Perchè Frances E. Allen ha vinto la Turing Award 2006?*

Eugenio Moggi
DISI, v. Dodecaneso 35, 16146 Genova, Italy
moggi@disi.unige.it

1 Cosa è la Turing Award

Nel 1901, quando furono assegnati per la prima volta i premi Nobel, l'Informatica non esisteva. La *Turing Award* (Premio Turing) è stato istituito nel 1966 dalla *Association for Computing Machinery* (ACM), ed è considerato l'equivalente del Premio Nobel per l'Informatica.

L'ACM è una associazione con sede centrale a New York fondata nel 1947, anno in cui fù creato il primo calcolatore digitale a *programma memorizzato*. In base a quanto scritto sul sito web dell'ACM [1]:

- ACM è la più vecchia e la più grande associazione al mondo che si occupa degli aspetti educativi e scientifici dell'Informatica. Sin dal 1947 ha rappresentato un *forum* per lo scambio d'informazioni, idee e scoperte.
- Al giorno d'oggi, ACM serve una comunità di associati, tipicamente costituita da professionisti e studenti, operanti in imprese, università e enti governativi distribuiti in più di 100 nazioni.
- Lo scopo di ACM è di far progredire l'Informatica sotto l'aspetto scientifico e professionale, favorendo lo studio, sviluppo, costruzione ed applicazione di macchine per il trattamento dell'informazione.

Tra i premi assegnati dalla ACM, il premio Turing è il più prestigioso, ed è assegnato ad individui selezionati per i loro contributi, che devono essere di persistente e notevole importanza tecnico-scientifica per il settore informatico.

Il premio Turing include un finanziamento di 250.000 dollari (100.000 fino al 2006). Al contrario del premio Nobel, tale finanziamento non deriva dal lascito di un ricco benefattore, ma dalla sponsorizzazione di enti o industrie del settore informatico. Attualmente gli sponsor sono *Intel Corporation* e *Google Inc.*

Il premio Turing prende il nome dal matematico e logico inglese *Alan Turing* (1912-54), che negli anni '30 fù tra i fondatori della *Teoria della Calcolabilità*. In particolare, introdusse un *calcolatore ideale*, la "macchina di Turing", che forniva una definizione precisa e convincente di funzione calcolabile (su sequenze di caratteri). Turing fù anche un precursore dell'*Intelligenza Artificiale*. A lui si

*Il vincitore del premio Turing per l'anno n viene annunciato l'anno successivo.

deve l'idea di "test di Turing" per verificare se una macchina può esibire un comportamento "intelligente" al pari di un essere umano. Maggiori informazioni su Alan Turing, i cui contributi ed idee visionarie non si limitarono all'Informatica, si possono reperire in [9].
Se scorriamo la lista dei vincitori del premio Turing su Wikipedia [12] vediamo che spesso le motivazioni per l'attribuzione del premio riguardano contributi dati nell'abito dei *linguaggi di programmazione* e/o *compilatori*:

1966 Alan J. Perlis: for his influence in the area of *advanced programming techniques* and *compiler construction*.

2005 Peter Naur: for fundamental contributions to *programming language design* and the definition of Algol 60, to *compiler design*, and to the art and practice of computer programming.

2006 Frances E. Allen: for pioneering contributions to the theory and practice of optimizing compiler techniques that laid the foundation for modern *optimizing compilers* and *automatic parallel execution*.

Come spiegheremo in seguito, non è accidentale questa insistenza sui *linguaggi* di programmazione e gli strumenti ad essi collegati, quali i *compilatori* (detti anche traduttori). Infatti, essi si sono dimostrati essenziali per rendere i calcolatore versatili e fruibili anche da soggetti senza specifiche competenze informatiche.

2 Breve CV di Frances E. Allen

Le seguenti informazioni sono state riprese dalla biografia di Frances E. Allen pubblicata sul portale dell'ACM [2] e su Wikipedia [11]. Per avere un quadro più completo, con aneddoti e note personali, si consiglia la lettura dell'intervista rilasciata da Frances E. Allen nel 2003 [10].

1932 Nasce nello stato di New York.

1954 Ottiene un BSc in Mathematics dalla Albany State Teachers College.

La sua intenzione era di diventare un'insegnante.

