Christopher Strachey and the Development of CPL

by

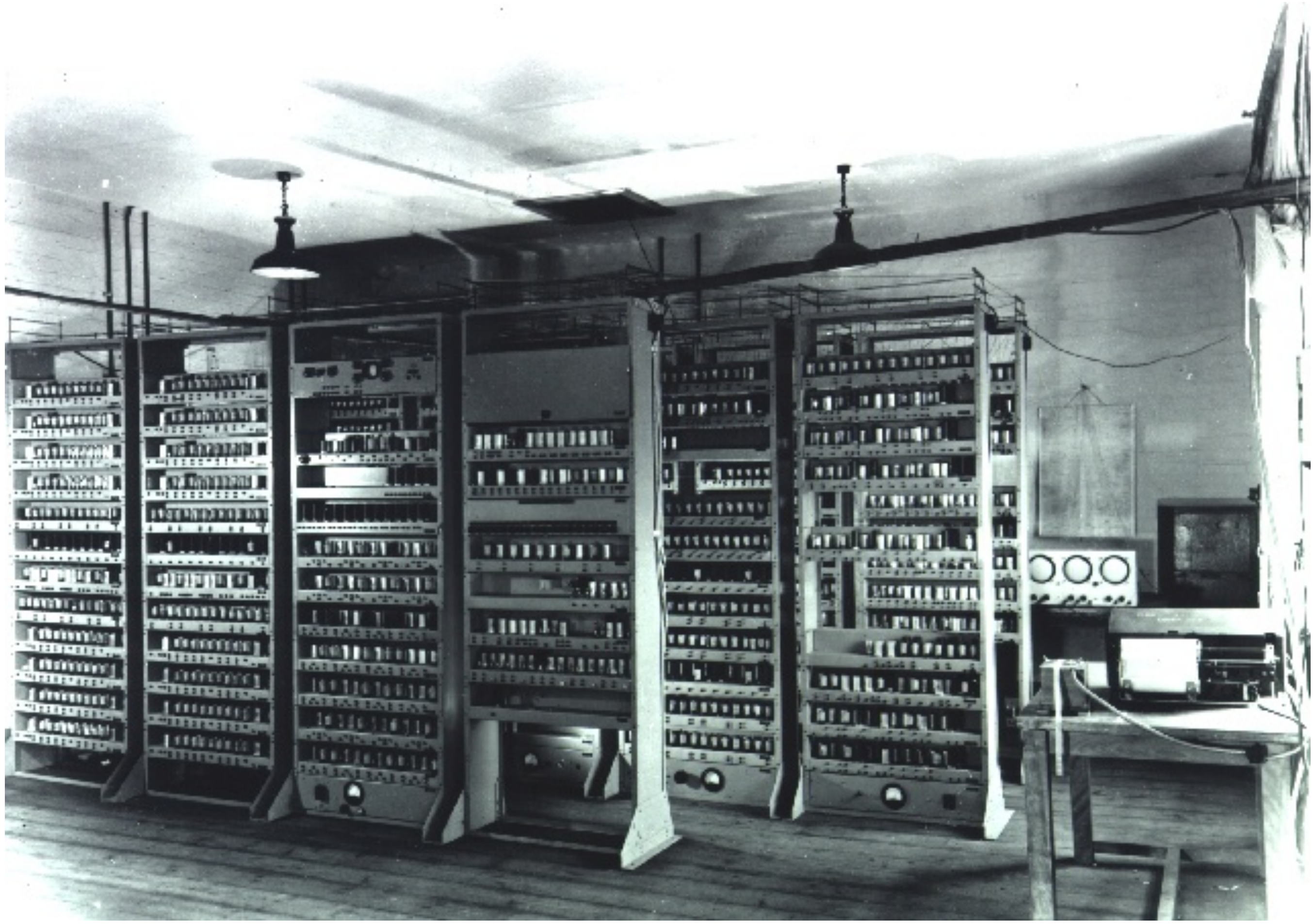
Martin Richards

www.cl.cam.ac.uk/users/mr10

The CPL Project

- The CPL Project ran from about 1962 to the end of 1966.
- I was involved as a Research Student from October 1963 to November 1966.
- The computing facilities in Cambridge at the time were as follows.

