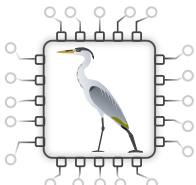
The Heron Core

A graph reduction processor with concurrent garbage collection

Craig Ramsay & Rob Stewart

March 2024

Heriot-Watt University

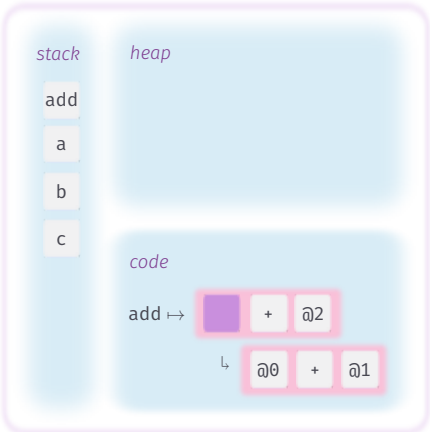


> *I wonder how popular Haskell needs to become for Intel to optimize their processors for my runtime, rather than the other way around.*
>
> — Simon Marlow, 2009

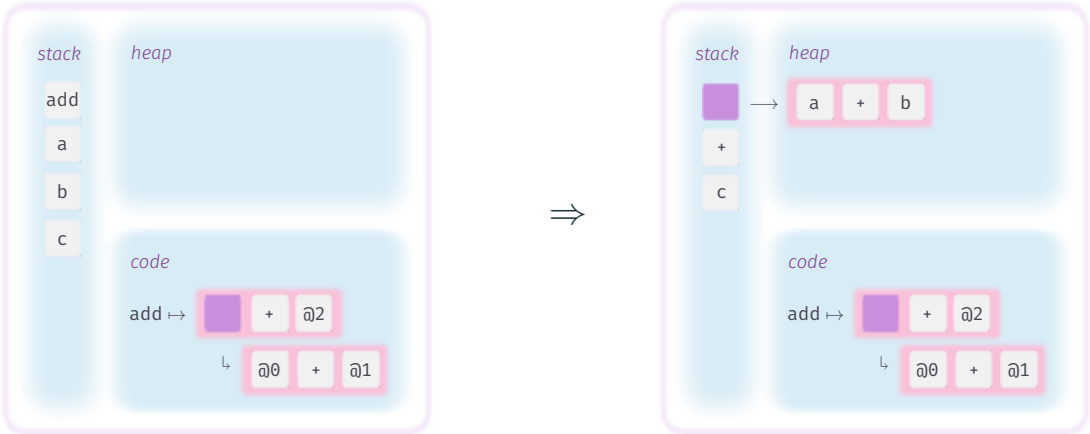
Graph Reduction

Function application of `add x y z = (x+y)+z`

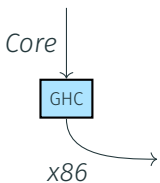
Function application of `add x y z = (x+y)+z`



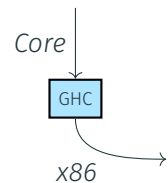
Function application of `add x y z = (x+y)+z`



```
add x y z =
  x + y + z
```



```
add x y z =
  x + y + z
```




```
s1ba_info: ; Code for helper (\a b -> a+b)
.Lc1bo: ; Check for stack space
   leaq -40(%rbp),%rax
   cmpq %r15,%rax
   jb .Lc1bp ; Jump if stack full
.Lc1bq: ; Reduce helper
   movq $stg_upd_frame_info,-16(%rbp)
   movq %rbx,-8(%rbp)
   movq 16(%rbx),%rax ; Load a & b from heap
   movq 24(%rbx),%rbx
   movl $base_GHCziNum_zdfNumInt_closure,%r14d
   ;; Push `a+b` onto stack
   movq $stg_ap_pp_info,-40(%rbp)
   movq %rax,-32(%rbp)
   movq %rbx,-24(%rbp)
   addq $-40,%rbp
   jmp base_GHCziNum_zp_info ; Enter
.Lc1bp: ; Ask RTS for stack space
   jmp *-16(%r13)
```

```
Add_add_info: ; Code for `add`
.Lc1br: ; Check for stack space
   leaq -24(%rbp),%rax
   cmpq %r15,%rax
   jb .Lc1bs ; Jump if stack full
.Lc1bt: ; Check for heap space
   addq $32,%r12
   cmpq 856(%r13),%r12
   ja .Lc1bv ; Jump if heap full
.Lc1bu: ; Reduce `add`
   ;; Build `x+y` thunk on heap
   movq $s1ba_info,-24(%r12)
   movq %r14,-8(%r12)
   movq %rsi,(%r12)
   leaq -24(%r12),%rax
   movl $base_GHCziNum_zdfNumInt_closure,%r14d
   ;; Push `thunk+z` to stack
   movq $stg_ap_pp_info,-24(%rbp)
   movq %rax,-16(%rbp)
   movq %rdi,-8(%rbp)
   addq $-24,%rbp
   jmp base_GHCziNum_zp_info ; Enter
.Lc1bv: ; Ask RTS for heap space
   movq $32,904(%r13)
.Lc1bs: ; Ask RTS for stack space
   movl $Add_add_closure,%ebx
   jmp *-8(%r13)
```

Opportunities for Custom Architectures

Intra-function parallelism

Usually a victim of the von Neumann bottleneck. Worse for lazy, pure languages.