1957 Ottiene un MSc in Mathematics dalla University of Michigan.

Poco dopo inizia l'attività di insegnante, ma oberata dai debiti contratti per poter studiare, decide di andare a lavorare per qualche tempo all'IBM (presso il T.J.Watson Research Center), per poterli ripianare. ... Rimase all'IBM per 45 anni. La IBM (International Business Machines) Corporation nel settore informatico risulta essere:

- tra le maggiori "corporation", i suoi interessi vanno dall'hardware alla consulenza ai clienti;
- la più longeva e quella detentrice di più brevetti;

– leader nel settore mainframe[1], e più di recente anche nel settore dei supercomputers[2].

1957 All'inizio le viene assegnato il compito di insegnare il FORTRAN a scienziati ed ingegneri dell'IBM. Il FORTRAN (FORmula TRANslation) è il primo linguaggio di programmazione *ad alto livello* di successo:

– nel 1954 all'IBM John Backus (Turing Award 1977) avvia il progetto;
– nel 1957 viene distribuito il primo compilatore, ma è accolto con scetticismo, sia per una certa inerzia mentale, sia perchè molti esperti (anche autorevoli come Von Neumann) ritenevano che avrebbe generato del codice eseguibile molto inefficiente;
– più recenti versioni del FORTRAN, e relativi compilatori, sono tuttora in uso per applicazioni scientifiche.

Essere entrata in IBM proprio quando veniva introdotto il compilatore FORTRAN ha avuto un profondo impatto sulla carriera di Allen. Al contrario di suoi colleghi di maggiore esperienza non aveva pregiudizi verso i compilatori, anzi doveva cercare di esaltarne i vantaggi.

Allen era quindi arrivata nel posto giusto al momento giusto per riconoscere prima di molti altri le potenzialità e le sfide poste dall'*High Performance Computing* (Calcolo ad Alte Prestazioni): fornire alte prestazioni, senza dover esporre l'architettura sottostante.

1966 Pubblica l'articolo "Program Optimization" [3], che getta le basi concettuali per l'analisi e la trasformazione sistematica dei programmi.

1970 Pubblica gli articoli "Control Flow Analysis" [4] e "A Basis for Program Optimization" [5], che forniscono il contesto per l'*analisi del flusso* e per avere *ottimizzazioni* efficienti ed efficaci.

1972 Pubblica con J.Cocke (Turing award 1987) l'articolo "A Catalog of Optimizing Transformations" [8], che fornisce la prima descrizione sistematica delle trasformazioni ottimizzanti.

Successivamente Allen lavora a compilatori sperimentali [7], che dimostrano la fattibilità dei moderni *compilatori ottimizzanti*.

1984 Dirige il progetto PTRAN [6], che affronta le sfide poste dai calcolatori paralleli e sviluppa vari concetti (p.e. *grafo delle dipendenze*) usati in molti *compilatori parallelizzanti*.

1995 Diviene presidente della IBM Academy of Technology, una struttura interna che fornisce consulenza tecnica all'IBM.

2002 Và in pensione dopo 45 anni di attività, con molte soddisfazioni e qualche "delusione" per alcune scelte "manageriali" da lei non condivise (vedi [10]).

L'ultimo ruolo ricoperto all'IBM è Senior Technical Advisor del Research Vice-President per Soluzioni, Applicazioni e Servizi.

[1] I mainframe sono calcolatori usati da grandi imprese o enti, che hanno bisogno di garantire un servizio affidabile e continuato (per uso interno o per terzi), p.e. banche, compagnie aeree.

[2] I supercomputers sono calcolatori in grado di risolvere problemi matematicamente complessi, p.e. previsioni meteoreologiche, in tempi accettabili.

Tra i riconoscimenti ricevuti da Allen i più significativi sono:

- 1989 prima donna ad essere nominata IBM Fellow;
- 2006 prima donna a vincere la Turing Award.

Nel 2000 IBM crea la *Frances E. Allen Women in Technology Mentoring Award*, per premiare quelle donne, che si sono distinte non solo per i loro contributi, ma anche per aver favorito una maggiore partecipazione femminile nei settori tecnologici.

3 Compilatori Ottimizzanti

Il premio Turing 2006 è stato assegnato a France E. Allen per

> i contributi pioneristici alla teoria e alla pratica delle tecniche di ottimizzazione che forniscono i fondamenti per i moderni *compilatori ottimizzanti* e *parallelizzanti*.

Questi contributi hanno notevolmente migliorato le prestazione dei programmi per calcolatori nel risolvere problemi, e hanno accellerato l'uso del calcolo ad alte prestazioni. In questa sezione si spiegherà cosa sono i compilatori ottimizzanti. Per fare ciò partiremo dalla distinzione tra hardware e software, per poi parlare di linguaggi di programmazione, interpreti (e macchine virtuali), e compilatori (detti anche traduttori).

