A Dynamic Memory Allocation Library for High-Level Synthesis

Nicholas V. Giamblanco and Jason H. Anderson
University of Toronto, Canada
Dept. of Electrical and Computer Engineering

FPL 2019

Dynamic Memory Allocation in HLS: Current Problems

No Obvious Way TO include it!

Where and How Big should the Arena(Heap) Be?

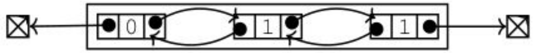
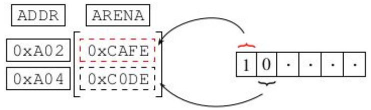
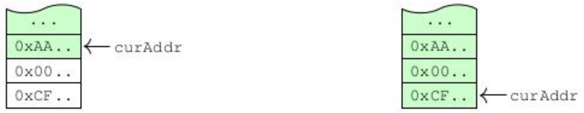
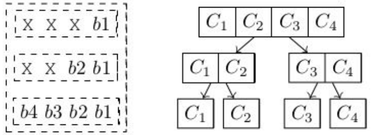
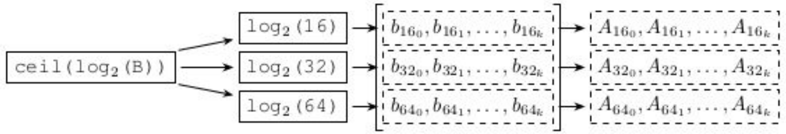
Which Allocator?

Performance & Area Problems

Dynamic Memory Allocation in HLS: Why Include it?

- No More Code-Refactoring!
- No More Memory Over-Provisioning
- Portability
- Marginal Performance and Area Impacts!!!

The Allocators

gnumem	Linked-List Allocator.	
bitmem	Bitmap Allocator.	 ADDR ARENA 0xA02 0xA04 0xCAFE 0xC0DE 1 0 · · · ·
linmem	Linear Allocator.	 ... 0xAA.. 0x00.. 0xCF.. curAddr ... 0xAA.. 0x00.. 0xCF.. curAddr
budmem	Buddy Allocator.	 X X X $b1$ X X $b2$ $b1$ $b4$ $b3$ $b2$ $b1$ C_1 C_2 C_3 C_4 C_1 C_2 C_3 C_4 C_1 C_2 C_3 C_4
lutmem	Look-Up Table Allocator.	 ceil($\log_2$(B)) $\log_2$(16) $\log_2$(32) $\log_2$(64) $b_{16_0}, b_{16_1}, \ldots, b_{16_k}$ $b_{32_0}, b_{32_1}, \ldots, b_{32_k}$ $b_{64_0}, b_{64_1}, \ldots, b_{64_k}$ $A_{16_0}, A_{16_1}, \ldots, A_{16_k}$ $A_{32_0}, A_{32_1}, \ldots, A_{32_k}$ $A_{64_0}, A_{64_1}, \ldots, A_{64_k}$