Hardened, concurrent run-time system

Tasks like garbage collection usually halt reductions in software implementations.

Inter-function parallelism

Exploiting the purity of functional languages.

Opportunities for Custom Architectures

Intra-function parallelism

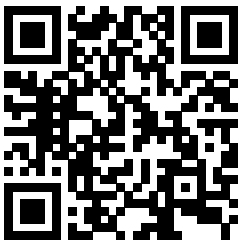
Usually a victim of the von Neumann bottleneck. Worse for lazy, pure languages.

Hardened, concurrent run-time system

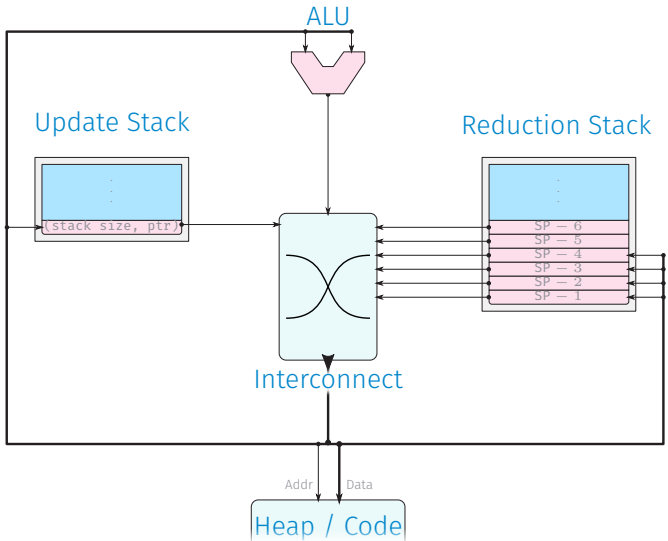
Tasks like garbage collection usually halt reductions in software implementations.

Inter-function parallelism

Exploiting the purity of functional languages.



A Template Instantiation Machine

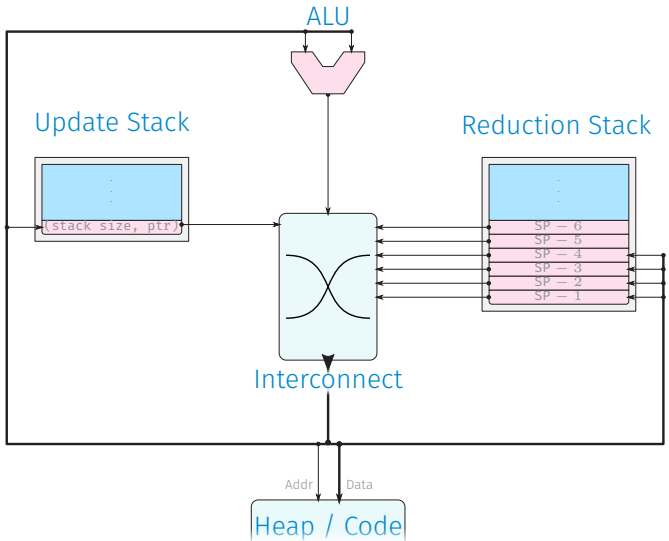


≈ Augustsson's
Big Word Machine[1]

One instruction to build wide node on heap, and one to permute the stack

λ-calculus + integers

[1]Augustsson, "BWM: A Concrete Machine for Graph Reduction".



Update avoidance with dynamic analysis.

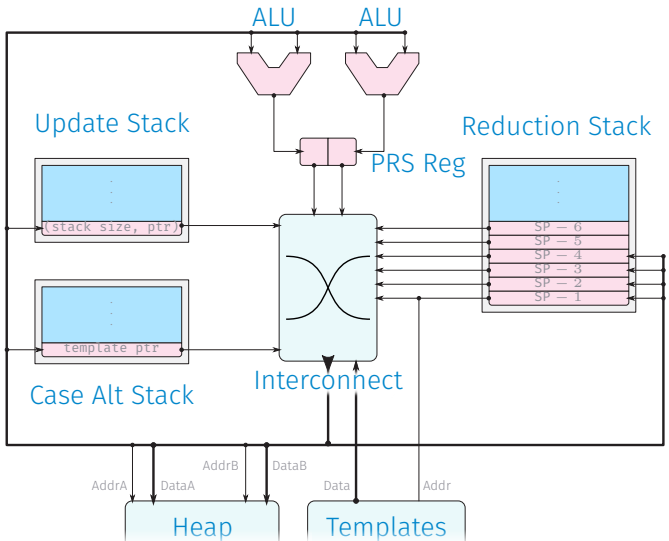
One-bit reference counting from Stoye[2]

[2]Stoye, *The implementation of functional languages using custom hardware.*

avoidance with
mic analysis.

bit reference
ng from Stoye[2]