3.1 Hardware e Software

In modo molto semplificato, possiamo dire che in un calcolatore, e più in generale in un sistema informatico:

- l'*hardware* (HW) è quella parte che si può prendere a calci;
- il *software* (SW) è quella parte contro cui si può solo imprecare.

Facendo un parallelo con un essere umano, l'hardware corrisponde alle parti anatomiche, mentre il software corrisponde al pensiero e alla memoria.
Dal punto di vista economico, la diffusione dei calcolatori ha portato ad una *rivoluzione informatica*, il cui impatto è confrontabile con quello della *rivoluzione industriale*. Nella rivoluzione industriale la novità è costituita dall'hardware (HW), p.e. il telaio meccanico

$$in \longrightarrow \boxed{\text{HW}} \longrightarrow out$$

che permette di realizzare un processo di trasformazione (di materiale grezzo in manufatti) senza richiedere competenze artigianali specifiche. Inoltre, la sostituzione della forza umana o animale con nuove forme di energia, permette di ottenere un considerevole aumento di produttività.
Nella rivoluzione informatica la novità è costituita dal software

$$in \longrightarrow \boxed{\text{HW+SW}} \longrightarrow out$$

che permette di modificare, cambiando il solo software (SW), il processo di trasformazione realizzato dalla combinazione HW+SW. Si ottiene così un considerevole aumento di flessibilità, poichè il software e l'informazione digitale hanno caratteristiche peculiari, diverse da quelle di materia ed energia:

- non si usura/consuma con l'uso
- è duplicabile e trasferibile a costo quasi nullo.

3.2 Linguaggi, Interpreti e Compilatori

In un calcolatore a *programma memorizzato* il software può essere identificato con i programmi, cioè informazioni che dicono cosa deve fare l'hardware. Più astrattamenti i programmi sono *descrizioni* in un opportuno linguaggio formale.

- Un linguaggio di programmazione L, definisce in modo preciso un insieme di programmi sintatticamente corretti (in Logica Matematica si definiscono in modo analogo insiemi di asserzioni ben formate).
- Dato un programma sintatticamente corretto P nel linguaggio L, il suo significato $[\![P]\!]_L$ è una funzione (dall'insieme dei possibili input a quello dei possibili output), che descrive l'effetto della trasformazione

  $$in \longrightarrow \boxed{\text{HW}+P} \longrightarrow out$$

  ottenuta *eseguendo* P su un calcolatore (HW) in grado di *capire* il linguaggio L. In questo caso si parla di *semantica input-output*.

In genere i calcolatori non operano direttamente su entità materiali, bensì su informazioni (dati). Nella stessa maniera il cervello di un essere umano, manipola informazioni provenienti dai nostri apparati sensoriali (dati di input) per generare informazioni dirette ai muscoli o altri organi (dati di output).
I programmi si possono classificare in due grosse categorie

- I *programmi applicativi*, che risolvono *problemi reali*, p.e.:
  - le previsioni meteo per domani
  - il calcolo degli interessi sui c/c di una banca per il mese corrente
  - la visualizzazione di una immagine ecografica
  - l'analisi strutturale (del modello matematico) di un edificio.
- I programmi che migliorano la *produttività* di chi sviluppa programmi (applicativi). Infatti, i programmi sono informazioni, e quindi possono essere trattati come dati (di input o di output) per altri programmi.

Esempi tipici di programmi che manipolano altri programmi sono

- un *interprete* P_{int} per L_1 scritto in L_0, cioè

  $$[\![P_{int}]\!]_{L_0}(P_1, in) = [\![P_1]\!]_{L_1}(in)$$

  ovvero $[\![P_{int}]\!]_{L_0}$ si comporta come un calcolatore che capisce L_1.

- un *compilatore* P_{comp} da L_1 a L_2 scritto in L_0, cioè

$$[\![[\![P_{comp}]\!]_{L_0}(P_1)]\!]_{L_2}(in) = [\![P_1]\!]_{L_1}(in)$$

  ovvero $[\![P_{comp}]\!]_{L_0}$ traduce un programma P_1 nel linguaggio L_1 in un programma P_2 nel linguaggio L_2 con lo *stesso* significato di P_1.

Per l'interprete P_{int} ed il compilatore P_{comp} il programma P_1 è un dato di input.
Intuitivamente, un interprete si comporta come un traduttore in simultanea, ovvero quello che viene tradotto è immediatamente eseguito, mentre un compilatore si comporta come un traduttore di un libro P_1, ovvero la traduzione P_2 può essere letta (ed eseguita) più volte e da lettori diversi.

