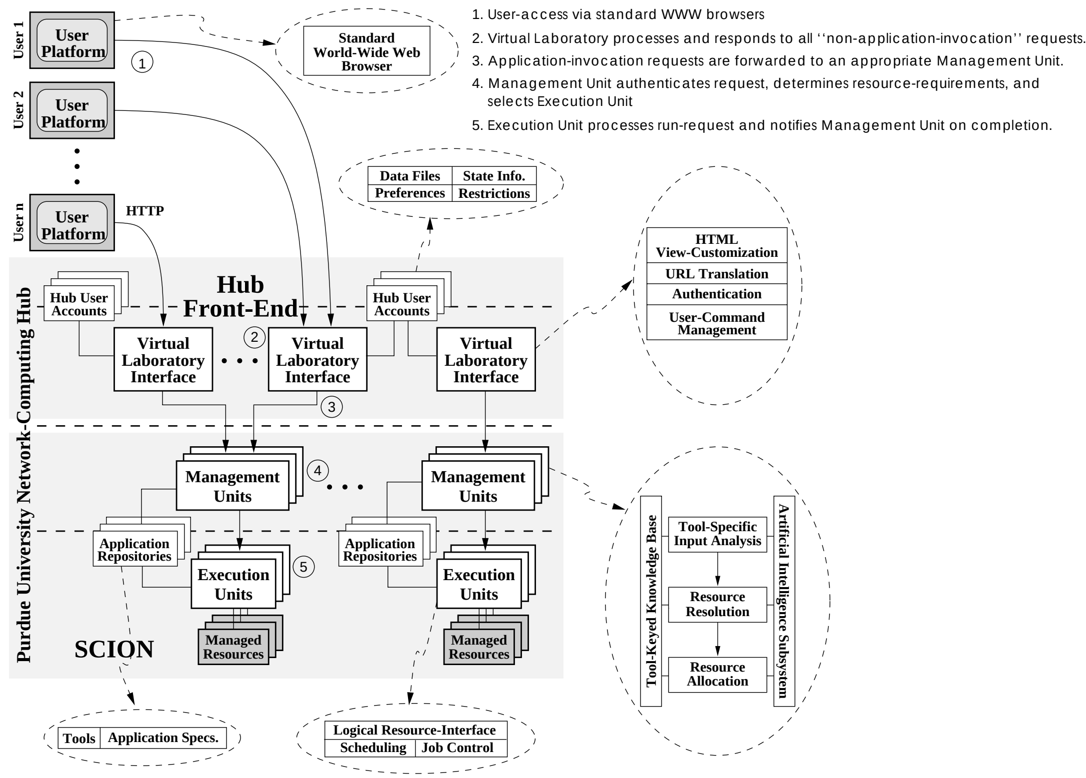

Figure 1: The organization of PUNCH. All components of the infrastructure (virtual laboratory interfaces, management units, and execution units) can be independently replicated. The horizontal dashed lines indicate network interfaces. The client units (part of the SCION infrastructure) are integrated into the virtual laboratory interfaces of the Hub front-end.

behavior, machines where it can run and its logical classification in the Hub.

The Hub - the customizable user's view of PUNCH

Users can "enter" PUNCH either as members or as guests. Members have their own (private) hub-accounts, and can access the full functionality of the Hub. Guests, on the other hand, can only view example (public) input files and precomputed output. PUNCH accounts can be requested on-line. Tools on the Hub are indexed and cross-referenced by way of keywords. For example, a tool may be indexed according to its functionality (e.g., parallelizing compiler), and/or according to the university at which it was developed. Tools on the Hub are also accompanied by descriptions of their capabilities, manuals, example input files and pre-computed output, and the email addresses of the associated support sites. If a tool has a home-page on the web, a link to it is also provided. Running a simulation on the Hub is a three-step process that can be started from any networked WWW client (Figure 2). The first step involves the creation of the input file(s) required for the relevant simulation, which can be done via the Hub's editor interface. (General file-manipulation and utility functions such as copying, deleting, and compressing files are also available). Files may also be uploaded to the Hub. In the second step, users define the input parameters (e.g., command-line arguments) for the program and start the simulation. In order to support programs with a wide range of input characteristics, the Hub uses a programmable interface-generator to