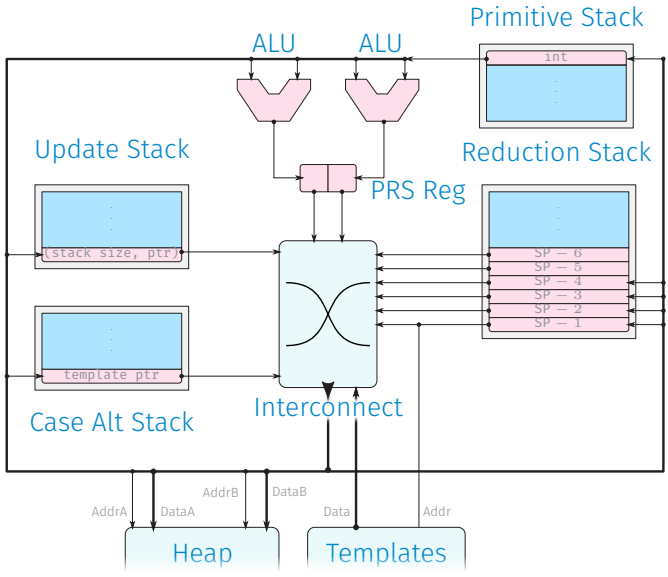
[2]Stoye, *The implementation of functional languages using custom hardware.*



Towards Naylor's Reduceron[3]

Goal: single-cycle β-reduction on real hardware

[3]Naylor and Runciman, "The Reduceron reconfigured and re-evaluated".



Our Heron core[4]:

+ Postfix primitive operations

`(f x) + (g y)` ⇒ `f x g y +`

+ Zero-constraint templates

+ Inline case alternatives

[4] Ramsay and Stewart, *Dataset for the Heron Core in "Heron: Modern Hardware Graph Reduction"*.

Circuit Results

For our “best” Heron configuration:

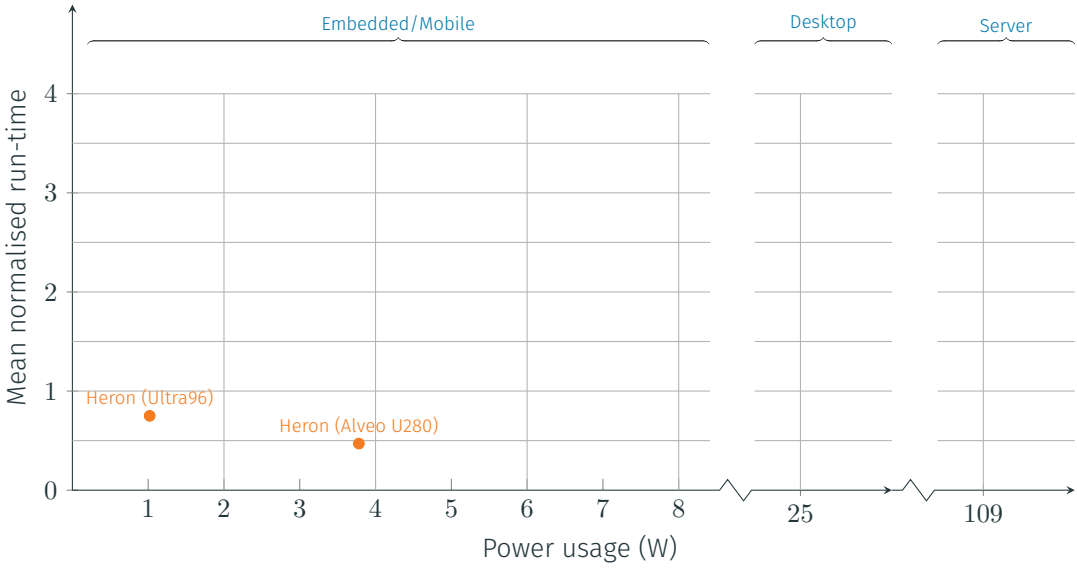
Heap applications are 6 atoms wide, and top 8 stack atoms are accessible

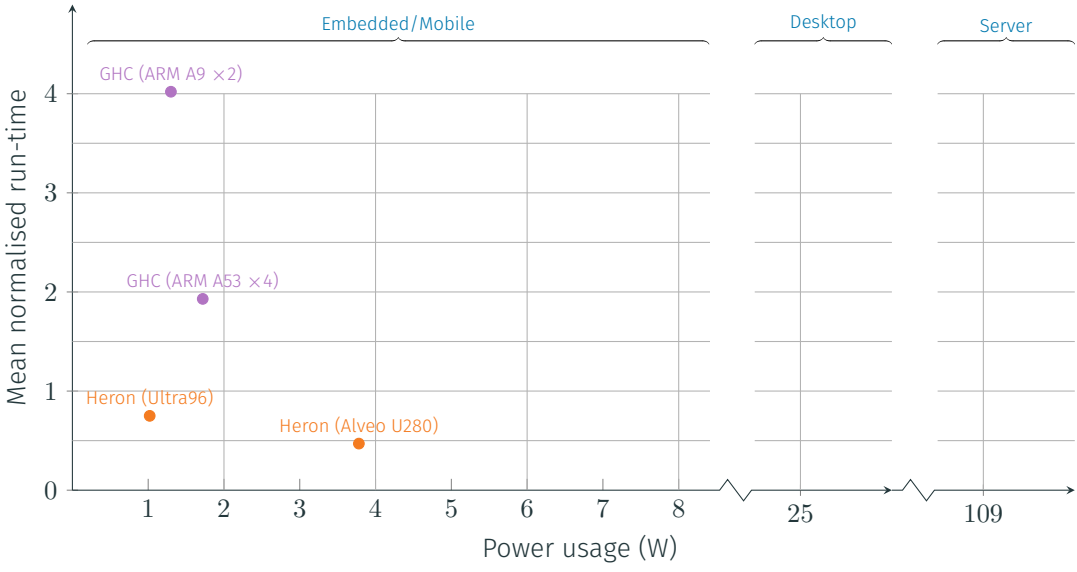
193 MHz, max usage $< 2\%$ on Alveo U280