- Se si usa la traduzione P_2 *molte volte*, conviene fare una *buona* traduzione.
- La *correttezza*, cioè $[\![P_2]\!]_{L_2} = [\![P_1]\!]_{L_1}$, è la proprietà che caratterizza un compilatore.
- Potremo dire che un compilatore è *ottimizzante*, se P_2 è il *migliore* tra i programmi P nel linguaggio L_2 tali che $[\![P]\!]_{L_2} = [\![P_1]\!]_{L_1}$.

La precedente definizione di compilatore ottimizzante è matematicamente sensata, ma purtroppo la Teoria della Calcolabilità ci dice che non esistono compilatori ottimizzanti per linguaggi di programmazione sufficientemente espressivi. Più precisamente, anche se esiste una funzione che mappa ogni P_1 nella sua migliore traduzione P_2, tale funzione non è *calcolabile*. Perciò dobbiamo ripiegare su un criterio empirico per definire un compilatore *ottimizzante*, p.e.

produce risultati *confrontabili* a quelli di un programmatore esperto.

Quindi la realizzazione di un compilatore ottimizzante diventa un problema ingegneristico, in cui si deve trovare un compromesso tra la *complessità* del compilatore e la *bontà* del suo output.
I linguaggi di programmazione possono essere classificati secondo vari criteri. Per i nostri scopi, cioè spiegare cosa è un compilatore ottimizzante, il criterio più appropriato è quello basato sul *livello di astrazione* rispetto all'hardware:

alto livello (utente) $\longrightarrow$ intermedio $\longrightarrow$ basso livello (hardware)

I linguaggi ad alto livello astraggono dall'hardware utilizzato per eseguirli (o mediante un interprete o dopo traduzione). Essi permettono di descrive e risolvere problemi con un formalismo molto simile a quello che userebbe un matematico o un esperto. Per questa ragione, i programmi scritti in questi linguaggi vengono detti *specifiche eseguibili*. Esempi di linguaggi ad alto livello sono

- I linguaggi funzionali. Un esempio generico di programma funzionale è

$$\begin{array}{ll} \text{let} & f_1(\overline{x}_1) = e_1 \\ & \ldots \\ & f_n(\overline{x}_n) = e_n \\ \text{in} & e \end{array}$$

esso *valuta* l'espressione e nel contesto di una dichiarazione locale[3] di funzioni f_i. La sintassi delle espressioni è specificata dalla seguente *definizione induttiva* (che copre tutti i possibili casi)

$e ::=$	x	nome di variabile x
$\mid$	$f(\overline{e})$	funzione f applicata alla sequenza di espressioni $\overline{e}$
$\mid$	let $\ldots$ in e	espressione e valutata in una dichiarazione locale $\ldots$ di variabili $x_j = e_j$ e/o di funzioni $f_k(\overline{x}_k) = e_k$

Il primo linguaggio funzionale è stato il LISP, introdotto alla fine degli anni '50 come linguaggio interpretato.

- I linguaggi logici. Un esempio di programma logico è

axioms	$a(x,y) \Leftarrow p(x,y)$
	$a(x,z) \Leftarrow a(x,y) \wedge a(y,z)$
	$p(adamo, abele)\ldots$
query	$\{x \mid a(adamo, x)\}$

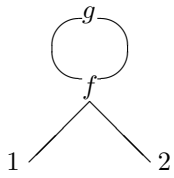
  esso *genera* tutti gli individui n tali che l'*istanza* $a(adamo, n)$ è conseguenza logica degli *assiomi condizionali*. Se $p(x,y)$ viene interpretato da "x è il padre di y", allora gli assiomi condizionali dicono che $a(x,y)$ vuol dire "x è un antenato di y". Quindi il programma logico ci restituisce tutti i discendenti di *adamo*. Il primo linguaggio logico è stato il PROLOG, introdotto all'inizio degli anni '70.

- I linguaggi d'interrogazione per basi di dati[4]. In questo settore vi è uno standard di fatto, ovvero il linguaggio SQL, introdotto nel 1974 e successivamente esteso. Il seguente esempio di interrogazione SQL

  SELECT nome, cognome FROM Impiegati WHERE salario> 1000

  specifica l'insieme di coppie $\{(i.n, i.c) \mid i \in \text{Impiegati e } i.s > 1000\}$ dove $i.n$ ($i.c$ e $i.s$) indica il nome (cognome e salario) dell'impiegato i.