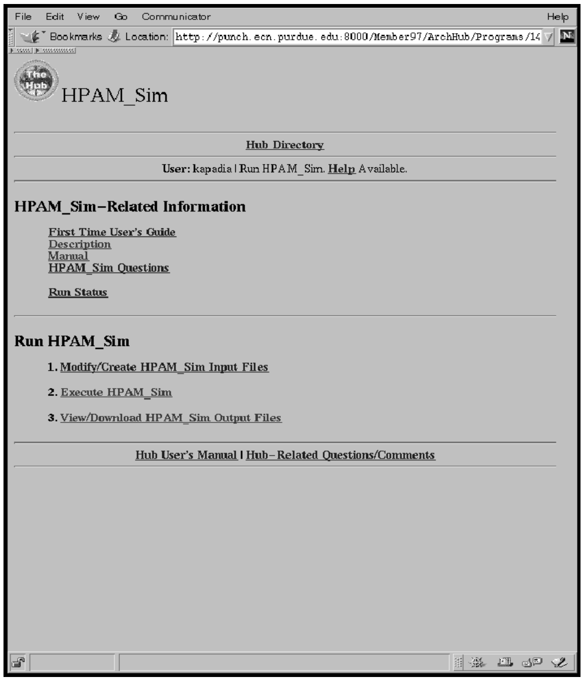

Figure 2: The first form of the dynamically generated user-interface for HPAM_Sim.

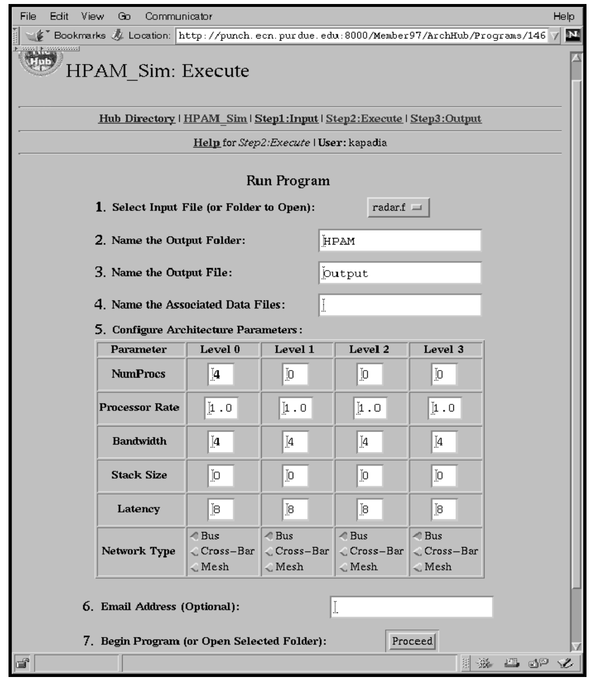

Figure 3: The dynamically generated execution interface form for HPAM_Sim.

dynamically generate the appropriate HTML pages from a template specification; an example page is shown in Figure 3. Finally, after the simulation is complete, the user can see, postprocess and download the results of the simulation via the Hub's output interface.

The web-based environment can be customized for different classes of users. Almost all the pages displayed by the Hub are dynamically generated. This allows the Hub to present different views of available resources to different users. For example, the Hub front-end can be configured so that a proprietary tool 'T' is only visible to a specified sub-set of users 'U'. With this configuration, the Hub will exclude 'T' when generating pages for users who do not belong to 'U'. Moreover, the Hub enforces such access-restrictions regardless of the URL used by the user.

Conclusions

A large percentage of computer engineering graduates will have to use computer-based tools in their jobs. These tools are used by computer architects, system integrators, system and application programmers, computer evaluators and consultants and others who must be able to understand how computer systems features affect the performance of their engineering artifacts. In many cases it is up to the engineer to (re)design a tool that addresses a specific project need. The knowledge and experience provided by the proposed curriculum innovations (on the use and development of computer system tools) will uniquely qualify engineers to succeed in carrying out the above mentioned activities.

Teaching students how to simulate systems and design tools for simulation and evaluation purposes significantly impacts their ability to master fundamental knowledge and think critically. Simulation entails two intellectual challenges: how to reproduce the behavior of the (sub)system of interest and how to do so at the appropriate level of abstraction. Performance evaluation focuses the students' mind on optimization and modeling issues. Analyzing software characteristics provides crucial insights into the applications that drive computer systems innovation. These intellectual competencies are core to the practice of engineering. By the inherent nature of tools and the assignments they enable, collaborative design and tool development activities can be expected to ensue from the proposed innovations.