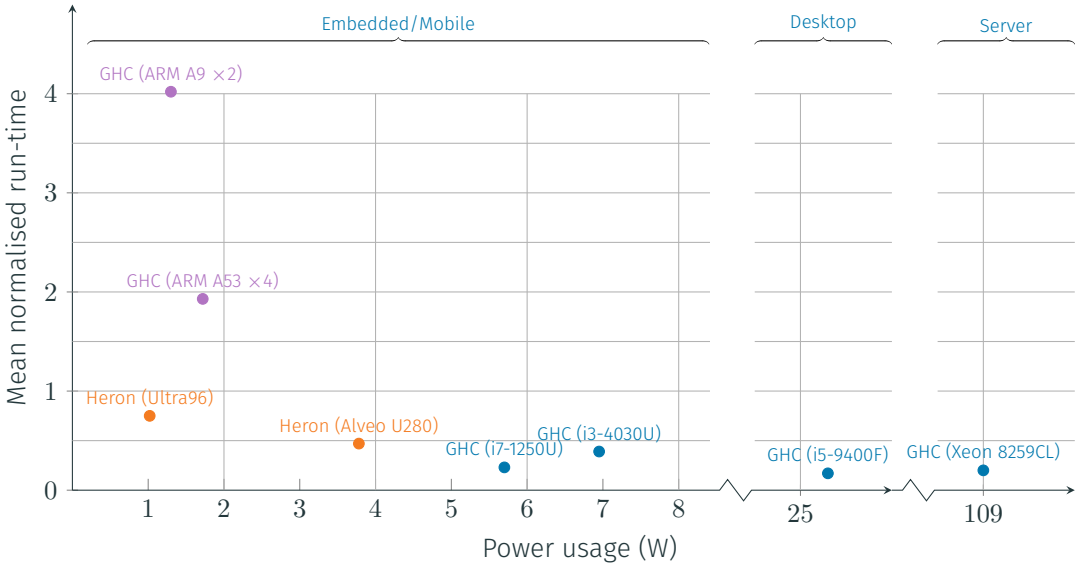
Mean 6% reduction in cycles (max 17%)

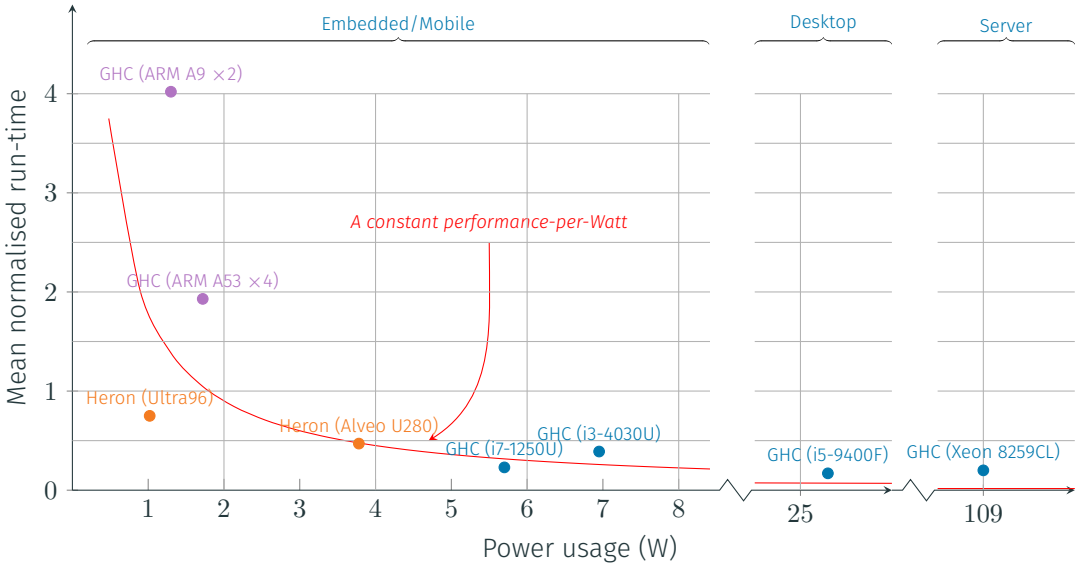
Mean 22% reduction in code size (max 34%)

Mean 12% reduction in heap allocs (max 100%)


Embedded/Mobile
Desktop
Server
Mean normalised run-time
0
1
2
3
4
Heron (Ultra96)
Heron (Alveo U280)
1
2
3
4
5
6
7
8
25
109
Power usage (W)


