Graph Reduction Hardware Revisited

Rob Stewart (R.Stewart@hw.ac.uk) [1] Evgenij Belikov [1] Hans-Wolfgang Loidl [1]
Paulo Garcia [2]

14th June 2018

[1]Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, UK

[2]United Technologies Research Center
Cork, Republic of Ireland

United Technologies Research Center

HERIOT WATT UNIVERSITY

FPUs on FPGAs?

- **CPUs** for *general purpose* software
- **GPUs** for *numeric computation*
- Growth of domain specific *custom hardware*
  - *e.g.* TensorFlow ASIC chip for *deep learning*
- How about **FPUs**? (Functional Processing Units)
  - **Goal:** accelerated, efficient graph reduction
  - **Deployment:** Amazon F1 *Cloud instances* include FPGAs
  - **Motivation:** widening use of functional languages

Graph Reduction

1. *Write programs in a **big language** e.g.* Haskell
2. *Compile to a **small language***
   - *e.g.* Haskell $\rightarrow$ GHC Core $\rightarrow$ hardware backend
   - Lambda terms

x	variable
$(\lambda x.M)$	abstraction
$(M\ N)$	application

3. Computation by *reduction*

   $(\lambda x.M[x]) \xrightarrow{\alpha} (\lambda y.M[y])$ α conversion

   $(\lambda x.M)\ E \xrightarrow{\beta} (M[x := E])$ β reduction

$$(\lambda y.y + 1)\ 3 \xrightarrow{\beta} (3 + 1)$$

Historic Graph Reduction Machines

- **Graph reduction machines in 1980s** *e.g.*
  - GRIP (Imperial College), ALICE (Manchester)
- **15 year conference series**
  - *Functional Programming Languages and Computer Architectures*
- **Dedicated workshop in 1986**
  - *Graph Reduction, Santa Fé, New Mexico, USA.*
    Springer LNCS, volume 279, 1987

Graph Reduction Hardware Abandonment

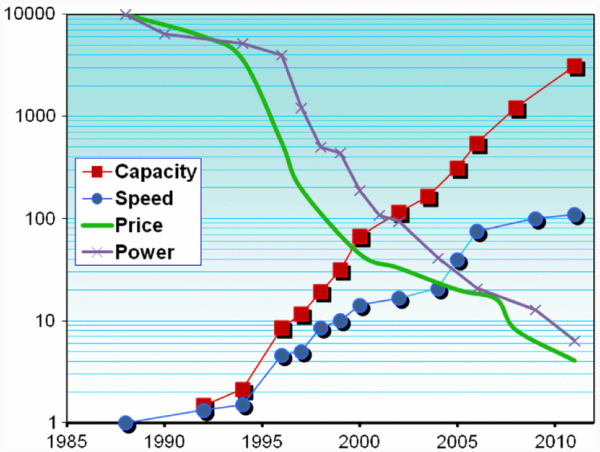
> *Programmed reduction systems are not so elegant as pure reduction systems, but offer the advantage that we can make use of technology developed over the last 35 years to implement von Neumann based architectures.*
>
> *Richard Kieburtz, 1985*

- Abandoned in favour of **commodity-off-the-shelf (COTS)** processors *e.g.* Intel/AMD CPUs
  - Custom hardware took years to build
  - *Free lunch:* clock frequencies speedups
  - Just build a compiler + runtime system **in software**

Graph Reduction Hardware Resurgence

> *Current RISC technology will probably have increased in speed enough by the time a [graph reduction] chip could be designed and fabricated to make the exercise pointless.*
>
> *Philip John Koopman Jr, 1990*

- **Historic drawbacks no longer hold thanks to FPGAs**
  - hardware development time reduced
  - design iteration: FPGAs are reconfigurable



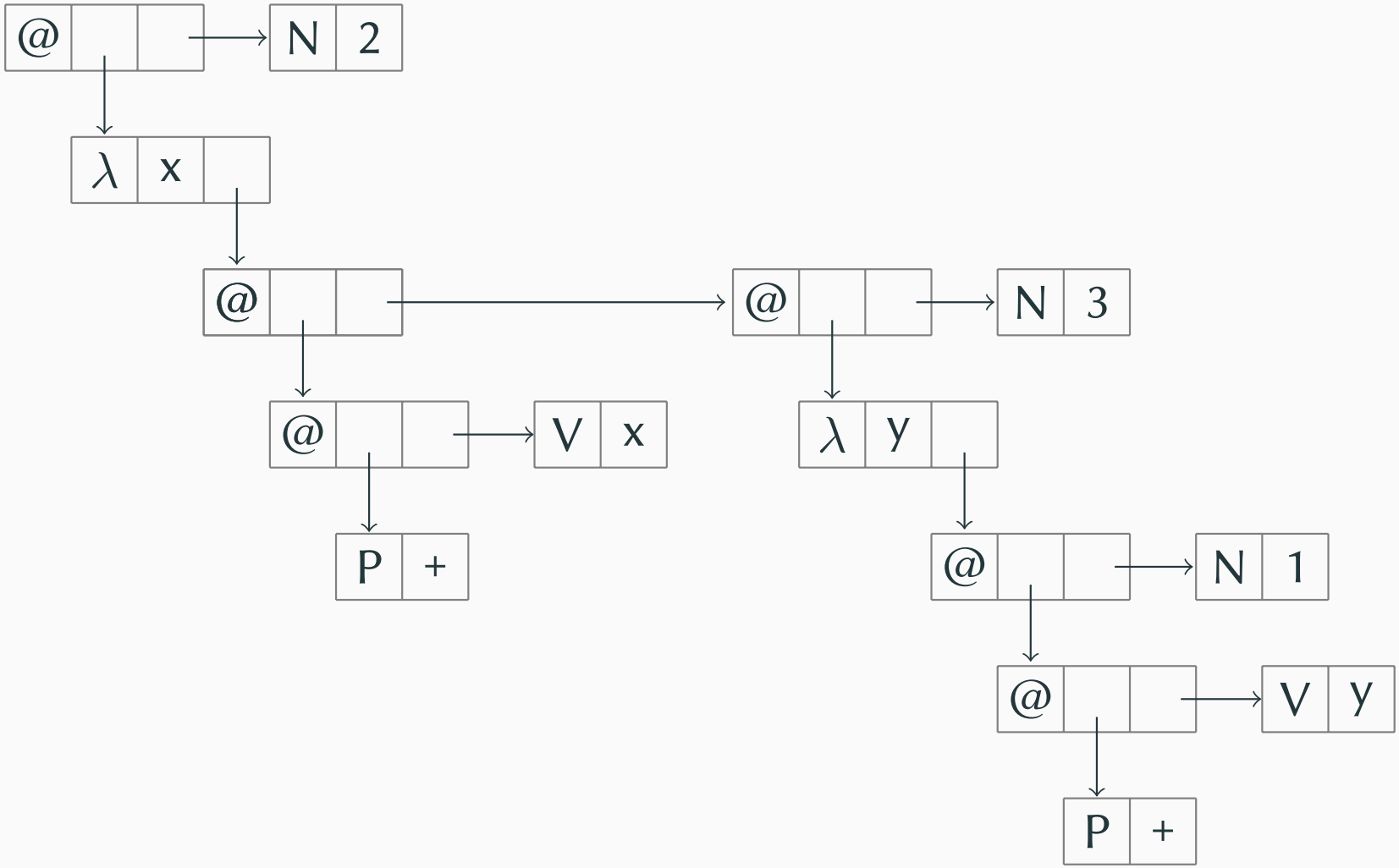
- Hardware trend to ***more space*** *rather than more performance*

"Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology", Stephen M. Trimberger, IEEE, 2015.

Graph Reduction

Concrete Graph Representation

$$(\lambda x.(x + ((\lambda y.y + 1)\ 3)))\ 2$$



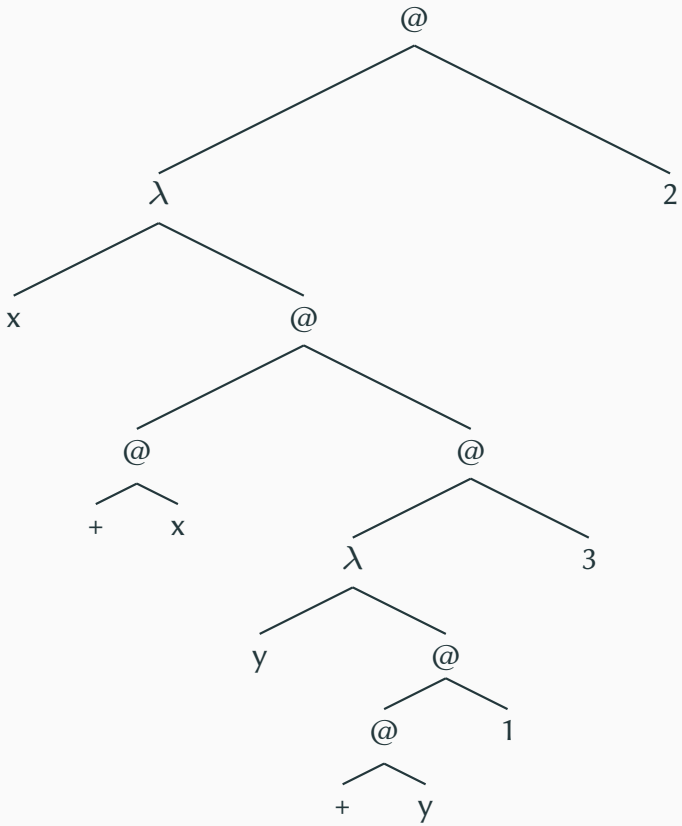
"Possible Concrete Representations", Chapter 10 of *The Implementation of Functional Programming Languages*. S. P. Jones, 1987.

Evaluation with β-reduction

let's *compute* $(\lambda x.(x + ((\lambda y.y + 1)\ 3)))\ 2$

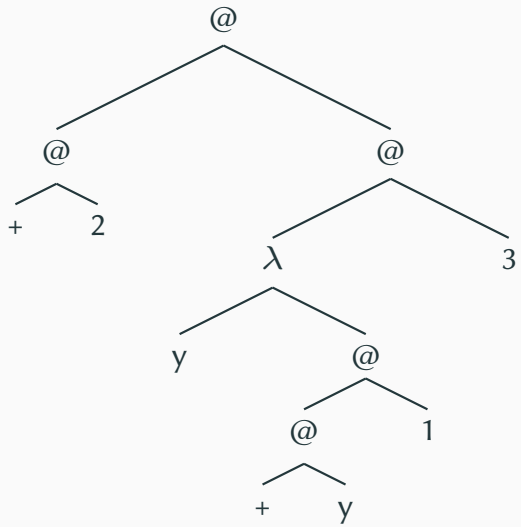
$$(\lambda x.M)N \xrightarrow{\beta} M[x := N]$$

$(\lambda x.(x + ((\lambda y.y + 1)\ 3)))\ 2$



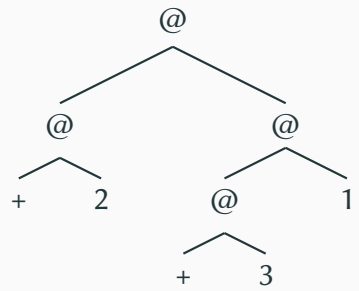
$\Rightarrow$

$(2 + ((\lambda y.y + 1)\ 3))$



$\Rightarrow$

$(2 + (3 + 1))$



Parallel Graph Reduction

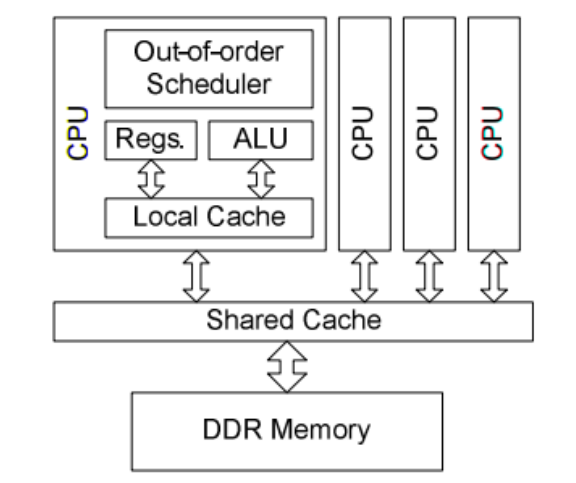
Parallel Graph Reduction in Software

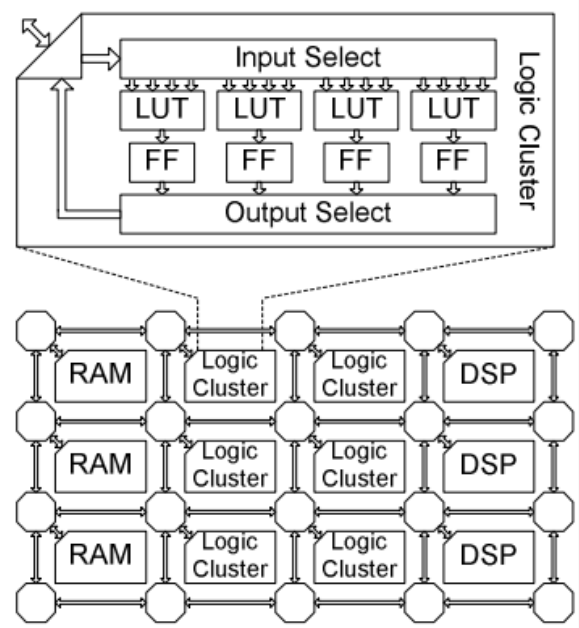
> *I wonder how popular Haskell needs to become for Intel to optimise their processors for my runtime, rather than the other way around.*
>
> *Simon Marlow, 2009*

