Theme Introduction

As Eniac Turns 50: Perspectives on Computer Science Support for Science and Engineering

Rudolf Eigenmann, *Purdue University*
George Cybenko, *Dartmouth College*

1996 marks the 50th year of the IEEE Computer Society. It also marks the 50th anniversary of Eniac.

Eniac's creation gave birth to an era of remarkable technical progress. Even more so, it made the wildest dreams about the applicability, performance, and miniaturization of the new technology come true. Eniac's line of descendants include most computers in use today: laptops, PCs, workstations, as well as supercomputers. It also includes every microprocessor that is embedded in today's typewriters, washing machines, automobile dashboards, and in control systems of industrial robots and assembly lines.

Eniac's creation will probably be remembered as one of the most important engineering events of the 20th century. This major milestone is now at a 50-year distance. As the next century is about to begin, it is appropriate to reflect on where "the age of computing" has brought us so far and where we might find ourselves in the future. Several well-known experts of our post-Eniac era have accepted an invitation to present their perspectives. They will give us their views of the past, present, and future—with a focus on the subject of this magazine: computational science and engineering.

In particular, this special theme section focuses on the current state of parallel computing and rules of thumb for its application, in an article by Cherri Pancake; a position paper on the structure of computational science and engineering education as seen by Greg Wilson, with responses from Rubin Landau and Steve McConnell; and five essays kicking off a series of short "forecasts" by experts in computational science. Finally, several of the department contributions in this issue round out our look at current and future trends.

Pancake offers a realistic look at where parallel programming stands

1070-9924/96/$5.00 © 1996 IEEE

today and how practitioners can take advantage of the technology without getting trapped in the pitfalls. Parallel computing will continue to be a major force in CSE applications but it needs to be harnessed deliberately, building on the experiences of others.

Wilson proposes a short course—only one week long!—to introduce modern computing technology to scientists and engineers as they begin their graduate careers. His particular decisions on what to include and exclude can certainly be debated. Rubin Landau provides counterpoint in response, advocating more attention to basic principles and less to what he sees as fleeting technologies. Yet another response to Wilson's proposal comes from Steve McConnell, who argues a case for teaching more software engineering principles. In his view, too many CSE students engage in large computing projects without much guidance or background in managing complex software projects. We hope that these lively exchanges are the start of some thoughtful discussion about CSE education from all perspectives, and readers are encouraged to respond to the positions taken in these articles.

Additionally, our "forecast" essays were contributed by Mani Chandy (California Institute of Technology), Jack Dongarra (University of Tennessee), Yoichi Muraoka (Waseda University, Japan), Yale Patt (University of Michigan), and Eric Grosse (Bell Laboratories). In the Fall issue we will also have forecasts from David Padua (University of Illinois), Howard Jay Siegel and Craig Stunkel (Purdue University and IBM T.J. Watson Research Center, respectively), Harry Wijshoff (Leiden University, the Netherlands), and Hans Zima (University of Vienna, Austria), as well as other essays about trends in individual CSE applications areas.

These essays describe the evolution of programming languages, networking, compilers, software libraries, performance evaluation, microprocessors, and interconnection networks. They show us roadmaps to where we are heading and discuss new approaches. Some of the ideas might sound like dreams—dreams that Eniac taught us can come true. Examples are a "scientist's infosphere" and a seamless, ocean-spanning infrastructure for computational science and engineering.

Rather than summarize the already short essays, we pick out a recurring theme: many of the authors refer to *problem-solving environments* in one form or another. PSEs are needed to bridge the gap between increasingly complex machines (whose exploitation takes more and more expertise) and the growing community of non–expert users. PSEs are needed to generate efficient code and still provide a user-friendly interface. PSEs are one of the forces driving the evolution of programming languages and compilers. PSEs are what users will see as part of their infosphere. PSEs will probably be part of the seamless infrastructure for computational science and engineering. And a PSE could describe the harness of microprocessors and interconnection networks, making them powerful resources for scientists and engineers.

But, what is a problem-solving environment? It is more than a programming language, it is more than a compiler. For the user it feels like a powerful interface that lets us specify a problem in our own words. Then it solves the problem efficiently with the right resources. These resources might be our desktop microprocessor providing some compute cycles. If this is not powerful enough, the PSE will locate more cycles in other computers on our network—possibly on a heterogeneous suite of machines, each best equipped for solving a certain subproblem. These resources could also be information servers located in some scientist's database across the ocean, so the PSE might solve this problem using World Wide Web technology.

Perhaps a shorter way to describe the future of scientific engineering is that we will create The Problem-Solving Environment. One of the biggest challenges will be to make sure this term does not become synonymous with The Magic Wand, which we don't know how to build. ◆

George Cybenko *is the Dorothy and Walter Gramm Professor of Engineering at Dartmouth, and is both editor-in-chief of* IEEE CS&E *and its area editor for signal and image processing. His research focuses on SIP algorithms, neurocomputing, and computer performance analysis. His PhD is in the applied mathematics of electrical engineering and computer science from Princeton. E-mail, george.cybenko@dartmouth.edu; WWW, http://mmm.dartmouth.edu/pages/gvc/gvc-hp.html.*

Rudolf Eigenmann *is an assistant professor of electrical and computer engineering at Purdue University. He holds a PhD in electrical engineering/computer science from ETH Zurich, Switzerland. His main interests are in compiler technology for parallel computers, engineering of computational applications, and performance evaluation. E-mail, eigenman@ecn.purdue.edu.*