Embedded/Mobile
Desktop
Server
GHC (ARM A9 ×2)
GHC (ARM A53 ×4)
Heron (Ultra96)
Heron (Alveo U280)
Mean normalised run-time
0
1
2
3
4
1
2
3
4
5
6
7
8
25
109
Power usage (W)

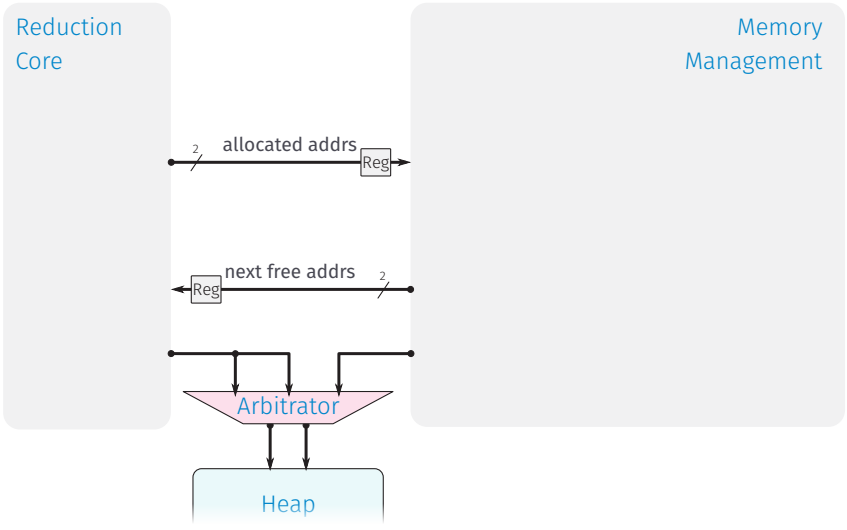

Embedded/Mobile
Desktop
Server
GHC (ARM A9 ×2)
GHC (ARM A53 ×4)
Heron (Ultra96)
Heron (Alveo U280)
GHC (i7-1250U)
GHC (i3-4030U)
GHC (i5-9400F)
GHC (Xeon 8259CL)
Mean normalised run-time
0
1
2
3
4
1
2
3
4
5
6
7
8
25
109
Power usage (W)

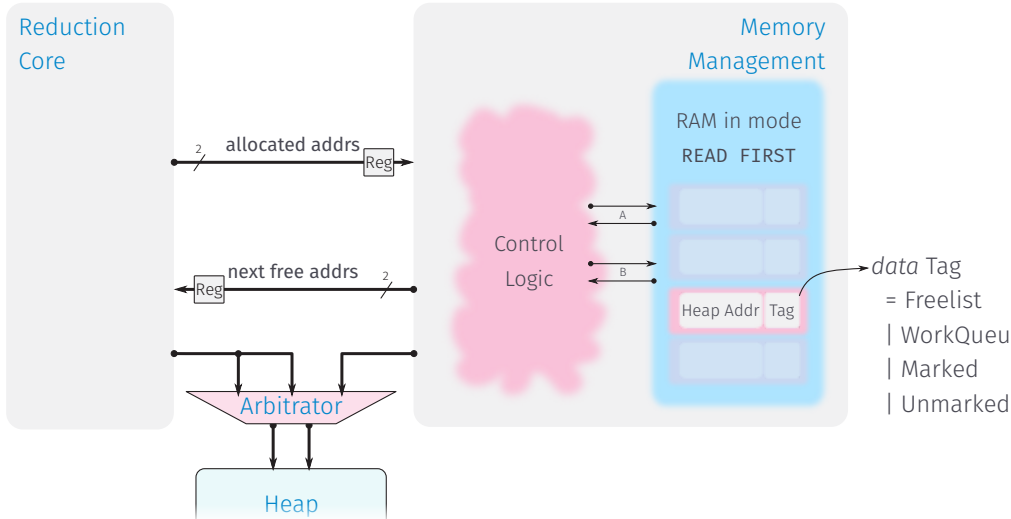

Embedded/Mobile
Desktop
Server
GHC (ARM A9 ×2)
GHC (ARM A53 ×4)
A constant performance-per-Watt
Heron (Ultra96)
Heron (Alveo U280)
GHC (i7-1250U)
GHC (i3-4030U)
GHC (i5-9400F)
GHC (Xeon 8259CL)
Mean normalised run-time
0
1
2
3
4
1
2
3
4
5
6
7
8
25
109
Power usage (W)

> *The absence of a garbage collector is a serious shortcoming and making this work efficiently may be hard with the memory optimizations that are implemented*

