Crossroads
The ACM Student Magazine

Spotlight: An Interview With 2006 Turing Award Winner Frances E. Allen

by ***Daniel Alex Finkelstein***

Often referred to as the Nobel Prize of computer science, the A.M. Turing Award recognizes extraordinary achievement in computer science research and development. Fran Allen received the award this year at the ACM Federated Computing Research Conference in San Diego, a meta-conference at which 16 ACM conferences are co-located once every three years. I spoke with Allen after her **keynote address** about her career, her thoughts on technology and education, and in particular, further commentary on her challenges to the status quo.

Allen's Turing Award citation singled out her "pioneering contributions to the theory and practice of optimizing compiler techniques that laid the foundation for modern optimizing compilers and automatic parallel execution," achieved during her 45-year career at IBM Research. The eponymous award was named in honor of Alan Turing, a remarkably influential British mathematician and cryptographer whose works ranged from devising the technique of decoding the German Enigma machine, a World War II era cryptography system, to the popular Turing Test, which postulates a set of questions needed for a human to determine if the second participant in a conversation is man or machine.

Allen's Turing acceptance keynote address was scheduled for the first Sunday of the conference in a hall at the end of the convention center. The cavernous hall contained at least several hundred chairs—perhaps a couple of thousand—five giant projection screens that hung from the ceiling, several TV cameras, and a large stage at the front that looked nearly identical to the stage from which Steve Jobs periodically announced the release of his newest and most sophisticated gadgets. The scale was startling, but its point was clear: the stage was set for something—someone—important.

Programming languages, compilers, and operating systems: all fields of computer science in which Allen is distinguished. Allen had just joined the ranks of Don Knuth, John Backus, Maurice Wilkes, and John Cocke, all former winners of the Turing Award, and all giants in the fields of mathematics or computer science; names that undergraduates and graduates alike encounter along their studies, if not while practicing research in their fields. As well may be suited for a giant, a favorite pastime of hers is mountain-climbing.

I took a seat in the front row near the podium, anxiously anticipating the arrival of the honoree. I tried to write out what I might say to introduce myself; although we had already communicated by email, I had never actually spoken with her in person. Not a moment later, however, she appeared. She walked onto the stage in an unassuming blue jacket and lavender shirt with an easy smile revealing she was, in fact, human. Human scale, as well: she was dwarfed by the stage and backdrop. Gathering up my strength, I hesitatingly walked over and introduced myself. We chatted briefly before she began delivering her hour-long keynote (the keynote is now available online at the ACM web site), and for such an accomplished engineer and scientist, Allen could not have been more accommodating and approachable.

In 1957, Allen joined IBM Research to teach Fortran to IBM's scientists. She gradually moved into the design of compilers for future generations of IBM computers, mostly the large scale business and scientific models. Her most recent major projects at IBM were Parallel Translation (PTRAN), and Blue Gene, both of which she discussed in her award keynote. The first female IBM Fellow, she retired from IBM five years ago, although this retirement by no means marked the end of an extensive career. Since then, she has traveled extensively, advocating careers in computer science for women, championing the profession and vocation of computer science, and also encouraging researchers, scientists, and students to pursue the significant and outstanding problems facing computer science both as a profession and as a field of study.

Allen entitled her keynote "Compiling for Performance," speaking most directly about the performance gap between software and hardware. As she gave a brief history of computers, she led the audience through hardware concurrency, massively parallel systems, today's common multicore computers, and ended with hugely multicore platforms already in development (there are processors in development that contain between 80 and 120 cores, if not more). Ambitious hardware, she said, requires equally ambitious compilers and software. The gap between software and hardware performance is due in large part, she asserted, to the lack of programming languages and compilers keeping pace with the multicore and hugely multicore processors in production and in development.

The problem of parallelism has been around since multiprogram systems. Hardware has aggressively added more parallelism to its capabilities, but software hasn't kept pace. Threading helps, as do compilers that can extract parallelism from the high level languages, but Allen was clear that so long as languages like C and C++ permitted easy access to pointers and memory, compilers would be hard-pressed to "disambiguate data references" and "transform the program to run more optimally." Simply put, C's introduction in 1973 was "a setback to easily deliver performance to the user. It derailed the advances made in compilers by letting the programmers play with the memory."