EDSAC 1949-1958



EDSAC Details

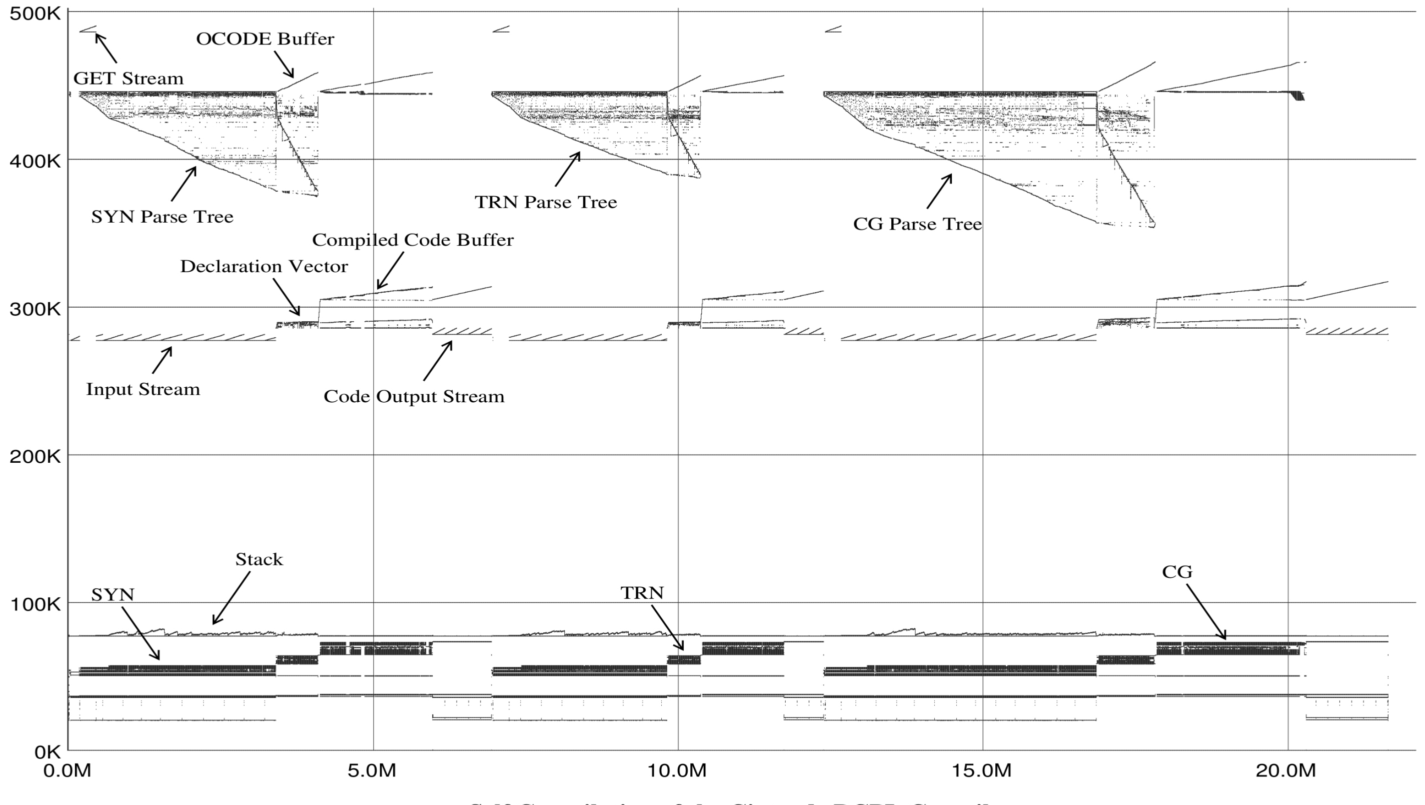
- Mercury delay line memory of 512 35-bit words.
- Words could hold two 17-bit instructions.
- Fairly slow clock rate, 500kHz.
- It had a 71-bit accumulator.
- Primitive instruction set but good for high precision arithmetic.
- Contributed to at least two Nobel Prizes.