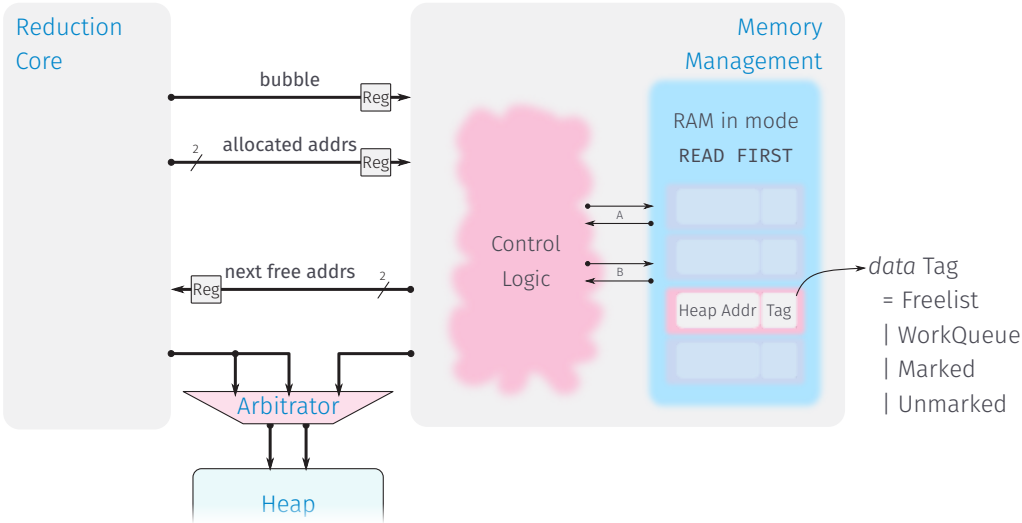
— Reviewer

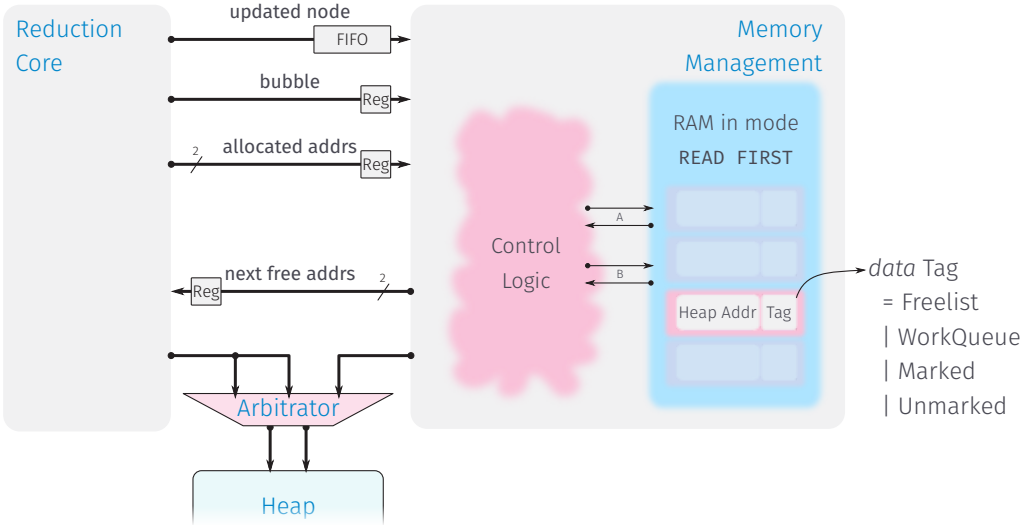
A concurrent garbage collector

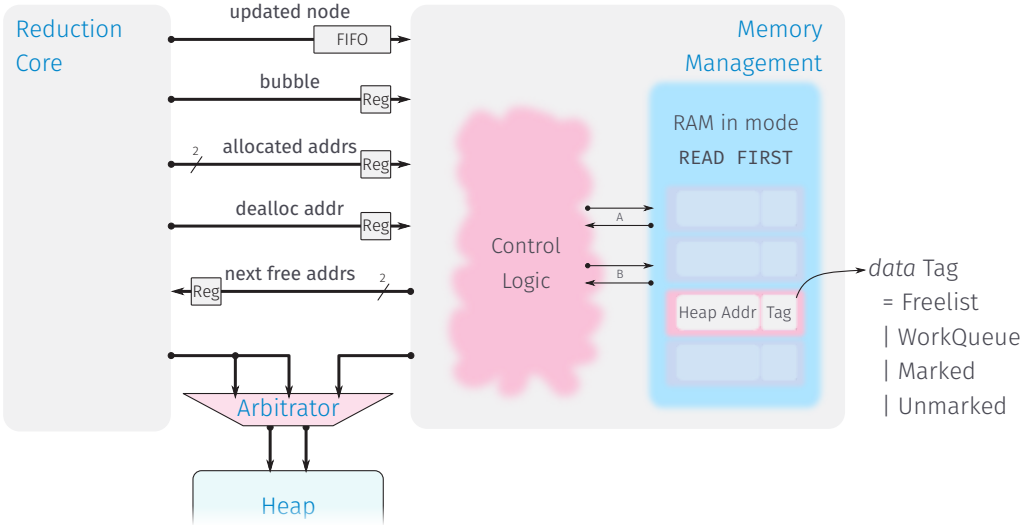

Reduction
Core
Memory
Management
2
allocated addrs
Reg
next free addrs
Reg
2
Arbitrator
Heap