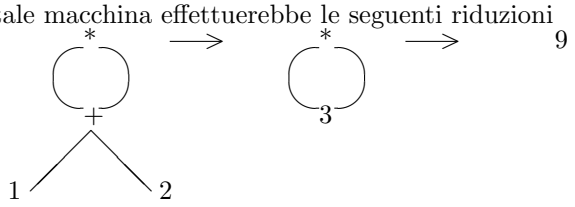
I Linguaggi intermedi astraggono dall'hardware, ma non sono pensati per essere usati direttamente dagli utenti. Essenzialmente un programma in un linguaggio intermedo è una *struttura dati* facilmente manipolabile da altri programmi (p.e. un interprete). Un esempio di linguaggio intermedio (in cui si traducono i linguaggi funzionali) è quello dei grafi annotati con *supercombinatori*. Per esempio, il programma funzionale let $x = f(1,2)$ in $g(x,x)$ verrebbe tradotto nel *grafo*



[3]Una dichiarazione, p.e. $x = e$, permette di usare il nome x al posto dell'espressione e. In una dichiarazione locale, (let $x = e$ in e'), solo all'interno di e'si può usare x al posto di e.

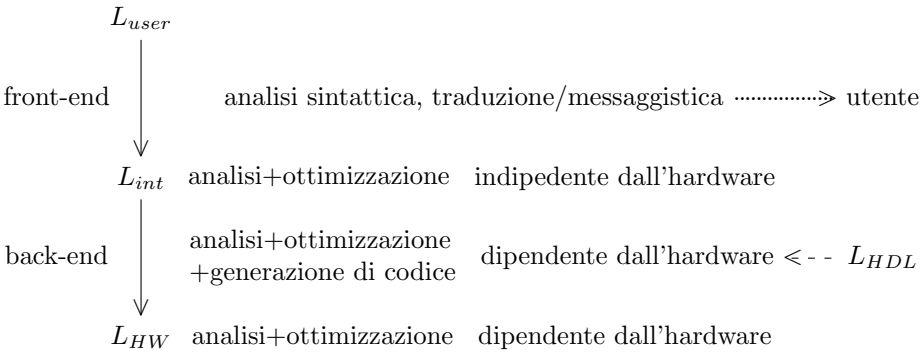
[4]Le basi di dati sono grandi raccolte di dati omogenei, p.e. le informazioni tenute dall'ufficio anagrafe di un comune.

In tale grafo si dimentica il nome x della dichiarazione locale. Inoltre, questo grafo può essere *valutato* da macchine virtuali che effettuano in modo efficiente la *riduzione di grafi*. Per esempio se f fosse la funzione somma e g quella prodotto, una tale macchina effettuerebbe le seguenti riduzioni



I linguaggi a basso livello sono interpretabili dall'hardware o direttamente (linguaggio macchina) o dopo un semplice traduzione (linguaggio assembler). Un tale linguaggio è strettamente legato ad un particolare calcolatore, p.e. in esso sono visibili i registri e le istruzioni della unità centrale (in gergo tecnico CPU).

Concludiamo descrivendo la struttura di un compilatore ottimizzante:



Tipicamente un compilatore è composto di due parti:

- Il *front-end*, che prende in input un *programma sorgente*, scritto nel linguaggio ad alto livello L_{user}, ed eventualmente delle direttive per la compilazione (p.e. quelle relative alle ottimizzazioni) e genera del *codice intermedio* in un linguaggio L_{int}.

  Il front-end può anche fallire, tipicamente perchè ha ricevuto un programma sorgente *sintatticamente sbagliato*, e comunque genera dei messaggi per l'utente, p.e. le ragioni che hanno impedito di generare codice intermedio, o degli avvisi di possibili *difetti* del programma sorgente.

- Il *back-end*, che prende in input il codice intermedio e genera del *codice oggetto*, in un linguaggio a basso livello L_{HW}, mirato ad un particolare calcolatore.

  In genere si è interessati a tradurre il programma sorgente in una pluralità di linguaggi macchina. Quindi il *back-end* ha bisogno di un'altro input, che identifichi il *linguaggio target* della compilazione ed eventuali caratteristiche dell'hardware che eseguirà il programma oggetto. Tale input non è altro che una espressione in un opportuno linguaggio L_{HDL} (per la descrizione dell'hardware).