- *par/pseq* to enforce evaluation order
- Parallel graph reduction with *par*, *e.g.*
  - `f `par` (e `pseq` (e + f))`
- Multicore: instructions for sequential reductions in each core
- *Distributed* parallel graph reduction: GUM supports *par/pseq*

Graph Reduction on FPGAs

FPGAs versus CPUs





CPUs: heap in DDR memory, sequential β reduction in each core

Idea: *soft processor on an FPGA for parallel graph reduction.*

A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation. D Thomas et al. Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, 2009.

Reduceron: Graph Reduction with Templates on FPGAs

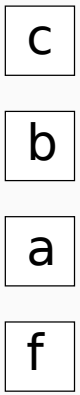
- Uses *template instantiation*, substitutes arguments into bodies
- F-lite functions compiled to templates
- Large reduction steps, avr. 2 reductions for function application
- 6 reduction operations each cost only 1 cycle because
  - parallel memory transactions
  - wide memories

"**The reduceron reconfigured and re-evaluated"**, Matthew Naylor and Colin Runciman
Journal of Functional Programming, Volume 22 Issue 4-5, 574-613, 2012.

Reduceron: reduction with templates

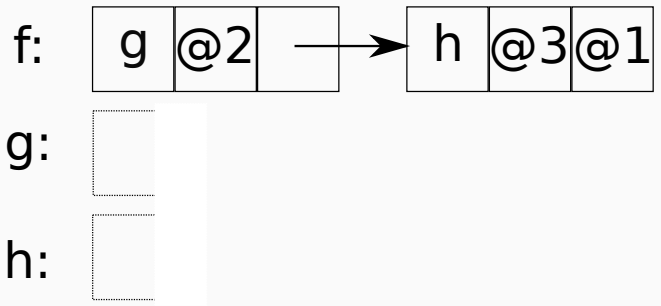
```
f ys x xs = g x (h xs ys)
```

Stack



Heap

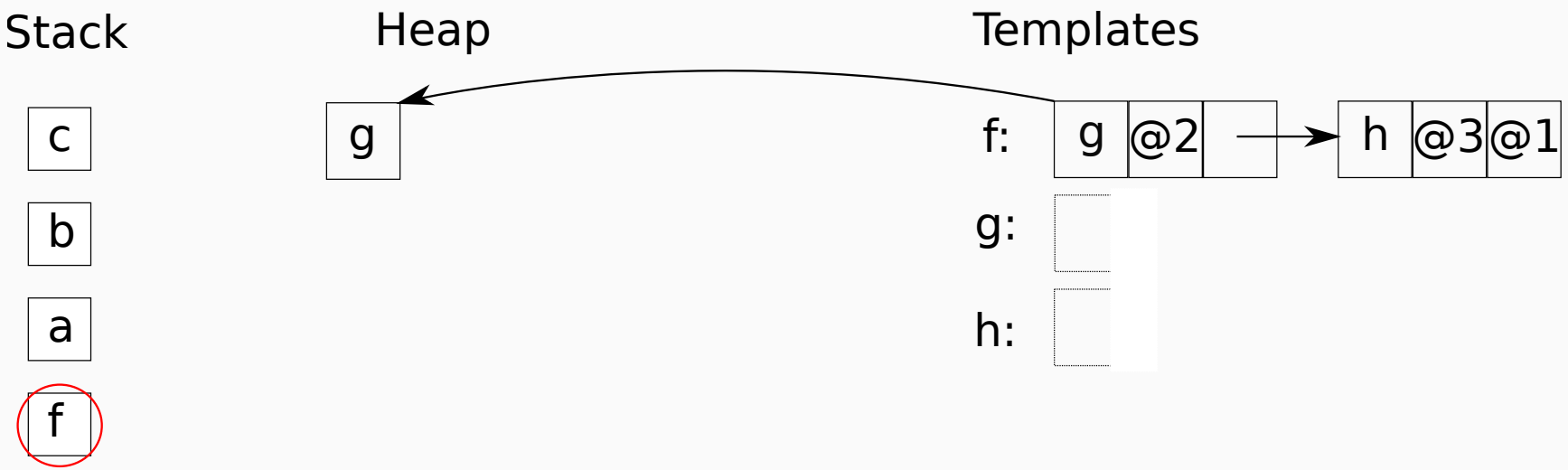
Templates



From “Dynamic analysis in the Reduceron”, Matthew Naylor and Colin Runciman, University of York, 2009.
https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf

