UNIVERSITY OF CAMBRIDGE

SPACE'01: Imperial College, 15–16 Jan 2001

Statically Allocated Systems

Alan Mycroft

Computer Laboratory, Cambridge University

AT&T Laboratories Cambridge

`http://www.cl.cam.ac.uk/users/am/`

15 Jan 2001

SAFL

UNIVERSITY OF CAMBRIDGE

- Language of the FLaSH project (Functional Languages for Synthesising Hardware) we have developed at AT&T Laboratories Cambridge.
- First-order call-by-value functional language
- Syntactically restricted to be statically allocatable (only recursion is tail recursion)—well-suited to hardware synthesis
- Parallel evaluation of tupled expressions (arguments)
- Shared resources protected by semaphores (arbiters)

Why good for hardware?

- Static allocation is exactly what hardware does (any form of dynamic addressing is hard—"von Neumann bottleneck").
- Parallelism (from call-by-value) is what hardware provides lots of.
- Resource aware: the *compiler* treats functions as hardware units; two calls to one function represent a shared unit (protected by semaphore) or can duplicate function for two unshared units.
- Separate *transformer* to map within area-time spectrum.
- We have built a hardware compiler which translates SAFL to circuit designs (hierarchical RTL Verilog).

  Promising results: 3-4 page spec. of a commercial processor.

SAFL Language

- c constants,
- x variables (`let` and formal parameters)
- a primitive functions
- f user-defined functions

Terms e and programs p given by:

$$
\begin{array}{rcl}
e & ::= & c \mid x \mid \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mid \texttt{let } \vec{x} = (e_1, \ldots, e_k) \texttt{ in } e_0 \\
 & & \mid a(e_1, \ldots, e_{arity(a)}) \mid f(e_1, \ldots, e_{arity(f)}) \\
p & ::= & \texttt{fun } f^1(\vec{x}) = e_1 \texttt{ and} \ldots \texttt{and } f^n(\vec{x}) = e_n
\end{array}
$$

General calls, but only (first-order) tail recursion (*tail-regular*).

Odersky's Functional Nets

UNIVERSITY OF CAMBRIDGE

A programming language based on the join calculus.

- Functional syntax:

```
def f(n) = if (n==0) then 1 else n*f(n-1)
```

- Fork-join concurrency:

```
def f(n) & g(m) = n+m & g(n+m)
```

`f(n)` returns sum of callers' arguments (including `n`).

Aspects in common with Coloured Petri Nets.

Some similarities to CSP or value-passing CCS—components have state and may communicate—better than SAFL for hardware?

Static Functional Nets

UNIVERSITY OF CAMBRIDGE

New research. Extend the ideas in [GoltzMycroft 84] (representability of CCS terms as finite Petri ST-nets).

Idea:

- A Functional net program is statically allocatable if its recursion is
  - tail-regular
  - fork-non-expanding (no more forks than joins in recursive loop)
- Can we do better?

Higher order PR is not Statically Allocatable


```
fun r(x) = ⟨some function depending on x⟩
fun f(x,g) = if x=0 then g(0)
                    else f(x-1, r(x) ∘ g)
```

Thanks to Martin Hoffman for this example.

Neil Jones' JFP paper says more.