In the '70s, C was packaged with UNIX as an application-specific language for operating systems. "And then it became free and available to students and the rest is history. It was really a giant step backwards because compilers can do a much better job of that, but operating systems' [languages] are not the right thing to pass through a compiler," she said. C and C++ are application-specific languages (ASL) for operating systems, and any ASL for an operating system is challenging for a compiler; because of pointers, interrupt handlers, and other issues the compiler simply cannot optimize code well nor can it extract much, if any, parallelism. Trying to use those languages for general-purpose applications does not make sense since the low-level memory management is not needed and compilers could do a much better job than the programmer. Until programmers began using languages with pointers, Allen noted that the so-called best handwritten assembly code was routinely outperformed by compiler-generated assembly.

Her research effort in ubiquitous parallelism in the '80s and '90s was dubbed PTRAN. Returning to the theme of high-level languages that allowed the user to express his or her algorithm naturally, Allen eschewed C and the newer C++ since they were, by their own definitions, systems languages. Fortran, as old as it was (although revised significantly in 1977 and 1990), was still a powerful language for scientists and could be compiled in a way that aggressively extracted parallelism. So, too, could Java. Even PL/1 could be compiled to aggressively extract parallelism. Going back to a project from the '50s and '60s, Allen reiterated the power of application-specific languages with Alpha, a language used for the Harvest-Stretch computer for the NSA that boasted powerful pattern matching capabilities for a streaming-data computational model, which is now back in vogue. How friendly were these languages to the programmer? Using a similar language to Alpha, a gene-matching program for a DNA sequencing application could probably be accomplished in less than a page of code, according to Allen.

Allen has been active in mentoring women for years, including the late Anita Borg of the Anita Borg Institute for Women and Technology. Additionally, she has served on the boards of numerous professional and advocacy organizations, including the Computing Research Association (CRA), which has an arm devoted to the status of women in computing research (CRA-W). Testifying to the scarcity of women in science, Borg said Allen was her first and only female professor in graduate school, at NYU in the 1970s.

Allen indicated that the situation has not necessarily improved, neither since Borg attended graduate school nor when she joined IBM in 1957. "It's not very good. It was better for women when I started," Allen said. "Programming was considered, in some ways, a woman's job. That changed as the field became a profession and one was expected to [meet] certain requirements, and those requirements came from courses at engineering schools which were largely male at that time."

Allen added, "The high point for undergraduate women percentage-wise in computer science was 1985, and it's gone down since then. It's dismal. Though women are going into just about every other science, particularly biology, [they are not going into] computer science. My belief is that our field is not presenting itself very well to potential students." Allen believes one problem in the presentation of computer science as a discipline for women lies in the misbelief that creativity or teamwork is subsumed by long bouts of programming at a desk with little or no interaction with others. Men might not be so turned off by this concept, but many women interpret it as restrictive. They see no such barriers in the other sciences, especially life sciences, suggests Allen.

It is not just a dropoff among women in computer science. CS departments nationwide have been hemorrhaging undergraduates. The CRA reported that since 2000, undergraduate majors in computer science have decreased by 70%. This broader problem cannot be solved by marketing computer science differently, which is merely window-dressing for a more, and desperately needed, substantial set of changes.

"The curriculum is antiquated, there's no question about it," Allen asserted. "Why would freshman students want to stay up all night programming C? They can't quite see the value in that." Students I spoke with agreed. Some argued that theory-based courses were not presented in a way to make the knowledge relevant to real-world problems or to methods in which the theories could be applied (a malaise felt by many students from time immemorial). Some students spoke earnestly about adding more project-centric work to the curriculum, so that ideas and theories learned in class could be tested, observed, and tinkered with by the student to help retain the material and gauge its relevance. These changes may be part of the solution, but the argument over relevance and purpose will continue. Some aspects of an earned degree aren't seen as necessary by students, yet professors and even older students swear by their value. Conversely, curricula cannot be defended by the status quo; as scientists, the designers and crafters of a curriculum must be able to justify and defend the presence or lack of its components.