Reduceron: reduction with templates

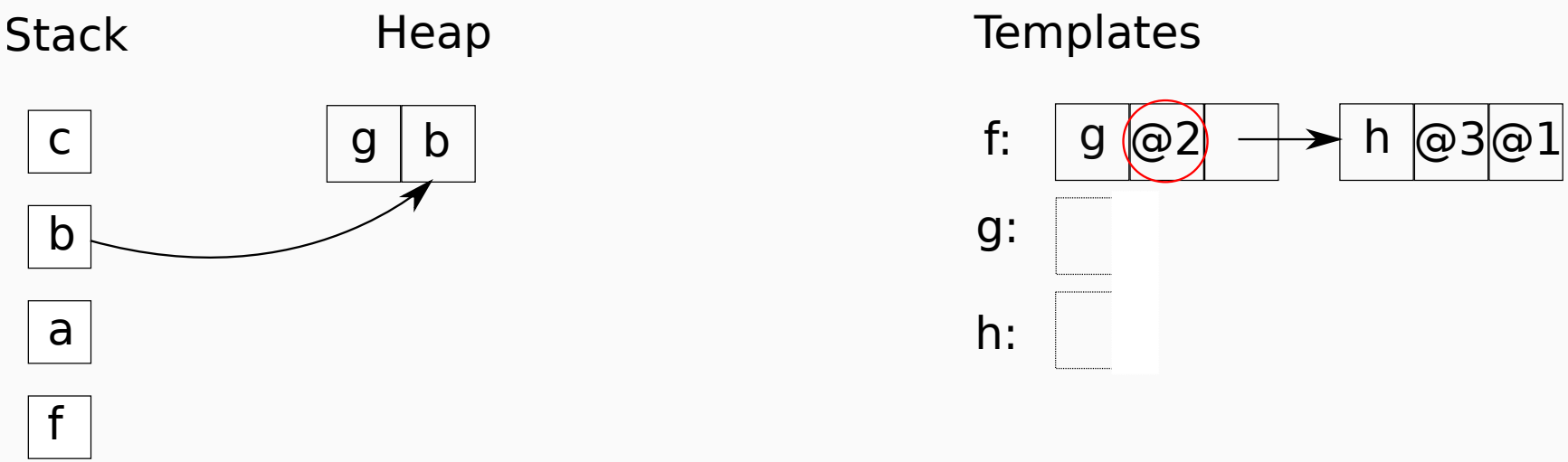
```
f ys x xs = g x (h xs ys)
```



From “Dynamic analysis in the Reduceron”, Matthew Naylor and Colin Runciman, University of York, 2009.
https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf

Reduceron: reduction with templates

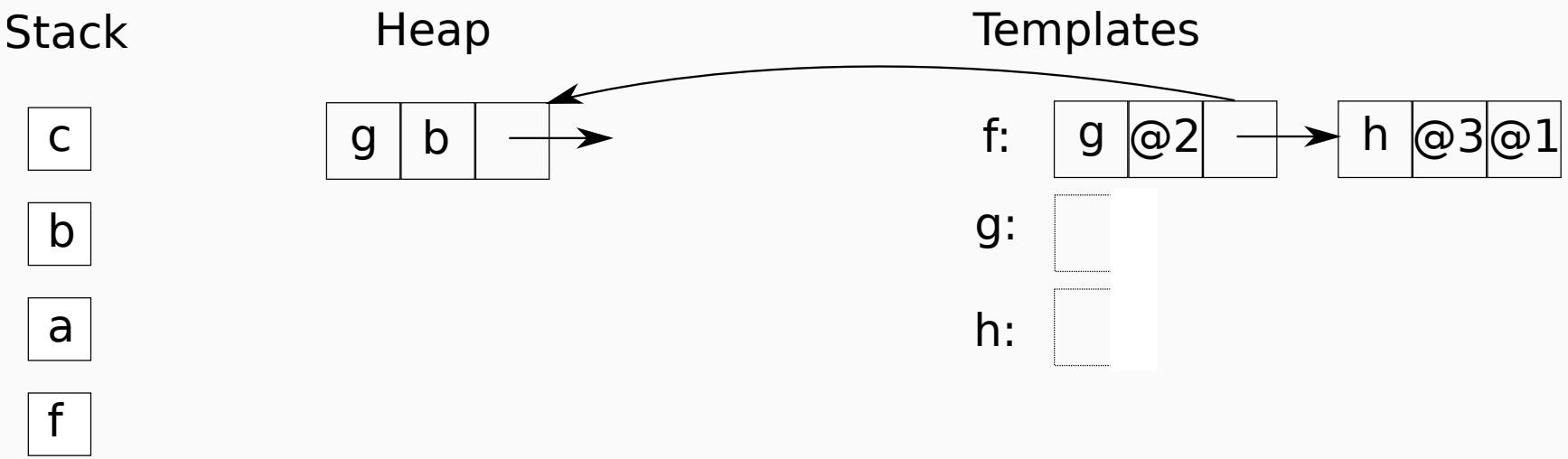
```
f ys x xs = g x (h xs ys)
```



From “Dynamic analysis in the Reduceron”, Matthew Naylor and Colin Runciman, University of York, 2009.
https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf

Reduceron: reduction with templates

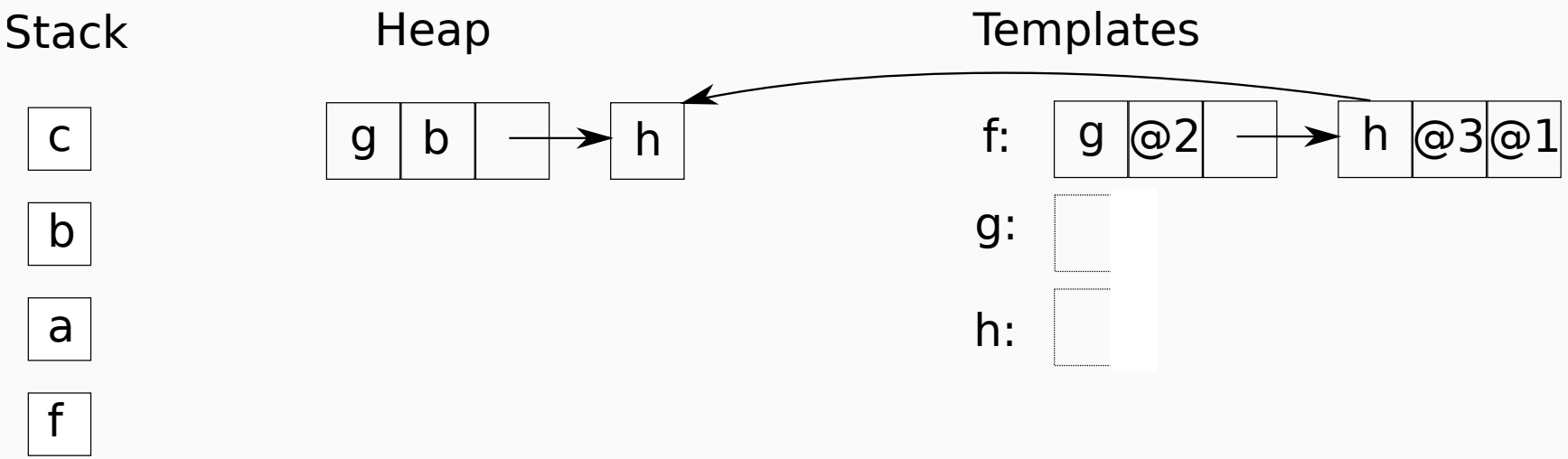
```
f ys x xs = g x (h xs ys)
```



From “Dynamic analysis in the Reduceron”, Matthew Naylor and Colin Runciman, University of York, 2009.
https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf

Reduceron: reduction with templates

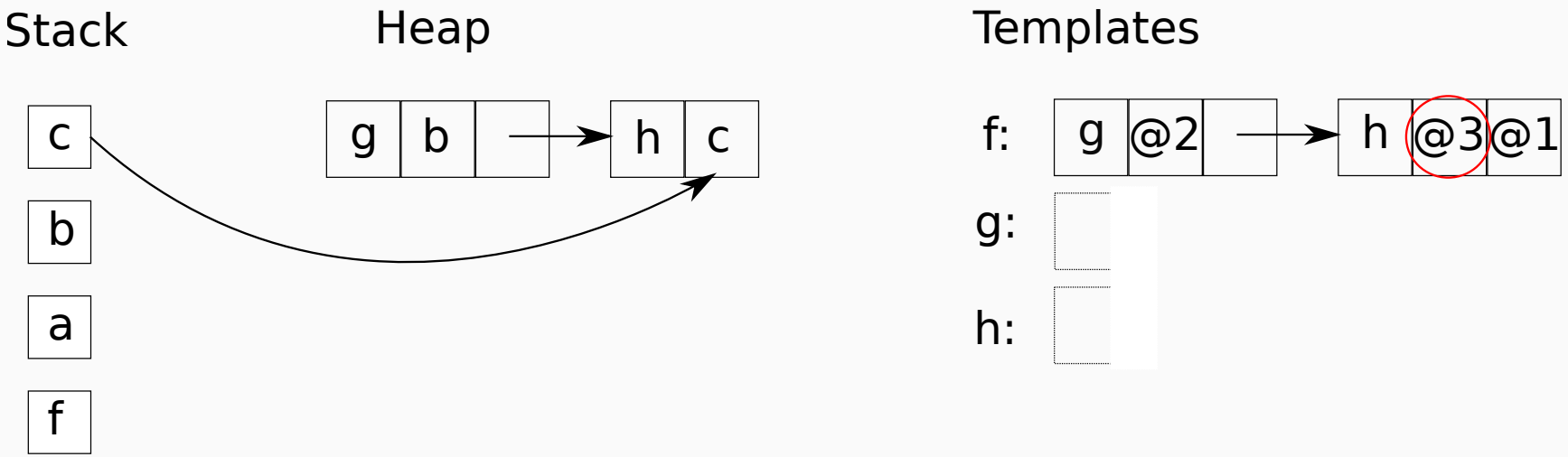
```
f ys x xs = g x (h xs ys)
```



From “Dynamic analysis in the Reduceron”, Matthew Naylor and Colin Runciman, University of York, 2009.
https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf

Reduceron: reduction with templates

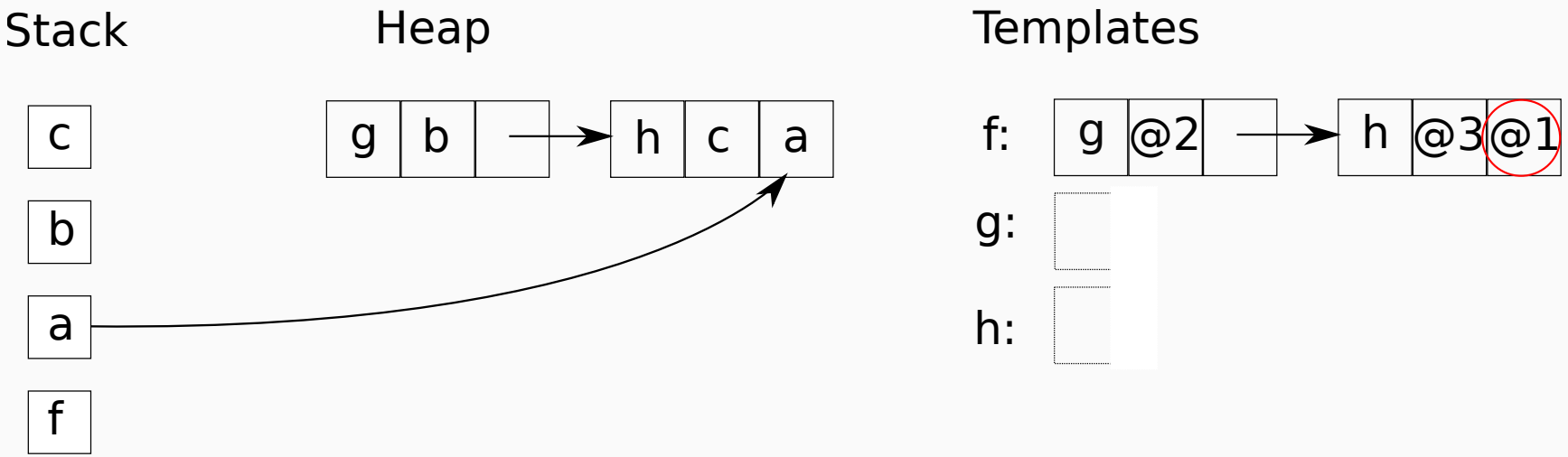
```
f ys x xs = g x (h xs ys)
```



From “Dynamic analysis in the Reduceron”, Matthew Naylor and Colin Runciman, University of York, 2009.
https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf

Reduceron: reduction with templates

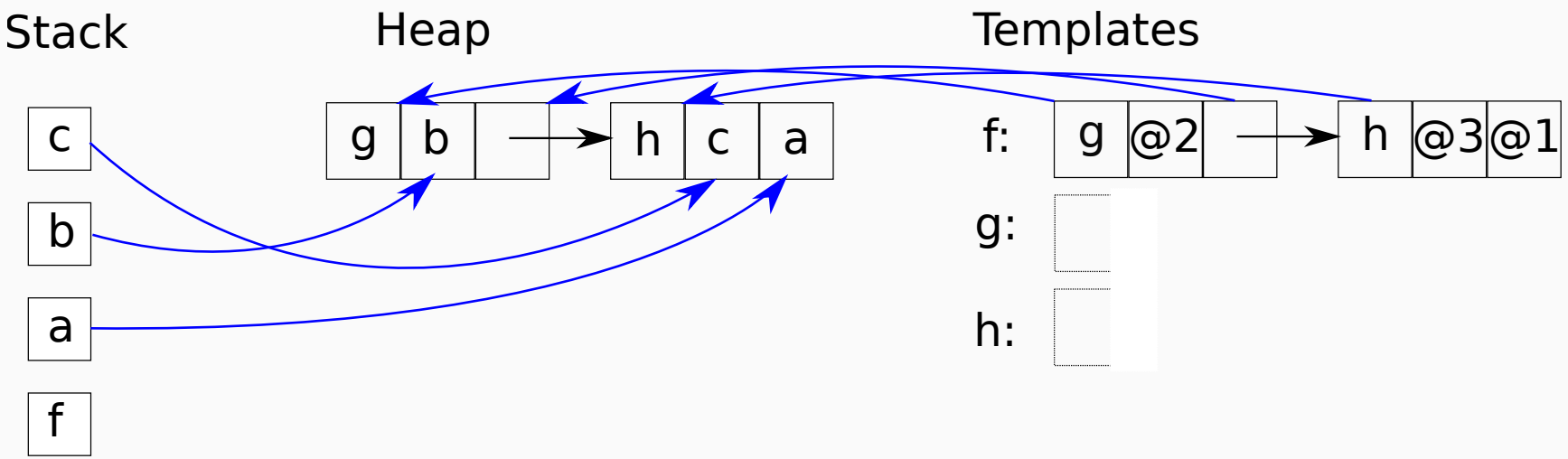
```
f ys x xs = g x (h xs ys)
```



From “Dynamic analysis in the Reduceron”, Matthew Naylor and Colin Runciman, University of York, 2009.
https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf

Reduceron: reduction with templates

```
f ys x xs = g x (h xs ys)
```



- Reduceron access to *parallel* memories:
  1. stack
  2. heap
  3. templates
- FPGAs: function application in a **single clock cycle**

From "Dynamic analysis in the Reduceron", Matthew Naylor and Colin Runciman, University of York, 2009.
`https://www.cs.york.ac.uk/fp/reduceron/memos/Memo41.pdf`

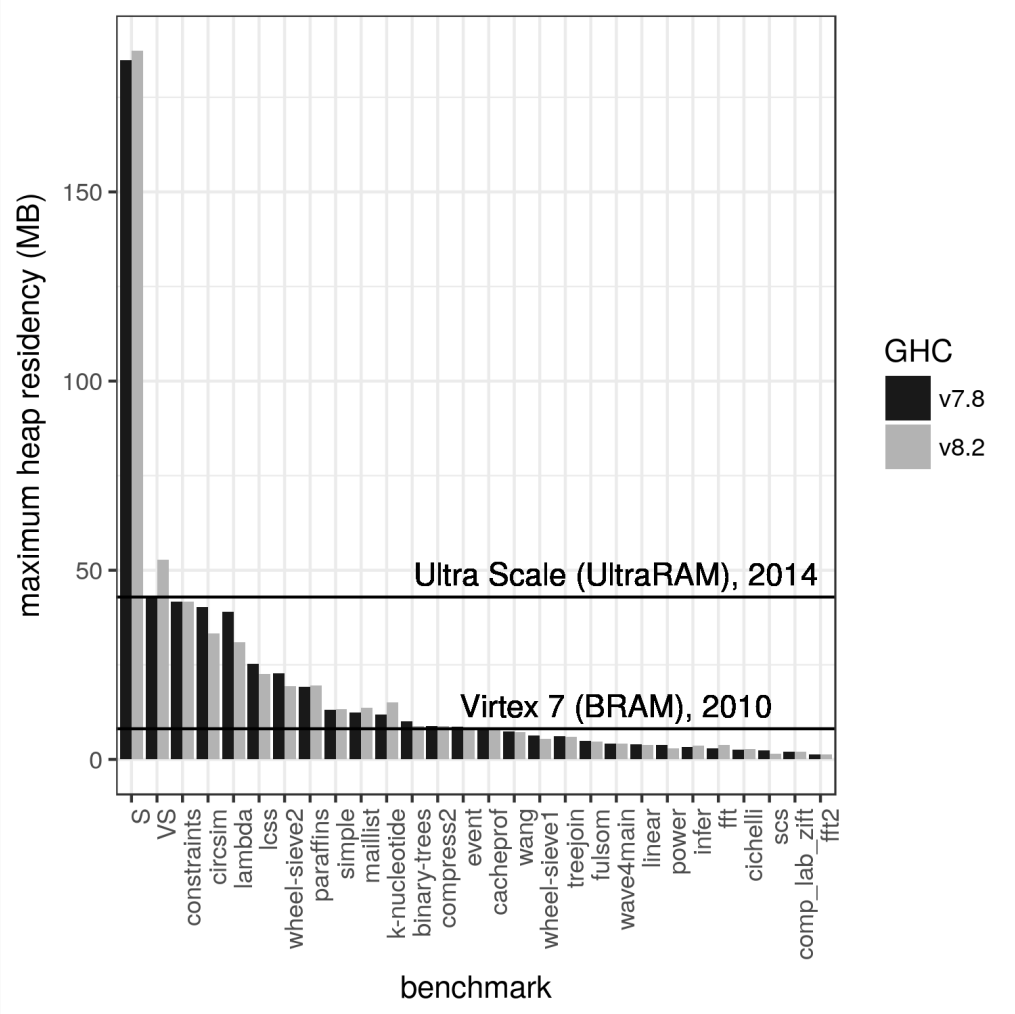
Extending Reduceron for Modern FPGAs

1. **Parallel graph reduction**
    - Fit multiple reduction cores onto FPGA fabric
2. **Off-chip heap**
    - Real world programs use need 100s MBs of heap space
3. **Caching**
    - Low latency on-chip successive reductions of an expression
4. **Compiler optimisations**
    - Profit from *space* and *time* saving *compiler optimisations*

Modern FPGAs for Graph Reduction

On-Chip Memory

Would a heap fit entirely on chip (with a garbage collector)?



On-Chip Space

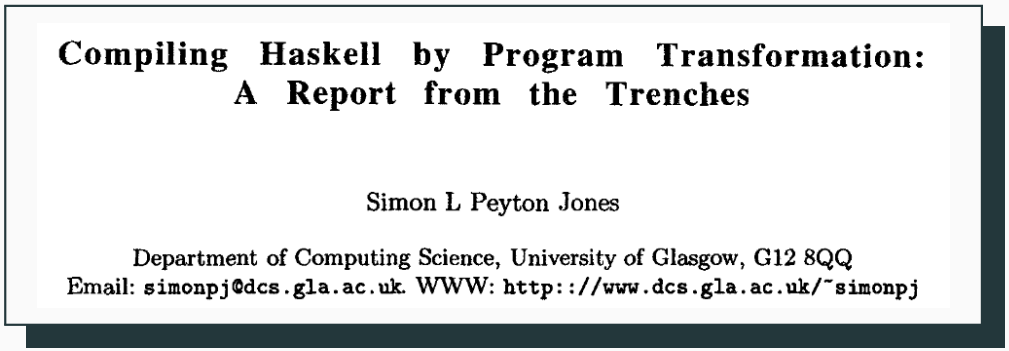
> *The functional language is a ballerina at an imperative square dance. A multiprocessor of appropriate design could better serve the functional language's requirements.*
>
> *William Partain, 1989*

FPGA	Slice LUTs	BRAMs	**Reduceron Cores**
Kintex 7 kc705	10%	30%	**3**
Zynq 7 zc706	9%	24%	**4**
Virtex 7 vc709	5%	9%	**10**
Virtex UltraScale vcu100	2%	3%	**28**

Multiple reducer cores, potential for **parallel** graph reduction.