In certi casi si vuole generare solo codice intermedio, che sarà successivamente interpretato. Nel descrivere la struttura di un compilatore abbiamo indicato i punti in cui è possibile effettuare delle ottimizzazione:

- è possibile trasformare il codice intermedio rimanendo nel linguaggio L_{int}, queste ottimizzazioni possono essere pensate come funzioni da L_{int} a L_{int}, e perciò possono essere composte;
- il *back-end* può sfruttare la descrizione dell'hardware (nel linguaggio L_{HDL}) per effettuare ottimizzazioni in fase di generazione del codice oggetto;
- infine è possibile trasformare il codice oggetto rimanendo nel linguaggio L_{HW} (analogamento a quanto fatto a livello di codice intermedio).

Allen è stata tra i pionieri dei compilatori ottimizzanti, dando importanti contributi sia alle tecniche di ottimizzazione indipendenti dall'hardware, sia a quelle per generare codice che sfruttasse le caratteristiche dei calcolatori paralleli.

4 Perle di Saggezza

Riportiamo alcuni brani presi da una intervista rilasciata da Frances E. Allen nel 2003 [10]. Essi contengono interessanti considerazioni basate sulla esperienza pluriennale di Allen. In seguito, per apprezzare l'attualità delle sue considerazioni, riportiamo anche alcuni estratti di recenti articoli attinenti al *multicore computing*[5], considerato una delle nuove *grandi sfide* per l'Informatica.

Per ogni brano estratto da [10] indichiamo un titolo e la pagina da cui è preso. Ogni brano è riportato in Inglese, con i passi più significativi sono evidenziato in **grassetto**. Dopo ogni brano è riportata la traduzione in Italiano, ed eventualmente dei commenti.

- Linguaggi Domain-Specific (pag 13)

  [In 1960] I ended up being the liaison with National Security Agency on the language that we were designing with them, which was **a language for code breaking which was a terrific match for the HARVEST machine** that **allowed the cryptologists to express their solutions in a very, very high level form**.

  [Nel 1960] ero la persona di collegamento con la National Security Agency per il linguaggio che stavamo progettado con loro, era **un linguaggio per la decifrazione in accoppiata perfetta con il calcolatore HARVEST** che **permetteva ai criptologhi di esprimere le loro soluzioni in una forma molto ad livello**.

- Compromessi tra hardware e software (pag 20)

  I ended up understanding so much about **the hardware and software trade-offs**, because, as a result of our experience with STRETCH, in the ACS project [1961-62], **we built the compiler before the machine, in order to be able to design the machine**.

  ---

[5] Si tratta di computers dove un unico chip contiene più CPU. Quindi in futuro un multicore computer delle dimensioni di un PC potrà avere la potenza di calcolo di un supercomputer.

Sono riuscita a capire così tanto sui **compromessi tra hardware e software**, poichè, a seguito della nostra esperienza con STRETCH, nel progetto ACS [1961-62], **costruimmo il compilatore prima del calcolatore, allo scopo di essere in grado di progettare il calcolatore**.

Occorre distinguere tra miglioramenti incrementali, che si possono portati avanti in modo indipendente, e discontinuità prodotte da nuove tecnologie o nuove applicazioni, che richiedono un ripensamento globale. Solo in progetti di punta (come ACS) si considerano soluzioni globali, in cui sia l'hardware che il software sono parametri modificabili. Nella maggior parte dei progetti (anche di ricerca) si hanno molti più vincoli.

- Compilatori (pag 21)

  [Compilers] **have to be considered at the same time, or ahead of time, because it really is ultimately how the performance gets delivered**... We've lost a lot of that.

[I compilatori] **devono essere considerati in contemporanea, o in anticipo, poichè è solo così che si ottengono le prestazioni**... Si è perso molto di questo.

La specializzazione delle competenze, e considerazioni di carattere economico limitano notevolmente la fattibilità di un *soluzione integrale* in cui hardware e software (in questo caso i compilatori) vengono co-progettati.

- Meta-compilatori (pag 26)

  I got involved with something called **experimental compiling system**. ... use the **compiler technology to make compiler writing easier**.

Fui coinvolta in qualcosa chiamato **sistema di compilazione sperimentale**. ... usare **la tecnologia dei compilatori per rendere più facile produrre compilatori**.

- Il fine ultimo (pag 26-27) – Questo passaggio identifica molto chiaramente quello che dovrebbe essere un obbiettivo strategico per l'Informatica.

  My goal - a goal I've not achieved - is **to support languages that are useful by the application writer**. Useful by the physicist, useful by the person who is solving a problem, and have **a language ... that is natural for the way that person thinks about the problem and the way the person want to express the problem**...

  That's what I've always felt was the ultimate role of compilers: to hide all the details of the hardware in the system, still exploit it, but hide that from users, so that they can get on with solving their problem and being comfortable with the results that they were getting, in terms of performance and cost time, and everything else. We've taken a bad direction in languages and compilers.