Our Approach

Implement Algorithms in HLS-friendly C Library

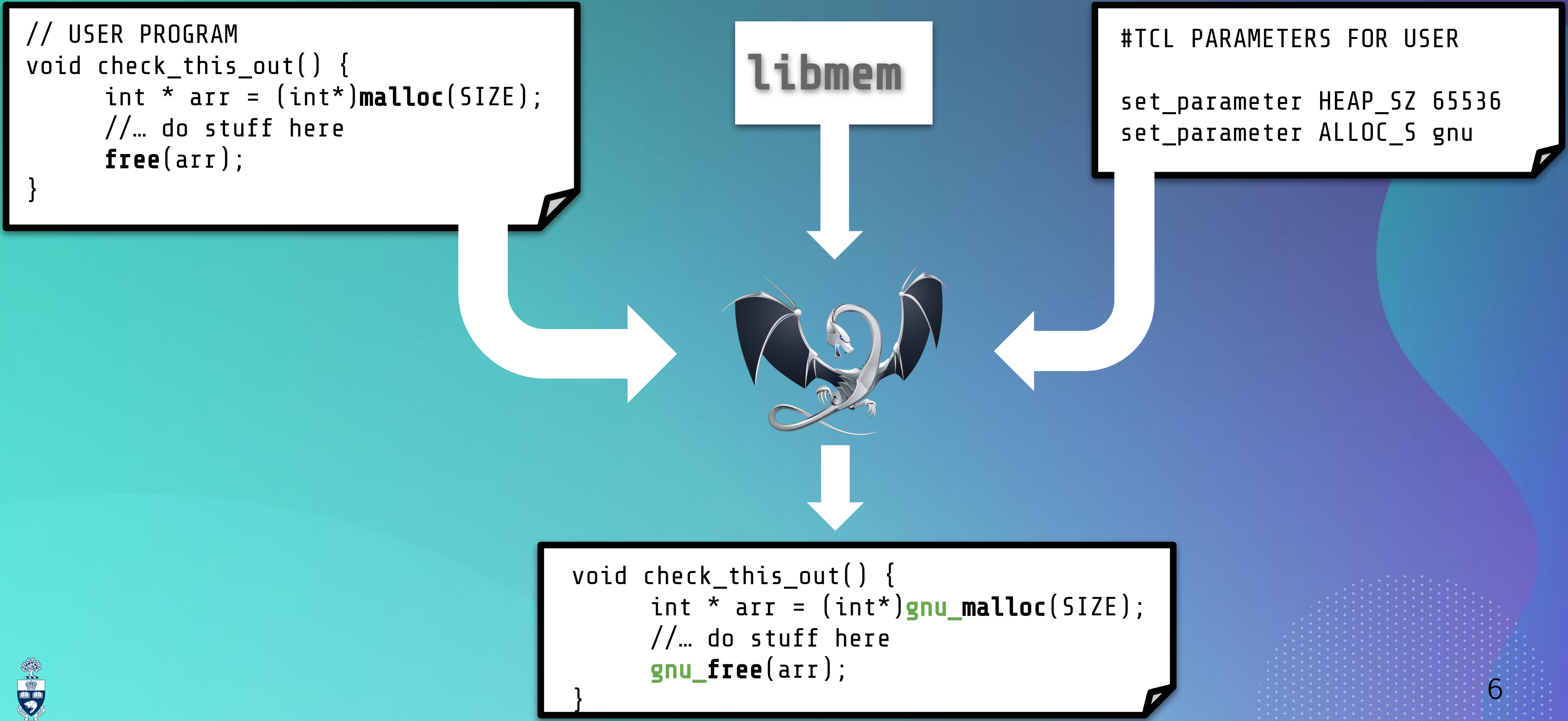
- Arena (heap) implemented as BRAM

Automate Transform with LLVM Pass

- User can select
  - Allocator Algorithm
  - Heap Size

Available on Github: https://github.com/ngiambla/libmem

Example:



```
// USER PROGRAM
void check_this_out() {
      int * arr = (int*)malloc(SIZE);
      //… do stuff here
      free(arr);
}
```