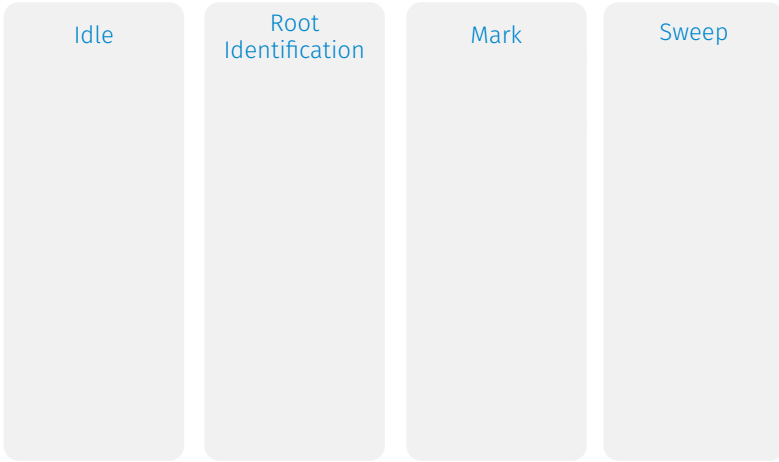
[4]Barker, Edwards, and Kim, “Synthesized In-BRAM Garbage Collection for Accelerators with Immutable Memory”.


Reduction Core
bubble
Reg
2
allocated addrs
Reg
next free addrs
2
Reg
Arbitrator
Heap
Memory Management
RAM in mode
READ FIRST
Control Logic
A
B
Heap Addr
Tag
data Tag
= Freelist
| WorkQueue
| Marked
| Unmarked


Reduction Core
updated node
FIFO
bubble
Reg
2
allocated addrs
Reg
next free addrs
2
Reg
Arbitrator
Heap
Memory Management
Control Logic
A
B
RAM in mode
READ FIRST
Heap Addr
Tag
data Tag
= Freelist
| WorkQueue
| Marked
| Unmarked



[4]Stoye, *The implementation of functional languages using custom hardware.*


Idle
Root Identification
Mark
Sweep

Idle

Redistribute Freelists

Root Identification

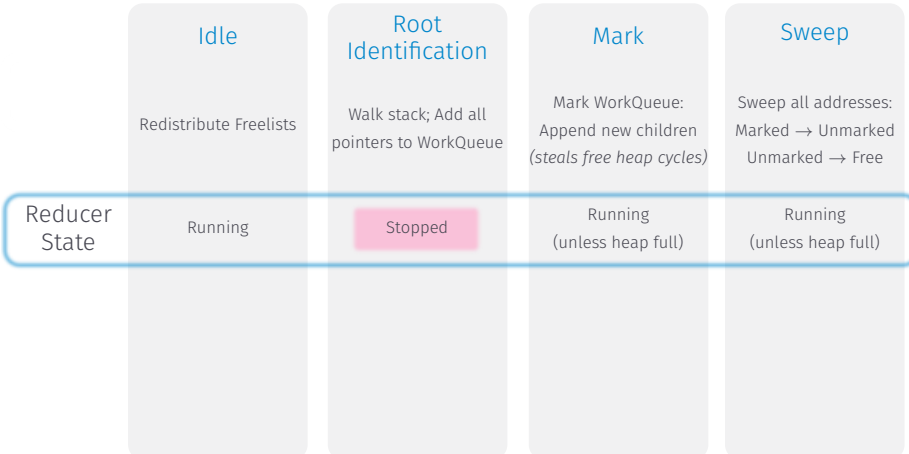
Walk stack; Add all pointers to WorkQueue

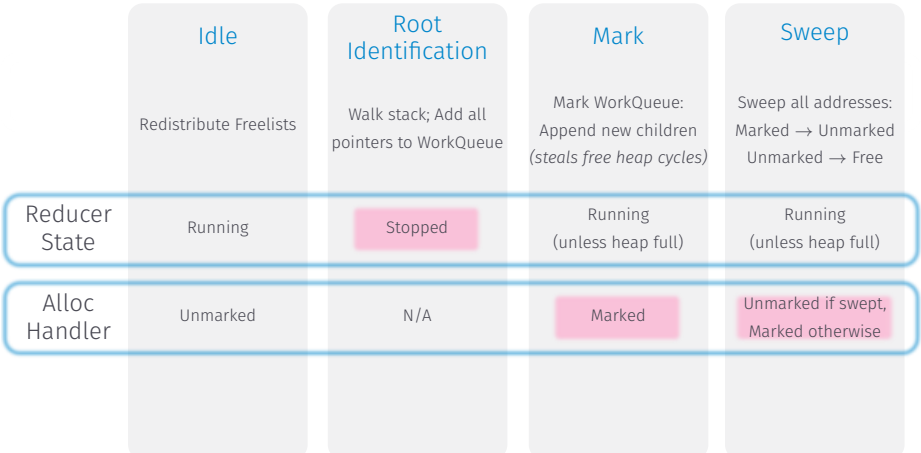
Mark

Mark WorkQueue:
Append new children
(steals free heap cycles)