This article argues that an innovative approach to the integration of tools into existing curricula is possible by leveraging emerging network-computing technology that reduces or eliminates the overheads involved in locating, testing, learning, using and maintaining tools. By complementing such a capability with educational materials developed in a modular fashion it becomes possible to enhance existing classes so to cover fundamental and practical aspects of computer tools. This allows for customized solutions that take into consideration size, class level, instructor views and other aspects of a specific program. Evaluation results of this project will be reported in future papers.

Bibliography

[1] G. Abandah and E. S. Davidson. Configuration Independent Analysis for Characterizing Shared-Memory Applications. 12th Int'l. Parallel Processing Symposium, 1998.

[2] D.C. Burger, T. M. Austin, and S. Bennett. Evaluating Future Microprocessors-the SimpleScalar Tool Set. UW Computer Sciences Technical Report #1308, July, 1996.

[3] E. A. Brewer, C. N. Dellarocas, A. Colbrook and W. E. Weihl. PROTEUS: A High-Performance Parallel Architecture Simulator. Technical report mit/lcs/tr-516, Massachusetts Institute of Technology, Sept. 1991.

[4] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger and P. Tu. Parallel Programming with Polaris. *IEEE Computer*, Dec 1996.

[5] Z. Ben-Miled, J.A.B. Fortes, R. Eigenmann and V. Taylor. Towards the Design of a Heterogeneous Hierarchical Machine: A Simulation Approach. In 30th Simulation Symp., Apr. 1997.

[6] B. Cmelik and D. Keppel. Shade: A fast instruction-set simulator for execution profiling. In Proceedings of the 1994 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1994.

[7] Helen Davis, Stephen R. Goldschmidt and John Hennessy. Multiprocessor Simulation and Tracing Using Tango. In Proceedings of the 1991 International Conference on Parallel Processing (ICPP, Vol. II, Software), August 1991.

[8] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, E. Bugnion and M. S. Lam. Maximizing Multiprocessor Performance with the SUIF Compiler. *IEEE Computer*, December 1996.

[9] John Hennessy and David Patterson. *Computer Organization and Design: The Hardware-Software Interface (Appendix A, by James R. Larus).* Morgan Kaufman, 1993.

[10] John L. Hennessy, David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1996.

[11] J. Robert Jump. *NETSIM Reference Manual.* Rice University, Houston, TX, May 1993.

[12] Kapadia, Nirav H. and Fortes, José A. B. On the Design of a Demand-Based Network-Computing System: The Purdue University Network-Computing Hubs. In Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC'98), pp71-80, July 1998.

[13] N. Kapadia, J. Fortes and M. Lundstrom. The Semiconductor Simulation Hub: A Network-Based Microelectronics Simulation Laboratory. In 12th Biennial IEEE University Government Industry Microelectronics Symposium, pp. 72-77, July 20-23, 1997.

[14] N. Kapadia, M. Lundstrom, J. Fortes and K. Roy. Network-Based Simulation Laboratories for Microelectronics Systems Design and Education Multiprocessors. In International Conference on Microelectronic Systems Education, pp. 23-24, July 21-23, 1997.

[15] N. Kapadia, J. Fortes and M. Lundstrom. The Computational Electronics Hub: A Network-Based Simulation Laboratory. In Workshop on Materials and Process Research and the Information Highway, National Materials Advisory Board, Commission on Engineering and Technical Systems, National Research Council, Apr 12-13, 1996 (pp. 31 of summary record published by National Academy Press).

[16] James R. Larus. Efficient Program Tracing. *IEEE Computer*, 26(5):52–61, May 1993.

[17] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam and Tia Newhall. The Paradyn Parallel Performance Measurement Tools. *IEEE Computer 28(11)*, November 1995.

[18] J. H. Moreno, M. Moudgill, K. Ebcioglu, E. Altman, C. B. Hall, R. Miranda, S.-K. Chen and A. Polyak. Simulation/evaluation environment for a VLIW processor architecture IBM Journal of Research and Development, Vol. 41, No. 3, 1997

[19] Shubhendu S. Mukherjee, Steven K. Reinhardt, Babak Falsafi, Mike Litzkow, Steve Huss-Lederman, Mark D. Hill, James R. Larus and David A. Wood. Wisconsin Wind Tunnel II: A Fast and Portable Architecture Simulator. In Proceedings, Workshop on Performance Analysis and its Impact on Design, June 1997 (PAID-97).

[20] Peter S. Magnusson and Bengt Werner. Efficient Memory Simulation in SimICS. In 28th Annual Simulation Symposium, 1995.