  What I wanted to do with the experiment on the compiler system, was to **build a system that would allow compilers to be built for**

automated, for multiple kind of source languages, or for multiple target machines.

Il mio scopo - che non ho raggiunto - è **di supportare linguaggi che siano utili per chi sviluppa applicazioni**. Utili per il fisico o la persona che sta risolvendo un problema, e avere **un linguaggio . . . che sia naturale per il modo in cui quella persona pensa relativamente al problema e per il modo in cui vuole esprimere il problema**. . .

Questo è ciò che ho sempre pensato fosse il fine ultimo dei compilatori: nascondere tutti i dettagli dell'hardware nel sistema, allo stesso tempo poterlo sfruttare, ma nascondere ciò agli utenti, in modo che essi possano risolvere il loro problema e essere sicuri dei risultati ottenuti, in termini di prestazioni e costi, e ogni altra cosa. Abbiamo preso un brutta direzione nel campo dei linguaggi e dei compilatori.

Quello che volevo fare con il sistema di compilazione sperimentale, era di **costruire un sistema che permettesse di costruire compilatori in modo automatizzato, per una pluralità di linguaggi sorgente, o per una pluralità di calcolatori**.

- Parallelismo (pag 29)

  We got into parallelism as a compiler problem. . . I had a PTRAN group, a fantastic group of young people. . . **The work was a huge stack of papers, which had vast influence on the direction of the field**.

Abbiamo affrontato il parallelismo come un problema di compialzione. . . Ho avuto un gruppo PTRAN, un gruppo fantastico di giovani. . . **Il lavoro ha prodotto un enorme mole di articoli, che hanno avuto una grande influenza sulla direzione presa dal settore**.

Riportiamo alcuni estratti di tre articoli del Marzo 2008 presi dalla rassegna stampa *ACM TechNews* (vedi sito web dell'ACM [1]). Tutti gli articoli sono attinenti al *multicore computing*, un tema di grande attualità e che pone una serie di nuove sfide. Per esempio, Dana S. Scott (Professore Emerito di Informatica, Filosofia e Logica Matematica alla Carnegie Mellon University) nel suo discorso per la EATCS Award 2007 identifica il multicore computing come la prossima **grande sfida per l'Informatica Teorica**. Infatti, i multicore computers rischiano di rimanere sottoutilizzati, se non si **co-progettano hardware e software** (linguaggi e applicazioni). Per ciascun articolo riportiamo il titolo, dove e quando è stato pubblicato, ed i brani più significativi in relazione alle considerazioni riprese dalla intervista di Allen. Dopo ogni brano è riportata la traduzione in Italiano.

Researchers Ready System to Explore Parallel Computing (EE Times - 13/03/08).

Researchers at the Univ. of California, Berkeley are nearly finished building the Berkeley Emulation Engine version 3 (BEE3), an **FPGA-based computer that could help researchers find a parallel programming model for advanced multicore processors**. BEE3 is intended to help researchers quickly

prototype processors with hundreds or thousands of cores and find new ways to program them.... **The system is the centerpiece of the Research Accelerator for Multiple Processors (RAMP) program**, a collaborative effort involving Berkeley, Microsoft, Intel, and five other U.S. universities, including MIT and Stanford....

I ricercatori all'Univ. della California a Berkeley hanno quasi completato la costruzione del emulatore Berkeley versione 3 (BEE3), un **calcolatore basato su FPGA**[6] **che potrebbe aiutare i ricercatori a trovare un modello per la programmazione parallela per processori multicore avanzati**. BEE3 dovrebbe aiutare i ricercatori a realizzare rapidamente prototipi di processori con centinaia o migliaia di core e trovare nuovi modi per programmarli.... **Il sistema è la componente centrale del programma RAMP**, un progetto collaborativo che coinvolge Berkeley, Microsoft, Intel, e cinque altre università americane, incluse MIT e Stanford....

Industry Giants Try to Break Computing's Dead End (New York Times - 19/03/08).

Intel and Microsoft yesterday announced that they will provide 20 million USD over five years to two groups of university researchers that will work to design a new generation of computing systems. **The goal is to prevent the industry from coming to a dead end that would halt decades of performance increases in computers**.... Each lab will work to **reinvent computing by developing hardware, software, and a new generation of applications powered by computer chips containing multiple processors**. The research effort is partially motivated by an increasing sense that **the industry is in a crisis because advanced parallel software has failed to emerge quickly**. The problem is that software needed to keep dozens of processors busy simultaneously does not exist....

