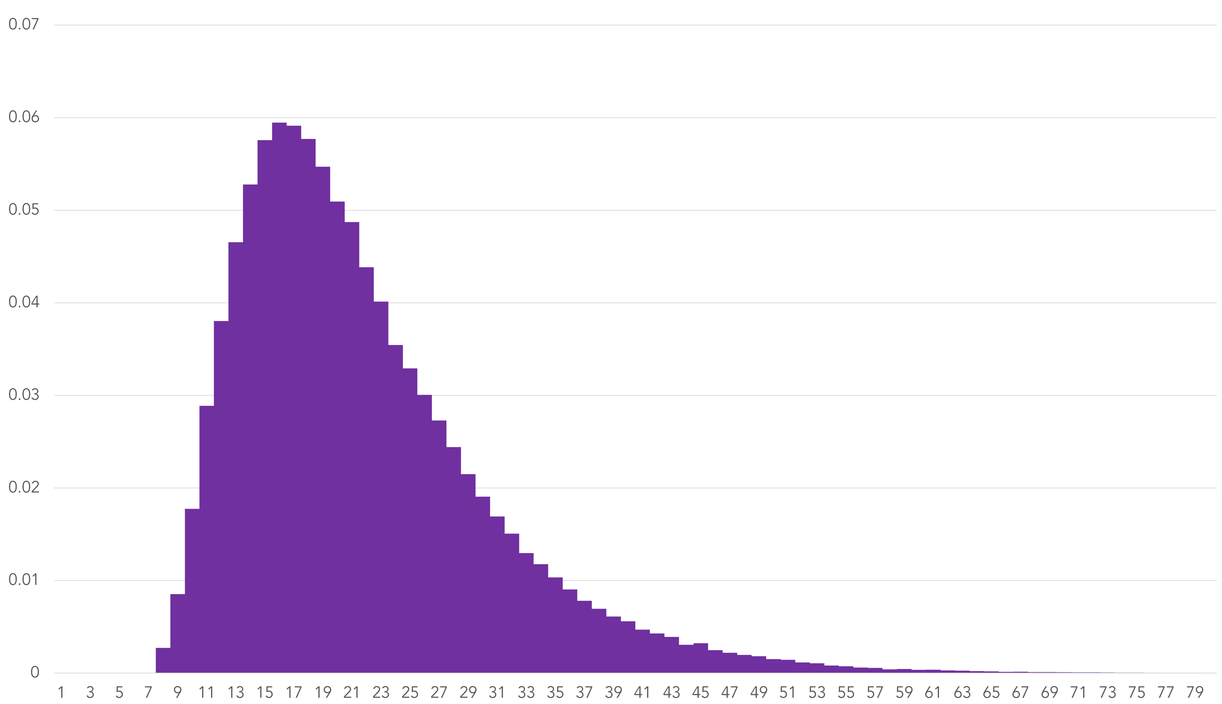
Teaching Parallel Computing

Programming in ARM Assembly Using Embedded Xinu

Benjamin Levandowski & Dr. Dennis Brylow

Motivation

- The 2013 ACM/IEEE curriculum guidelines highlight the importance of teaching parallel computing to undergraduates in higher education. [1]
- Universities have responded to this need, but have not yet integrated parallel computing concepts with assembly programming.
- While techniques for parallelizing code and using tools such as MPI and OpenMP seem to be well covered, concepts of shared versus distributed memory and race conditions are taught at an abstracted level.
- Using the educational operating system Embedded Xinu, students can write ARM assembly code that makes use of instructions meant specifically for multi-core interaction.
- By having students implement the coupon collector's problem, they have a much deeper understanding of potential issues with sharing memory and maintaining cache coherence.



