Hashing Strategies for the Cray XMT
MTAAP 2010

Eric Goodman (SNL)
David Haglin (PNNL)
Chad Scherrer (PNNL)
Daniel Chavarría-Miranda (PNNL)
Jace Mogill (PNNL)
John Feo (PNNL)

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

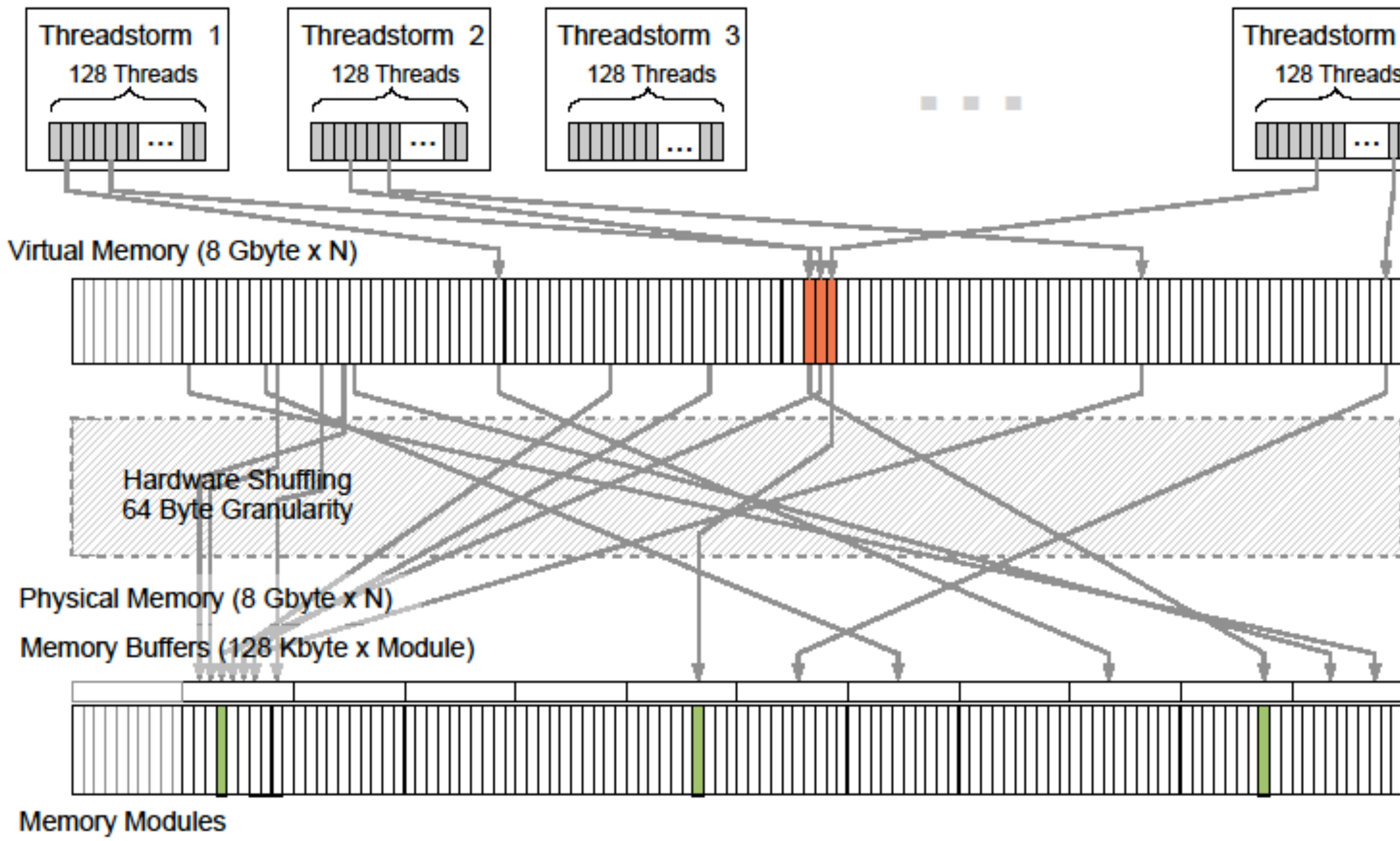
Hash Tables Background

- **Fundamental computer science concept and data structure**
  - First described in 1953
- **A fast and scalable implementation for the Cray XMT has been lacking**
- **Our contribution:**
  - Two scalable algorithms that perform well on uniform and power law distributions
    - Open addressing with linear probing – static table sizes
    - Hashing with Chaining and Region-based Memory Allocation (HACHAR) – dynamic table sizes