Edsac 2 1958-1965

Edsac 2 Details

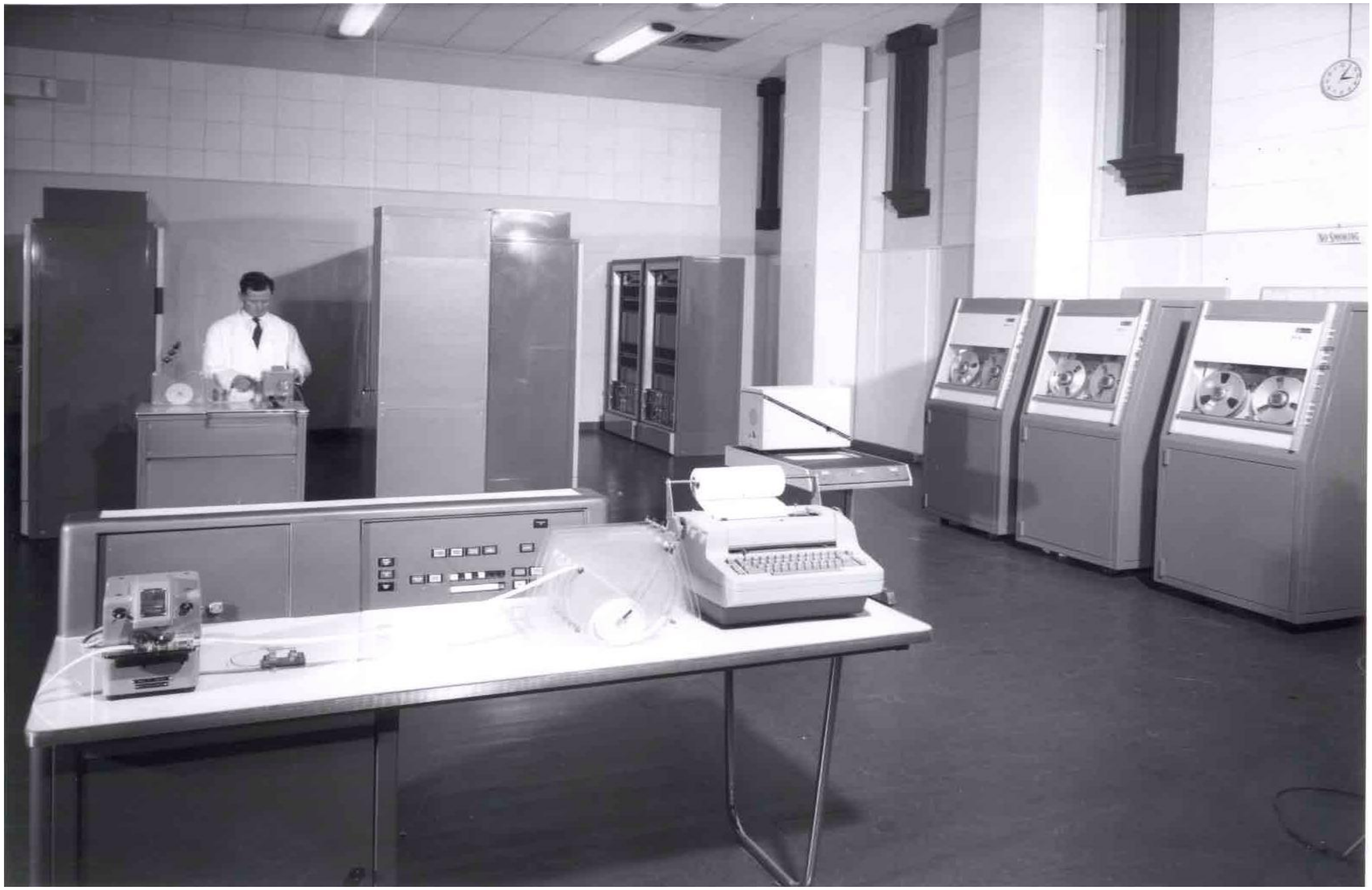
- Bit-sliced machine with 2048 20-bit words controlled by a micro program.
- User friendly machine code.
- Could perform 40-bit integer and floating point arithmetic.
- Extra 16K words of memory added in 1962.
- The fifth bit of the effective address of instructions was connected to a loudspeaker. Users found this useful.

BCPL Self Compilation



Self Compilation of the Cintcode BCPL Compiler

Titan 1965-1973



Titan Details

- Transistor machine with a core memory of 32K 48-bit words.
- 128 24-bit index registers.
- 48-bit instructions.
- It was a cut down versions of the Ferranti Atlas Computer.
- When first installed it had no assembler, no operating system and no compilers.

Development of CPL

- Initial ideas started in 1961 by Strachey and Wilkes.
- The language was to be an extension of ALGOL 60 including the good features and removing or modifying features that were hard to implement efficiently.

CPL Name Change

- CPL changed from Cambridge Programming Language to Combined Programming Language when the University of London Institute of Computer Science joined the project.
- They had a full blown Atlas Computer.

CPL Design Committee

- From Cambridge: Christopher Strachey, Peter Landin, David Hartley, David Park, David Barron.
- From London: John Buxton, Eric Nixon, George Coulouris.
- As a Research Student involved with CPL, I was able to attend most of the meetings.

CPL Design Meetings

- Usually in Cambridge or in Strachey’s house in London.
- Lively animated discussions trying to make compromises between the wishes of the different members.