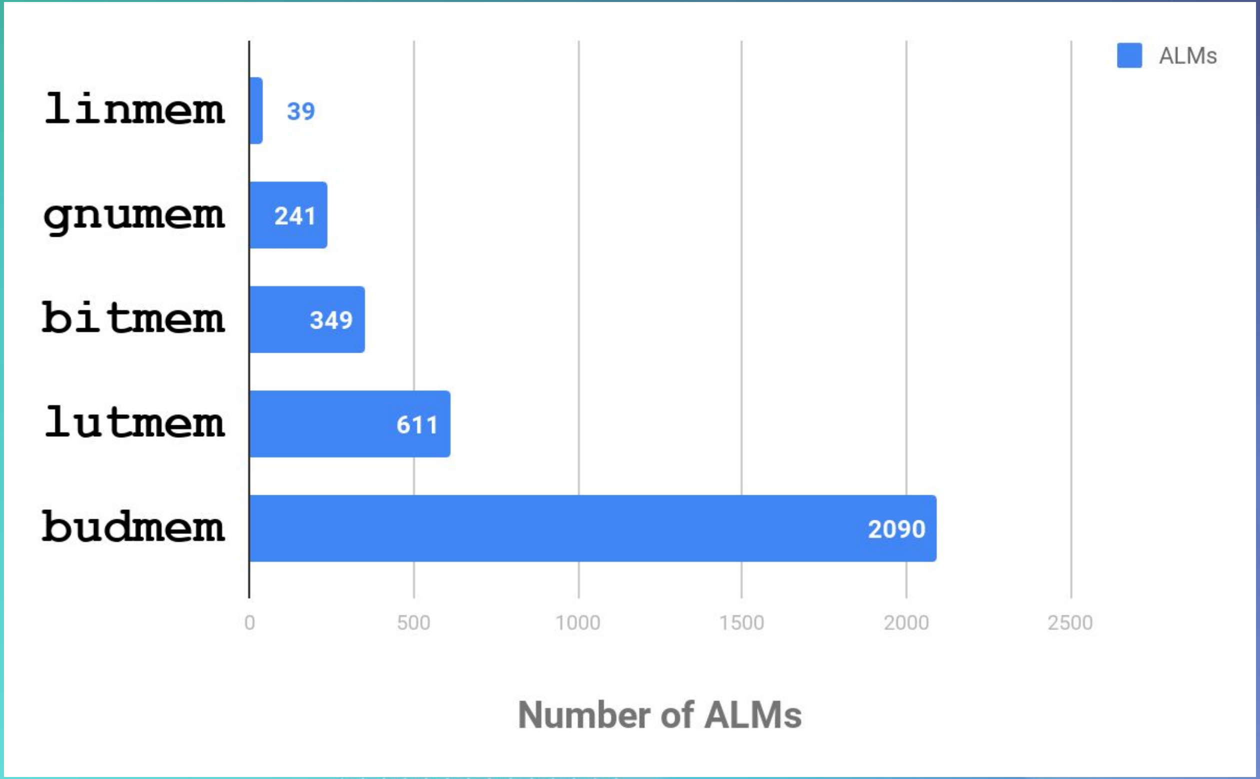
```
#TCL PARAMETERS FOR USER

set_parameter HEAP_SZ 65536
set_parameter ALLOC_S gnu
```

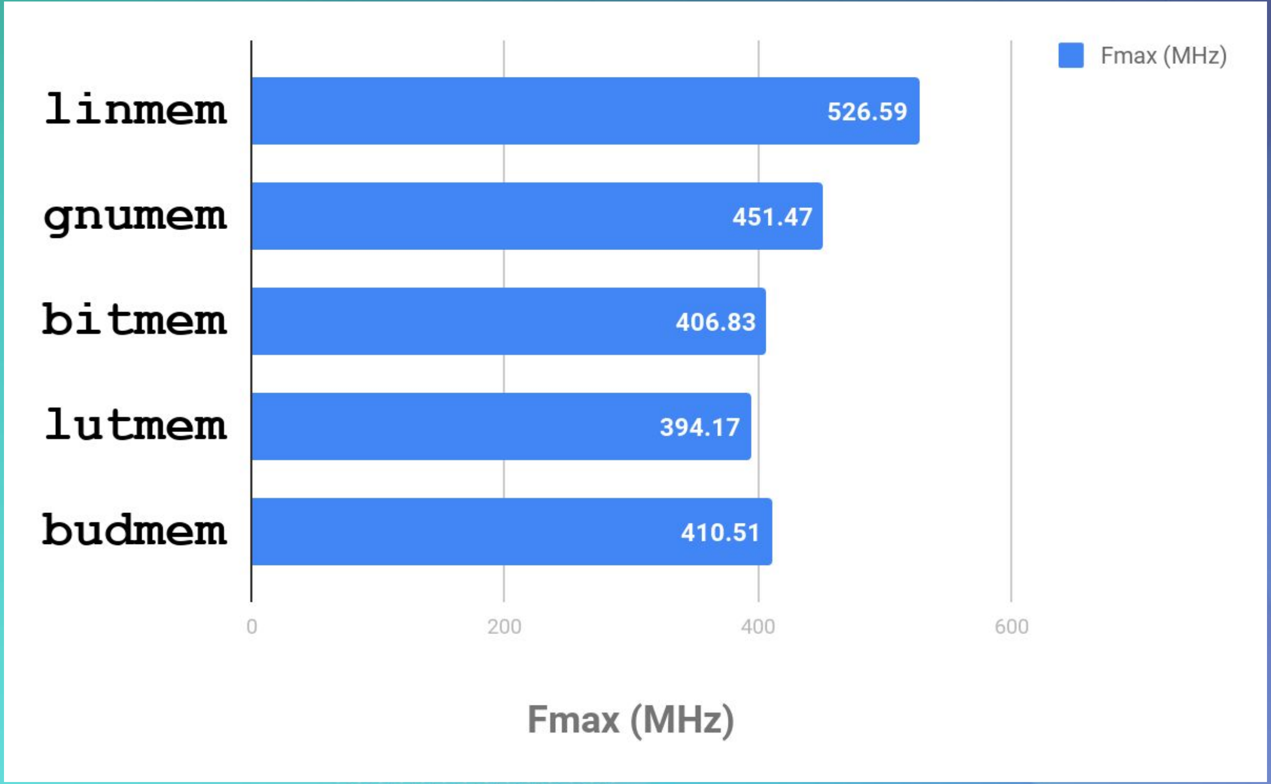
```
void check_this_out() {
      int * arr = (int*)gnu_malloc(SIZE);
      //… do stuff here
      gnu_free(arr);
}
```