The Cray XMT

- **Shared memory machine**
  - 128 threads per processor
  - 8 GB of globally accessible memory per processor
  - 500 MHz
- **Custom compiler**
  - Lightweight synchronization mechanisms
    - Full/empty bit
    - readfe, writeef
    - int_fetch_add
  - Implicit Parallelism
    - For loops
  - Explicit Parallelism
    - Futures
- **Hashing Considerations**
  - Potential for memory contention with frequently occurring keys

Memory



Threadstorm 1
128 Threads
Threadstorm 2
128 Threads
Threadstorm 3
128 Threads
Threadstorm N
128 Threads
Virtual Memory (8 Gbyte x N)
Hardware Shuffling
64 Byte Granularity
Physical Memory (8 Gbyte x N)
Memory Buffers (128 Kbyte x Module)
Memory Modules

Avoiding Memory Contention: Two-Step Acquisition

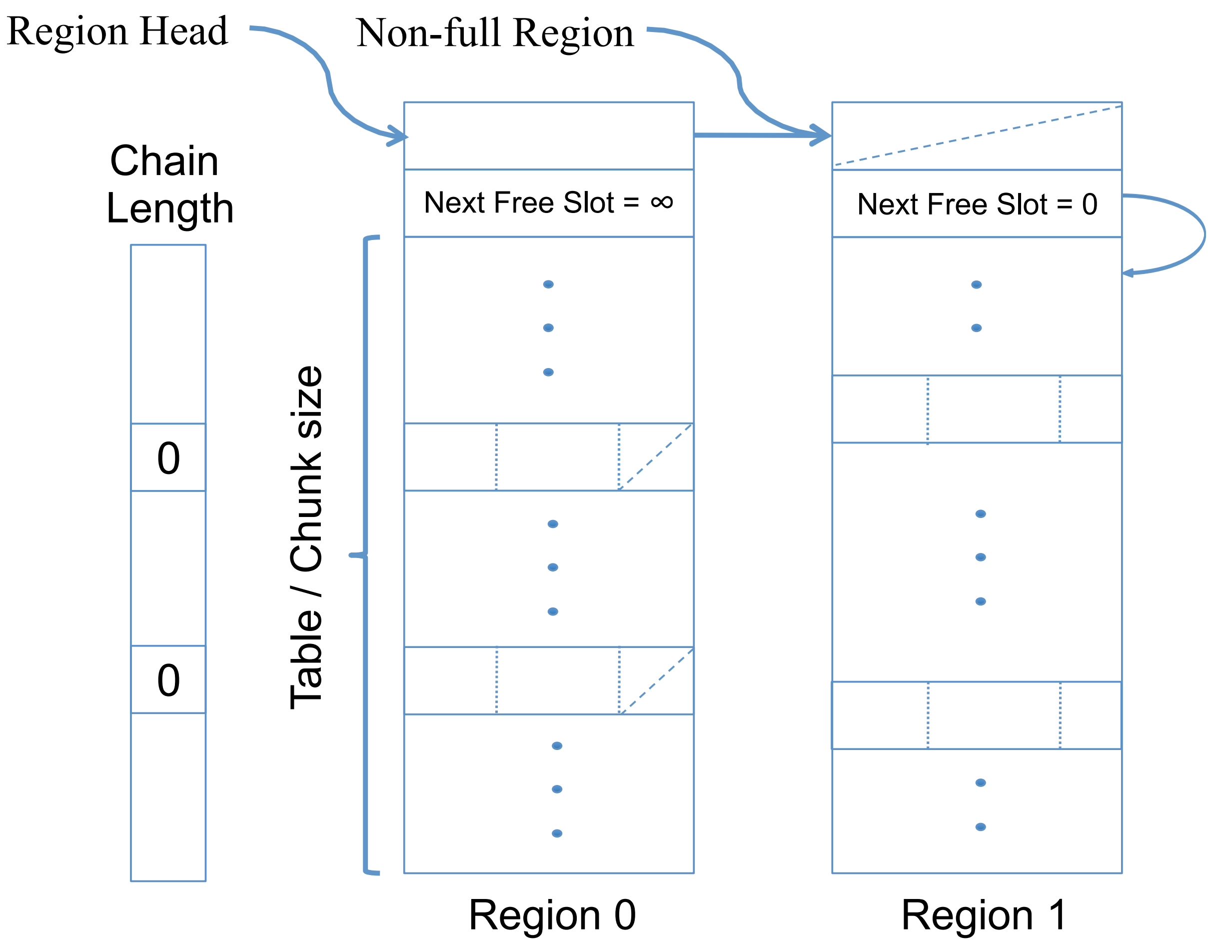
```
Procedure: TwoStepAcquireAttempt
Parameters: (location, key)

  // first check without locking
if location is empty then
    lock(location)
      // then check with the lock
    if location is still empty then
        Reserve location for key
        unlock(location)
        return true
    end if
      // Another thread modified the structure before
      // we could acquire location
    unlock(location)
end if
return false ▷ Caller needs to continue searching
```