Are we at a critical point, or will it reach a critical point, to reinvigorate the various fields of computer science including computer engineering and information science? Allen believes globalization will shift some work, including research, offshore. It has already happened to an extent, but she does not believe we should fear a wholesale decampment of professionals and students fleeing the US: "It's very probable a lot of that will go overseas, probably at the more boring level." By implication, that which remains will be critical, interesting, and likely very challenging. As one example of an applied problem, Allen cited Wal-Mart (though she was quick to point out that she was by no means defending unrelated corporate practices and other controversies surrounding the company). "The specialization of the kind of applications one can enable and the power of those applications, such as the businesses' successes with integration like Wal-Mart's supply chain and [logistics] is a very complex system that allows them to cut costs everywhere. There's probably a whole series of things, right at this juncture, [when] one realizes how weak the actual systems sometimes are."

Though the discipline of computer science is relatively new, its roots can be found in a very old one; giving some historical perspective, Allen noted that "computer science has shrunk relative to the information sciences, which itself came out of the library sciences." Consider the structure of a library and the properties of a computer system: there is a set of knowledge that changes from time to time; it may require an external contributor to change the set of knowledge; interfaces must be well understood or apparently usable, otherwise the system and the knowledge contained therein will go unused; new knowledge is created from the old or spontaneously. Perhaps that last bit might not be true: Allen predicted a sudden change in the capability of technology, aided by the aforementioned ubiquitous parallelism in hardware as well as its partner in crime, sophisticated data-parallel software.

I asked her what she thought the most significant advancement in computer science, or technology in general, has been. Without hesitation she said, "Certainly the search engines. Google. Stunning, stunning, stunning. And it happened so fast, only twelve years old." I then asked which technology or advancement she thought would succeed but didn't. She named natural language translation, but noted it is not a pipe dream. For comparison she said, "AI isn't AI once you've achieved it." IBM has produced computers that go undefeated in chess, which Allen said were considered AI research projects until they actually accomplished their goals. There are now computers that go undefeated in checkers, and others in poker. Games, true, but each are huge computational challenges and they have been achieved. So with that success in mind, natural language translation and its twin, or close relative, natural language processing (NLP) no longer seem impossible. For other areas, such as sporting events and the Olympics, a set of rules governs the games and the positions or performances of players are the data; Allen asks a critical question, "Could a computer be a judge?"

Combining both NLP and search is the province of the latest challengers to Allen's favorite advancement. Startups like Powerset aim to provide search results based on natural language queries, a feat only possible since hardware became fast enough to provide the processing substrate and software became usable enough to enable high-level descriptions of algorithms to execute the tasks. Even so, there may be a limiting cost factor: Powerset requires 50 times the processing capability compared with so-called traditional search engines like Google and Yahoo to index, classify, and search the same documents. With the seemingly never-ending improvement in processor speeds, that limit may prove to be not much of a barrier.

One feature these new search engines have yet to achieve Allen predicts also will happen: knowledge creation. She said, "What's going to be really interesting is integrating different forms of information and creating knowledge." Advances in document classification and NLP should allow for richer results when a search is performed, as well as "offline" results, when no search is input from the user and the system analyzes its own set of facts and deductions to produce new relationships among and between the data. The current capability of Powerset, for example, might produce a list of relevant answers to a particular query (an example the company often gives is "Whom did IBM acquire?"), but in the form of results per-document. Constructing a single list as the response to the query from all available indexed documents is not yet available, but that sort of answer seems to be partly what Allen had in mind.

Allen moreover contends the ability of NLP to enable knowledge creation is critical to the success of ubiquitous computing. Ubiquitous computing is, in a sense, ubiquitous parallelism: intelligent devices and appliances operating simultaneously to deduce relationships between them, determine relationships between their collective data, and offer the results to the user in an effort to become more useful, more efficient, and more insightful to him or her. Today's monolithic search engines and desktop computers will eventually morph into ubiquitous intelligent devices and search tools.

Several years ago, Allen bucked the trend in computer science pedagogy by encouraging Java as a first programming language for undergraduates, compared with C or C++ as her peers espoused. Considering her life with compilers and concerns about performance, perhaps it is not surprising that Allen still suggests undergraduate students learn Java as their first programming language. And, yes, she still climbs mountains.

Biography

Daniel Alex Finkelstein is a PhD candidate in computer science specializing in computer architecture at Polytechnic University in New York. He holds a BS degree in materials science from Columbia University and an MS degree in computer science from Polytechnic University.