Allocations off-chip

Can GHC optimisations reduce need for FPGA off chip allocation?



Compiling Haskell by Program Transformation: A Report from the Trenches

Simon L Peyton Jones

Department of Computing Science, University of Glasgow, G12 8QQ
Email: simonpj@dcs.gla.ac.uk. WWW: http://www.dcs.gla.ac.uk/~simonpj

European Symposium on Programming, 1996.

- inlining,
- strictness analysis,
- let floating,
- Eta expansion, ...

Allocations off-chip

Can GHC optimisations reduce need for FPGA off chip allocation?

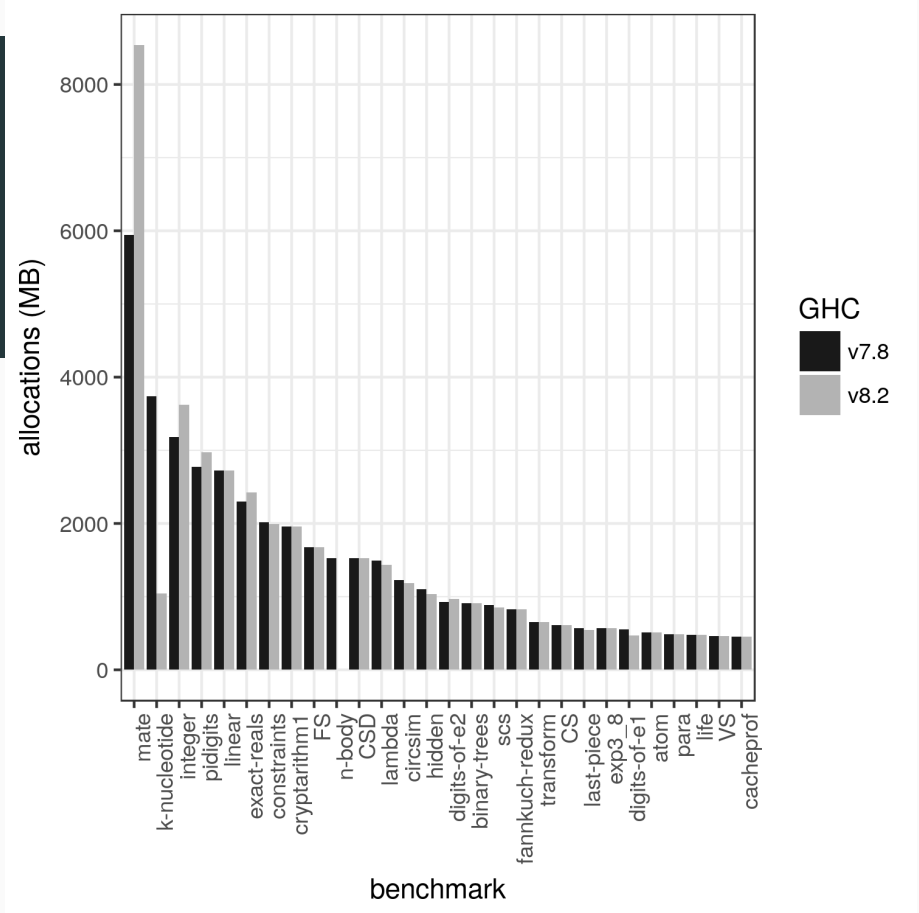
Compiling Haskell by Program Transformation: A Report from the Trenches

Simon L Peyton Jones

Department of Computing Science, University of Glasgow, G12 8QQ
Email: simonpj@dcs.gla.ac.uk. WWW: http://www.dcs.gla.ac.uk/~simonpj

European Symposium on Programming, 1996.

- inlining,
- strictness analysis,
- let floating,
- Eta expansion, ...