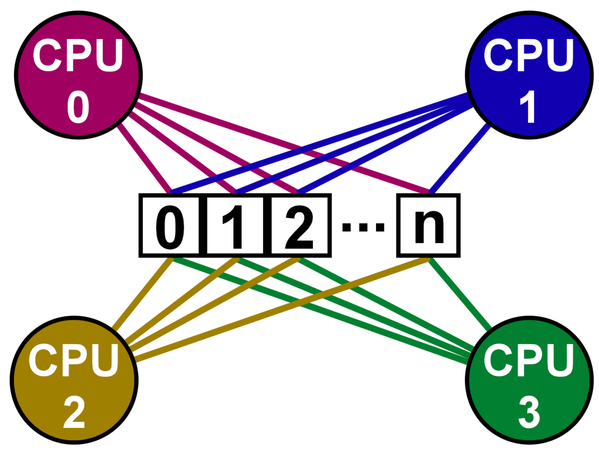
Distribution of count frequencies after running 100 million trials among four cores

Cache hierarchy on the Raspberry Pi 3

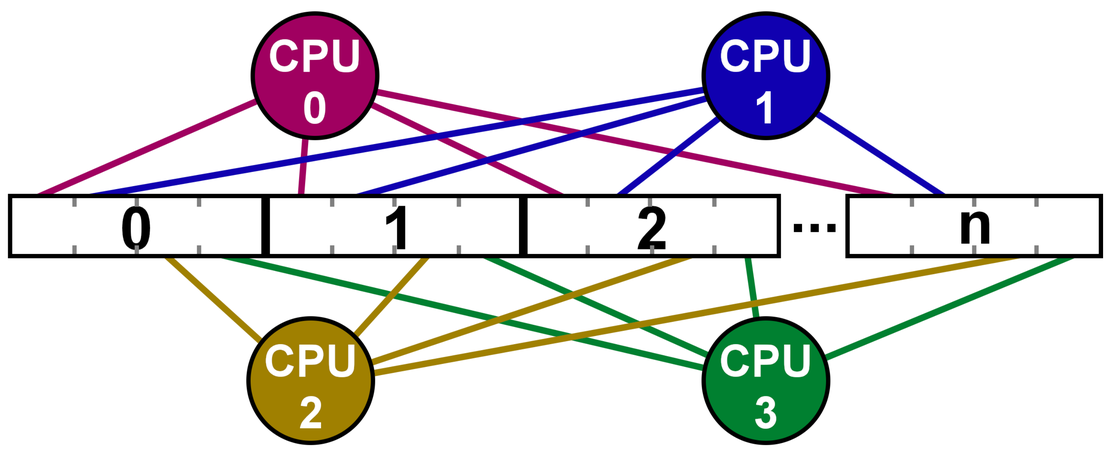
Methods

- We can create an environment on Raspberry Pi 3's using Embedded Xinu to simulate running bare metal assembly code on four cores.
- Students are tasked to implement the coupon collector's problem:
  - From an urn of n coupons, draw one at a time with replacement and record how many attempts it takes to collect all n coupons.
- Each core runs this program and updates the frequency for the corresponding result in a shared array.
- This creates the possibility for race conditions and cache coherence issues as multiple cores will have many chances to read/write the same memory.
- Both correct memory attributes and the use of exclusive instructions are required for successful operation.
- This problem additionally gives the opportunity to compare shared versus distributed memory models.

- Shared memory is easier to program and uses less space but requires that only one processor access each entry at any given time.
- Distributed memory takes up significantly more space and requires additional operations at the end of the program but is often faster.



Shared memory model for array accesses



Distributed memory model for array accesses

```
preload:
  pldw [r0]          // Invalidates other cores' cache lines
  nop                // by pre-loading data with intention to write

  ldrexb r2, [r0]    // Load lock value with exclusive access
  cmp r2, #0         // Is the lock taken?
  strexbeq r2, r1, [r0] // Attempt to store lock taken value
  cmpeq r2, #0       // Did the store succeed?
  bne preload        // Try again if lock taken or store failed
  dmb                // Data memory barrier to synchronize data
```

Code to acquire a mutex lock—note the use of exclusive access instructions

Future Work

- Use this assignment to measure growth in college sophomores taking Hardware Systems at Marquette University
- Continue to develop curriculum that gives a better understanding of parallel computing at a low level
- Assess weak points in current model to ensure students focus on parallel computing aspect instead of struggling too much with ARM assembly

Valparaiso University

Reference

[1] The ACM/IEEE Joint Task Force on Computing Curricula. 2013. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. (December 2013).

Acknowledgements

This work was sponsored in part by National Science Foundation REU Site grant #ACI-1461264, 'Computation Across the Disciplines', at Marquette University.
Thank you to Dr. Debbie Perouli as well as those who made Embedded Xinu possible.

MARQUETTE UNIVERSITY