Allocator Evaluation

Results: Area



Results: Performance



Benchmarks

Typical Memory Request Patterns

Random: random request, random release
Square: request-do-release
Triangular: iterative-request do iterative-release

Real world apps

list	hash
priq	dfs
stack	

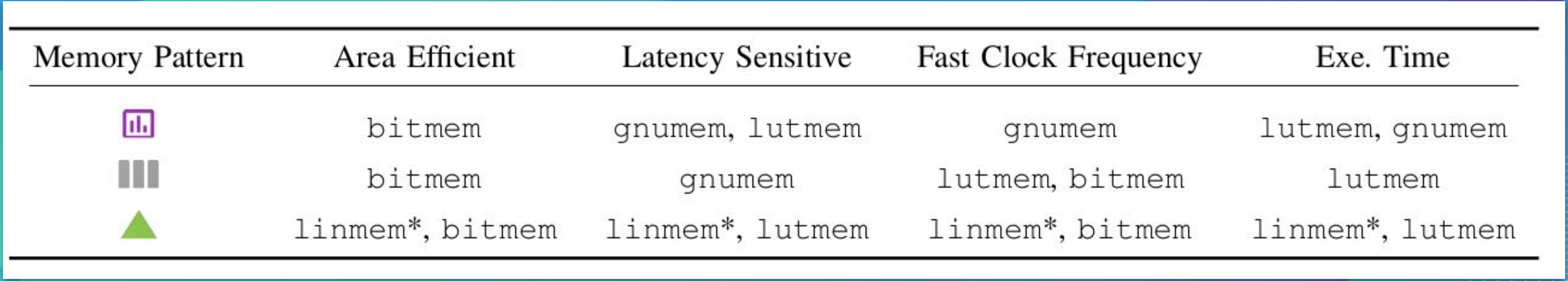
Available on Github: https://github.com/ngiambla/dmbenchhls

Take-away

Suggest an allocator based on **Memory Pattern**

AND

User Requirements



Memory Pattern	Area Efficient	Latency Sensitive	Fast Clock Frequency	Exe. Time
📊	bitmem	gnumem, lutmem	gnumem	lutmem, gnumem
▮▮▮	bitmem	gnumem	lutmem, bitmem	lutmem
▲	linmem*, bitmem	linmem*, lutmem	linmem*, bitmem	linmem*, lutmem

Conclusions

- One Allocator does not ‘rule them all’
- Performance and area are marginally affected by allocators!
- Allocators within HLS work and are useful

THANKS!

SEE ME AT THE POSTER

Downloads:
  https://github.com/ngiambla/libmem
  https://github.com/ngiambla/dmbenchhls