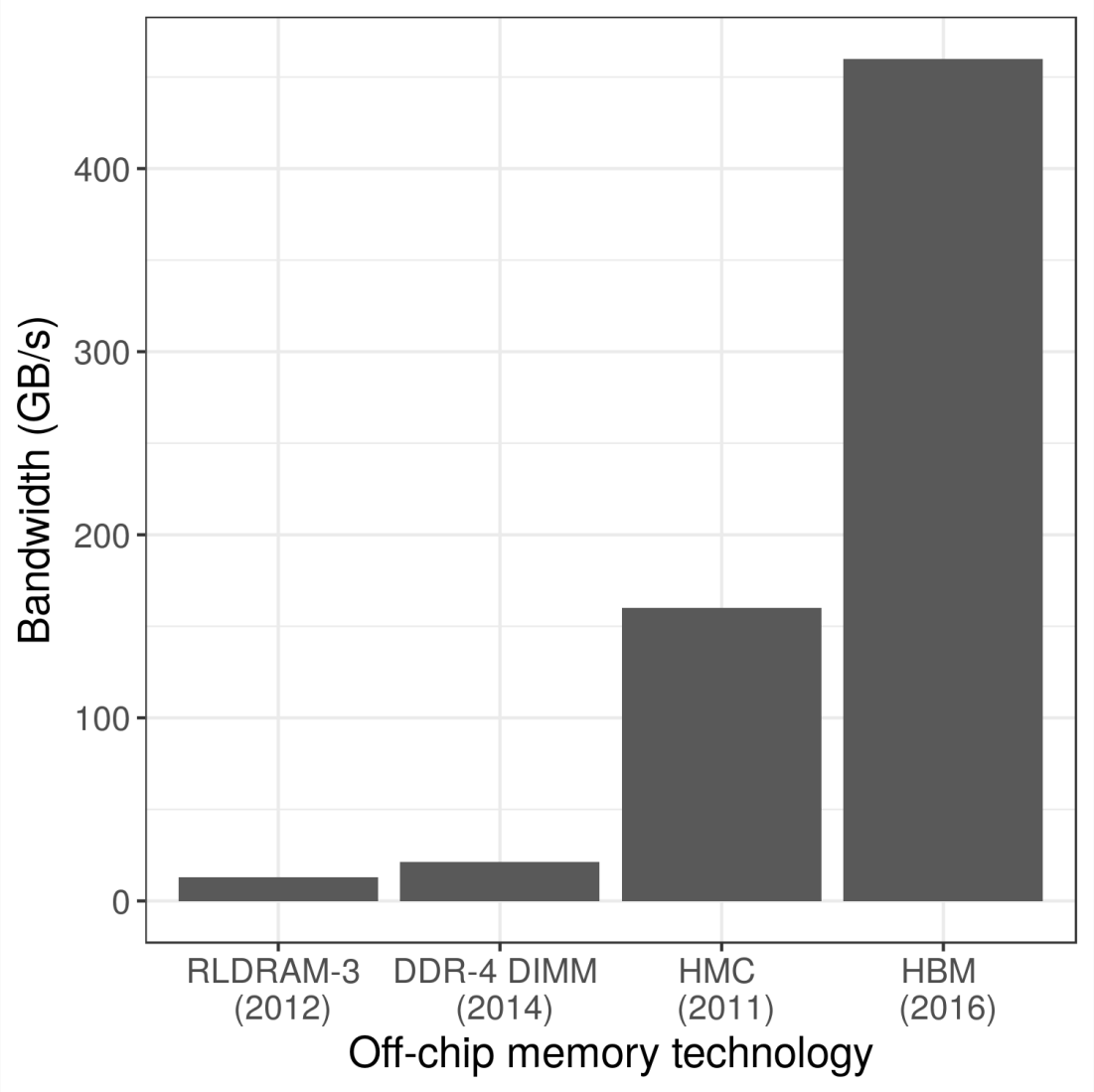


GHC 7.8 $\rightarrow$ 8.2:

72% reduced allocation for k-nucleotide, almost 100% for n-body.

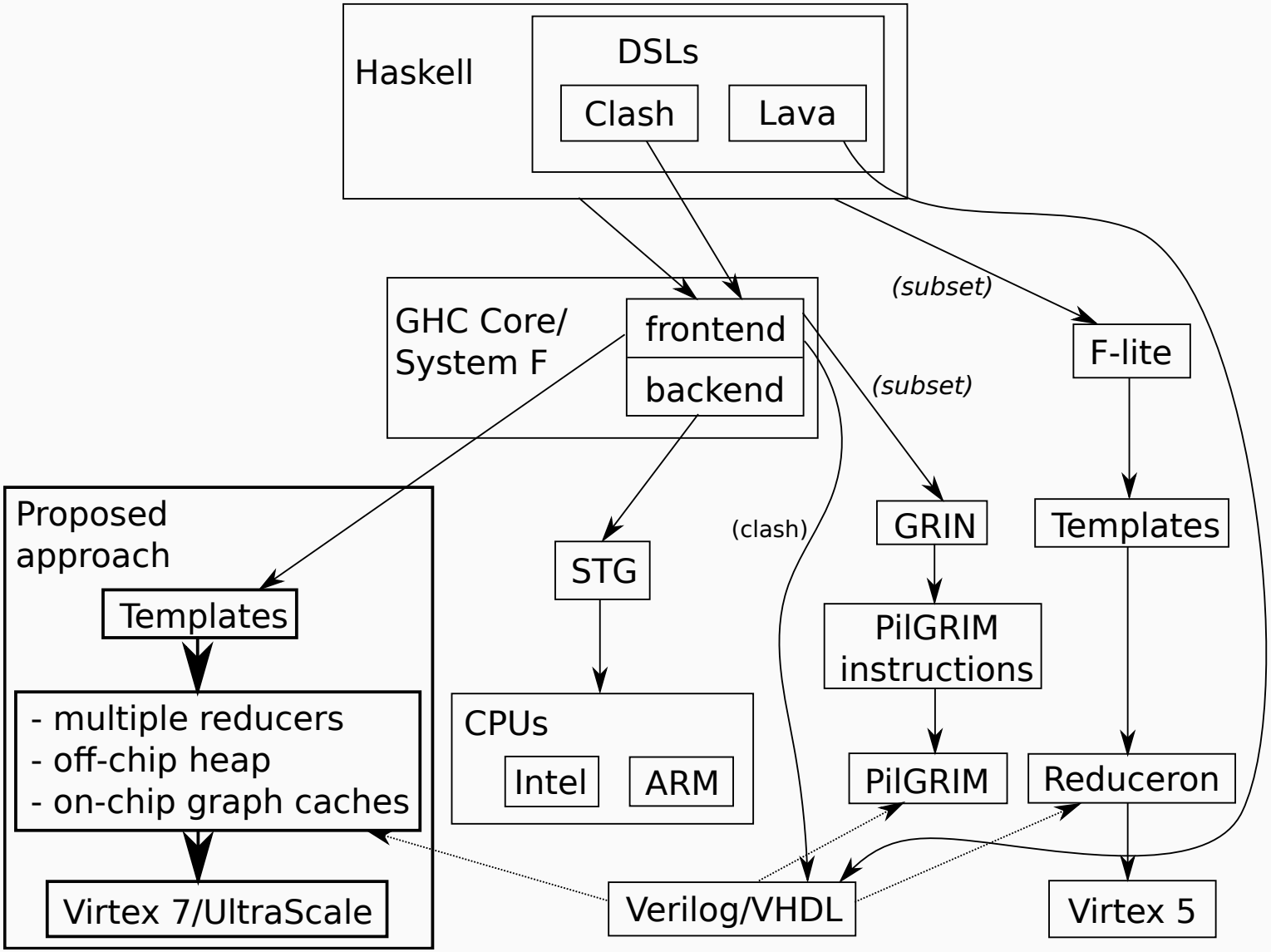
FPGA ↔ Memory Bandwidth

Potential for throughput of off-chip heap reads/writes?



Summary

- **Simple parallelism?**
  - evaluate strict (always needed) function arguments in parallel
- **Dynamic parallelism?** borrow GHC RTS ideas:
  - *par/pseq* for programmer controlled parallel task sizes
  - black holes to avoid duplicating work
  - load balancing between cores
- **Proposed hardware**
  - HDL $\rightarrow$ synthesis $\rightarrow$ *graph reduction FPGA machine*
  - Dedicate cache manager
  - Off-chip memories for heap
  - Parallel reduction with multiple reduction cores
- **Compiling Haskell to it**
  - Haskell $\rightarrow$ GHC Core $\rightarrow$ templates
  - profit from GHC optimisations
  - ... but GHC Core is bigger language than F-lite (Reduceron)
  - ... therefore more challenging to support


Haskell
DSLs
Clash
Lava
GHC Core/
System F
frontend
backend
(subset)
F-lite
(subset)
Proposed
approach
Templates
- multiple reducers
- off-chip heap
- on-chip graph caches
Virtex 7/UltraScale
STG
(clash)
GRIN
Templates
PilGRIM
instructions
CPUs
Intel
ARM
PilGRIM
Reduceron
Verilog/VHDL
Virtex 5