Ieri Intel e Microsoft hanno annunciato che offriranno 20 milioni di dollari in cinque anni a due gruppi di ricercatori universitari che lavoreranno alla progettazione di una nuova generazione di sistemi di calcolo. **Lo scopo è di impedire all'industria [informatica] di trovarsi in un vicolo cieco che arresti decenni di aumenti nelle prestazioni dei calcolatori**.... Ogni laboratorio lavorerà per **reinventare l'Informatica attraverso lo sviluppo di hardware, software, e una nuova generazione di applicazioni realizzate con chips contenenti una molteplicità di processori**. Lo sforzo di ricerca è in parte motivato da una crescente sensazione che **l'industria è in una crisi perchè non si è avuta un rapido emergere di software parallelo avanzato**. Il problema è che non esiste del software capace di sfruttare simultaneamente dozzine di processori....

Making Parallel Programming Synonymous with Programming (HPC Wire - 21/03/08).

Academic experts are engaged in efforts to **transform mainstream pro-**

[6] FPGA è l'acronimo di Field Programmable Gate Array, si tratta di dispositivi digitali le cui funzionalità sono programmabili via software.

gramming by forcing multiple processing units to cooperate on the performance of a single task, which is being funded by Intel and Microsoft.. . . Leading academic teams will focus on **developing an effective methodology for multicore processor programming**,. . . The UC center's software work concentrates on two different layers,. . . The productivity layer will employ **abstractions to conceal much of the complexity of parallel programming**, while the efficiency layer will allow experts to **retrieve the details for maximum performance**.

Esperti accademici sono impegnati in sforzi per **trasformare la programmazione convenzionale forzando una pluralità di processori a cooperare nell'esecuzione di un singolo compito**, con finanziamenti da parte di Intel e Microsoft.. . . Gruppi accademici di punta si concetreranno sullo **sviluppo di una metodologia efficace per la programmazione dei processori multicore**,. . . Il lavoro sul software del centro UC si concentra su due differenti livelli,. . . Il livello di produttività userà **delle astrazioni per nascondere il grosso delle difficoltà inerenti alla programmazione parallela**, mentre il livello di efficineza consentirà agli esperti di **ottenere i dettagli necessari per massimizzare le prestazioni**.

5 Conclusioni

Il premio Turing a Frances E. Allen ha una doppia valenza:

1. un premio alla carriera in un settore *di nicchia*, ma *strategico*, come quello del Calcolo ad Alte Prestazioni;
2. un riconoscimento della rilevanza di sue *idee datate*, per affrontare una grande sfida come quella del *multicore computing*.

In relazione all'ultimo punto, l'*approccio integrato* sostenuto da Allen, in cui hardware, software e compilatori vengono co-progettati, appare la via più promettente per rendere fruibile il *multicore computing* a chi sviluppa applicazioni.

References

[1] ACM. Sito Web dell'ACM. `http://www.acm.org`.

[2] ACM. First Woman to Receive ACM Turing Award, 2007. `http://www.acm.org/press-room/news-releases-2007/fran-allen/`.

[3] F.E. Allen. Program optimization. *Annual Review of Automatic Programming*, 5:239–307, 1966.

[4] F.E. Allen. Control flow analysis. *SIGPLAN Notices*, 5(7):1–19, July 1970.

[5] F.E. Allen. A basis for program optimization. In *IFIP Congress (1)*, pages 385–390, 1971.

[6] F.E. Allen, M. Burke, P. Charles, R. Cytron, and J. Ferrante. An overview of the PTRAN analysis system for multiprocessing. *Journal of Parallel and Distributed Computing*, 5(5):617–640, October 1988.

[7] F.E. Allen, J.L. Carter, J. Fabri, J. Ferrante, W.H. Harrison, P.G. Loewner, and L.H. Trevillyan. The Experimental Compiling System. *IBM Journal of Research and Development*, 24(6):695–715, November 1980.

[8] F.E. Allen and J. Cocke. A catalogue of optimizing transformations. In R. Rustin, editor, *Design and Optimization of Compilers*, pages 1–30. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

[9] A. Hodges. The Alan Turing Home Page. `http://www.turing.org.uk/turing/`.

[10] P. Lasewicz. Frances Allen's Oral History Interview, 2003. `http://www-03.ibm.com/ibm/history/witexhibit/pdf/allen_history.pdf`.

[11] Wikipedia. Frances E. Allen. `http://en.wikipedia.org/wiki/Frances_E._Allen`.

[12] Wikipedia. Turing Award. `http://en.wikipedia.org/wiki/Turing_Award`.