CPL Design Principles

- Firm Mathematical Foundation.
- Generality.
- Ease of description.
- Only include features that were useful.
- Only include features that could be implemented efficiently.

CPL Features

- Recursive and non recursive functions
- Modes of calling: value, ref and subst
- L and R values, L and R mode evaluation
- Fixed and free functions
- Automatic deduction of data types
- Allow declarations to be qualified by other declarations using **in** and **where**
- And many more

Automatic Type Deduction

- Data types were little understood at the time. For instance the type of a function did not include the types and modes of calling of their arguments
- Much later Robin Milner received the Turing Award in 1991 partly for the superb polymorphic type system of ML

ML Types

- The type of every variable and expression can be deduced in finite time by the ML compiler, but only just. Consider

```
fun a x y = y x x;
fun b x = a(a x);
fun c x = b(b x);
fun d x = c(c x);
fun e x = d(d x);
fun f x = e(e x);
```

The Cambridge CPL Compiler

- This compiler was developed between 1963 and 1966 and halfway into the project we had to move from Edsac 2 to Titan
- We wrote the compiler in a subset of CPL and hand translated it into a macro calls that could be expanded easily into code for either Edsac 2 or Titan

GPM

- Strachey invented the truly wonderful macrogenerator GPM for the purpose
- I will describe a variant of GPM called BGPM that uses the ASCII characters set rather that the Flexowriter code used at the time. BGPM also corrects a subtle bug in GPM but is otherwise essentially the same.

BGPM

- Typical macro call: `[aaa,bbb,ccc]`
- `aaa` is looked up in an environment of defined macros, input temporarily comes from the body of macro `aaa`.

BGPM Example

```
[def,hi,{Hello #1}]’
[hi,Sally]
[hi,James]
```

This generates the following:

```
Hello Sally
Hello James
```

BGPM Example

```
[def,counter,0000]’
[def,inc,{[set,#1,[eval,[#1]+1]]}]’
[inc,counter]cointer = [counter]
[inc,counter]counter = [counter]
```

This generates:

```
counter = 1
counter = 2
```

BGPM Conditionals

If a definition is made in the argument list of a macro call, the definition is removed when the call completes.

This allows macros such as `ifeq` to be defined. You can even define `prime` so than `[prime,100]` expands to `541`.

let rec Fact(n) = n=0 -> 1, n * Fact(n-1)

```
[Prog,Fact]
[Link]
[LRX,1]
[LoadC,0]
[Eq]
[JumpF,2]
[LoadC,1]
[End,1,1]
[Label,2]
[LRX,1]
[LoadC,1]
[Sub]
[Fn,Fact]
[LRX,1]
[Mult]
[End,1,1]
```

Effect of using GPM

All values were the same size.
Functions were represented by just their entry points.
All arguments were called by value.
Memory consisted of consecutively numbered cells.
There were macros to obtain the addresses of variables.
There were macros for indirect access to memory.

The subset of CPL we used was a prototype version of BCPL, but we still felt we were still programming in CPL.

The macro code was semantically similar to the intermediate code OCODE used in BCPL compilers.

writef

`writef` was a result of our experience using GPM.

The first argument of `writef` is a format very similar to the body of a macro, and the remaining arguments are output in turn by substitution items in the format, such as `%c`, `%s` and `%n`.

printf

C adopted a variant of `writef` called `printf`.

Since the much loved function `printf` has a variable number of arguments which can each have different types, it is difficult to provide a similar function in strictly typed languages such as ML or Java.

PAL

Strachey visited Jack Wozencraft at MIT who was designing a new course to teach first year students the principles of programming.

He arranged for both Peter Landin and myself to move to MIT to help design and implement a form of sugared λ-calculus based on ISWIM.

This resulted in PAL which used Peter’s SECD machine.

OS6

Strachey was enthusiastic about BCPL and its portability to the extent he caused it to be implemented on the KDF-9 at Oxford, and also on the Modula 1 at the Programming Research Group in Banbury Road.

He also implemented an operating system in BCPL called OS6 which contained many innovations such as the way I/O streams worked.

Tripos

Later I lead a team that implemented the portable operating system Tripos.

BCPL and Tripos has been used by Ford in many of its plants since 1982 to control the factory floor.

A demonstration version of their system runs happily on a Raspberry Pi.

Conclusion

I am deeply indebted to Christopher for his influence when I was a Research Student and for arranging that I could go to MIT.

Without him and CPL, BCPL would not have been developed and it would not have been seen by Ken Thompson.

Ken's language B would not have been developed.

His collaboration with Dennis Ritchie would not have created C.

Even C++ and Java might not have been created.