[21] A-T. Nguyen, M. Michael, A. Sharma, J. Torrellas. The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures. In Proceedings of 1996 International Conference on Computer Design, October 1996.

[22] Vijay S. Pai and Parthasarathy Ranganathan and Sarita V. Adve. RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors, In Proceedings of the Third Workshop on Computer Architecture Education, October 1997.

[23] Insung Park, Michael J. Voss, Brian Armstrong, and Rudolf Eigenmann. Parallel programming and performance evaluation with the URSA tool family. *Int'l Journal of Parallel Programming*, 1998. to appear.

[24] Daniel A. Reed, Ruth A. Aydt, Roger J. Noe, Phillip C. Roth, Keith A. Shields, Bradley Schwartz and Luis F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In Proceedings of the Scalable Parallel Libraries Conference, IEEE Computer Society, 1993.

[25] Mendel Rosenblum, Stephen A. Herrod, EmmettWitchel, and Anoop Gupta. Complete Computer Simulation: The SimOS Approach. In *IEEE Parallel and Distributed Technology*, Fall 1995.

[26] Amitabh Srivastava and Alan Eustace. ATOM: A System for Building Customized Program Analysis Tools. In Proceedings of the 1994 ACM Conference on Programming Language Design and Implementation (PLDI), June 1994.

[27] Jack E. Veenstra and Robert J. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 201–207, 1994.

[28] J. Wilkes. The Pantheon Storage System Simulator Tech. report HPL-SSP-95-14, Hewlett Packard, Palo Alto, CA

Dr. JOSÉ A. B. FORTES is currently a Professor and Assistant Head for Education of the School of Electrical and Computer Engineering at Purdue University. Dr. Fortes received a B.S.E.E. degree from Univ. de Angola in 1978, a M.S.E.E. degree from Colorado State University in 1981, and a Ph.D. in Electrical Engineering and Systems from the University of Southern California in 1984.

NIRAV KAPADIA is currently a doctoral candidate at Purdue University. Nirav Kapadia received his B.E. degree in Electronics and Telecommunications from Maharashtra Institute of Technology (India) in 1990. He subsequently worked at the institution for a year as a lecturer. He received his M.S.E.E. degree from Purdue University in 1994.

Dr. RUDOLF EIGENMANN is currently an Associate Professor of Electrical and Computer Engineering at Purdue University. Dr. Eigenmann received a Diploma in Electrical engineering from ETH Zürich, Switzerland in 1980, and a Ph.D. in Computer Science from ETH Zürich in 1988.

RENATO FIGUEIREDO is currently a doctoral candidate at Purdue University. Renato Figueiredo received both B.S.E.E. and M.S.E.E. degrees from the State University of Campinas (UNICAMP), Brazil, in 1994 and 1995.

Dr. VALERIE TAYLOR is currently an Associate Professor of Electrical and Computer Engineering at Northwestern University. Dr. Taylor received a B.S.C.E.E. degree with highest honors from Purdue University in 1985, a M.S.E.E. degree from Purdue University in 1986, and a Ph.D. degree in Electrical Engineering and Computer Science from the University of California at Berkeley in 1991.

Dr. ALOK CHOUDHARY is currently an Associate Professor of Electrical and Computer Engineering at Northwestern University. Dr. Choudhary received a B.E. degree in Electrical Engineering from Birla Institute of Technology and Science in 1982, a M.S. degree in Electrical and Computer Engineering from the University of Massachusetts in 1986, and a Ph.D. in Electrical and Computer Engineering from the University of Illinois in 1989.

Dr. LUIS VIDAL is currently an Associate Professor of Mathematics and Computer Science at Chicago State University. Dr. Vidal received a B.S. degree in Mathematics from the University of Trujillo in 1970, M.S. degrees in Computer Science and Mathematics from Washington State University in 1981 and 1983, and a Ph.D. degree in Computer Science from the Illinois Institute of Technology in 1990.

Dr. JAN-JO CHEN is currently an Assistant Professor of Mathematics and Computer Science at Chicago State University. Dr. Chen received a B.S. degree in Architecture from Tunghai University, Taiwan in 1985, a M.S. degree in Computer Science from the Illinois Institute of Technology in 1990, a M.S. degree in Mathematics, Statistics and Computer Science from the University of Illinois at Chicago in 1996, a M.S. degree in Management Information Systems from the University of Illinois at Chicago in 1997, and a Ph.D. degree in Mathematics, Statistics and COmputer Science from the University of Illinois at Chicago in 1997.