Sweep

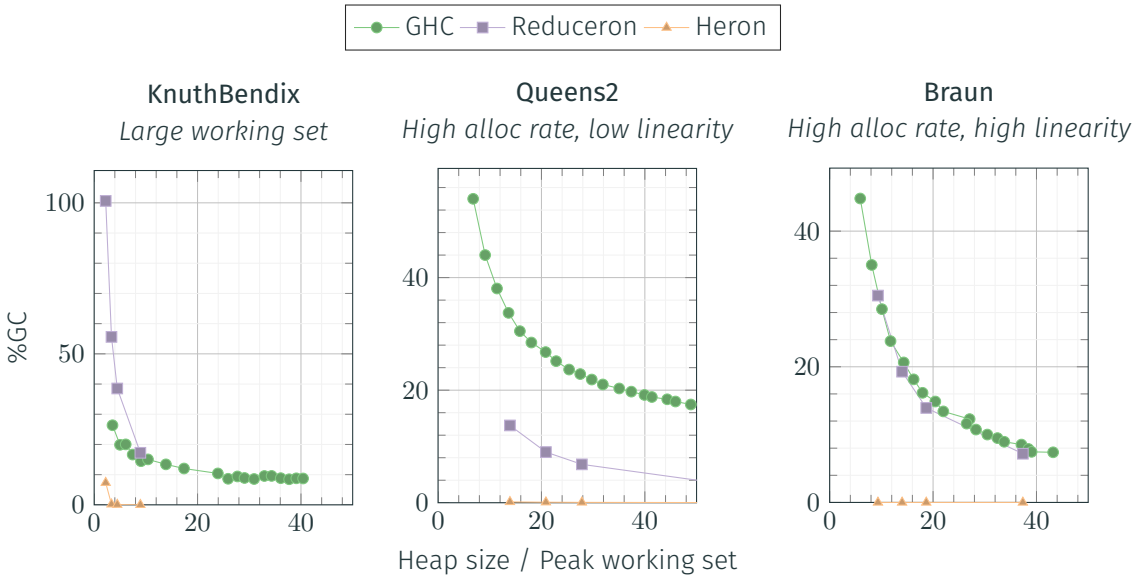
Sweep all addresses:
Marked → Unmarked
Unmarked → Free


Idle
Root Identification
Mark
Sweep
Redistribute Freelists
Walk stack; Add all pointers to WorkQueue
Mark WorkQueue: Append new children (steals free heap cycles)
Sweep all addresses: Marked → Unmarked Unmarked → Free
Reducer State
Running
Stopped
Running (unless heap full)
Running (unless heap full)


Idle
Root Identification
Mark
Sweep
Redistribute Freelists
Walk stack; Add all pointers to WorkQueue
Mark WorkQueue: Append new children (steals free heap cycles)
Sweep all addresses: Marked → Unmarked Unmarked → Free
Reducer State
Running
Stopped
Running (unless heap full)
Running (unless heap full)
Alloc Handler
Unmarked
N/A
Marked
Unmarked if swept, Marked otherwise

	Idle	Root Identification	Mark	Sweep
	Redistribute Freelists	Walk stack; Add all pointers to WorkQueue	Mark WorkQueue: Append new children *(steals free heap cycles)*	Sweep all addresses: Marked → Unmarked Unmarked → Free
Reducer State	Running	Stopped	Running (unless heap full)	Running (unless heap full)
Alloc Handler	Unmarked	N/A	Marked	Unmarked if swept, Marked otherwise
Dealloc Handler	OK	N/A	Ignore	Ignore if swept, OK otherwise

Benchmark examples



What’s next?

Intra-function parallelism

Usually a victim of the von Neumann bottleneck. Worse for lazy, pure languages.

Hardened, concurrent run-time system

Tasks like garbage collection usually halt reductions in software implementations.

Inter-function parallelism

Exploiting the purity of functional languages.

Summary

Single-core processor to evaluate functional programs directly.

It's small — $< 2\%$ of an Alveo U280.

Not *that* slow — often faster than 4th Gen i3.

Better performance-per-Watt than a 12th Gen mobile i7.

Some techniques that are prohibitively expensive in software become simple.

Low-hanging fruit for multi-core architecture.

References

- Augustsson, Lennart. **“BWM: A Concrete Machine for Graph Reduction”.** In: *Functional Programming, Glasgow 1991*. Ed. by Rogardt Heldal, Carsten Kehler Holst, and Philip Wadler. London: Springer London, 1992, pp. 36–50. ISBN: 978-1-4471-3196-0.
- Barker, Martha, Stephen A. Edwards, and Martha A. Kim. **“Synthesized In-BRAM Garbage Collection for Accelerators with Immutable Memory”.** In: *32nd International Conference on Field-Programmable Logic and Applications, FPL 2022, Belfast, United Kingdom, August 29 - Sept. 2, 2022*. IEEE, 2022, pp. 47–53. DOI: 10.1109/FPL57034.2022.00019. URL: https://doi.org/10.1109/FPL57034.2022.00019.
- Naylor, Matthew and Colin Runciman. **“The Reduceron reconfigured and re-evaluated”.** In: *Journal of Functional Programming* 22.4-5 (2012), pp. 574–613. DOI: 10.1017/S0956796812000214.


Ramsay, Craig and Robert Stewart. ***Dataset for the Heron Core in "Heron: Modern Hardware Graph Reduction".*** 2024. DOI: 10.17861/f4fab5ef-ae98-4300-8328-ea59e47ff8c6. URL: https://doi.org/10.17861/f4fab5ef-ae98-4300-8328-ea59e47ff8c6 (visited on 02/21/2024).


Stoye, William Robert. ***The implementation of functional languages using custom hardware.*** Tech. rep. UCAM-CL-TR-81. University of Cambridge, Computer Laboratory, Dec. 1985. DOI: 10.48456/tr-81. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-81.pdf.