Open Addressing with Linear Probing

- **Data Structures**
  - Key, Value, and Occupied arrays all of size *table_size*
- **General Procedure**
  - Get an index for a key with *hash(key) % table_size*
  - If slot is claimed, linearly probe forward until open spot is found
- **How to Claim a spot**

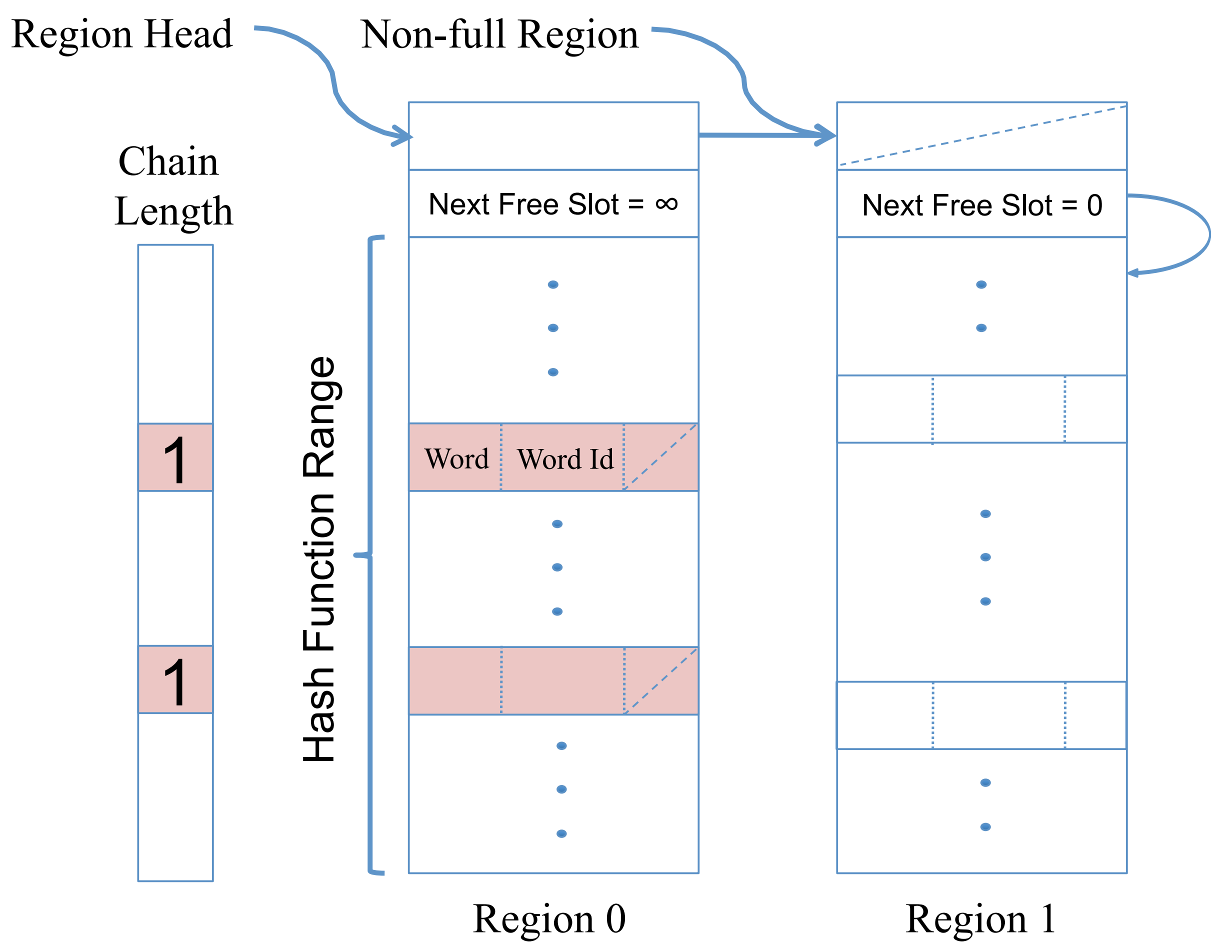
```
int probed = occupied[i]; //non-blocking read
if(probed > 0) { //already taken
  if(compare(keys[i],key)) {
    return i;
  }
}
else { //not taken yet
  probed = readfe(&occupied[i]); //blocking read
  if (probed == 0) { //not taken yet
    keys[i] = key;
    writeef(&occupied[i], 1) //unlock the slot
    return i;
  }  else { //already taken
    if (compare(keys[i], key)) { // the right slot
      writeef(&occupied[i], 1); //unlock the slot
      return i;
    }
    writeef(&occupied[i], 1);
  }
```

Global HACHAR – Initial Data Structure



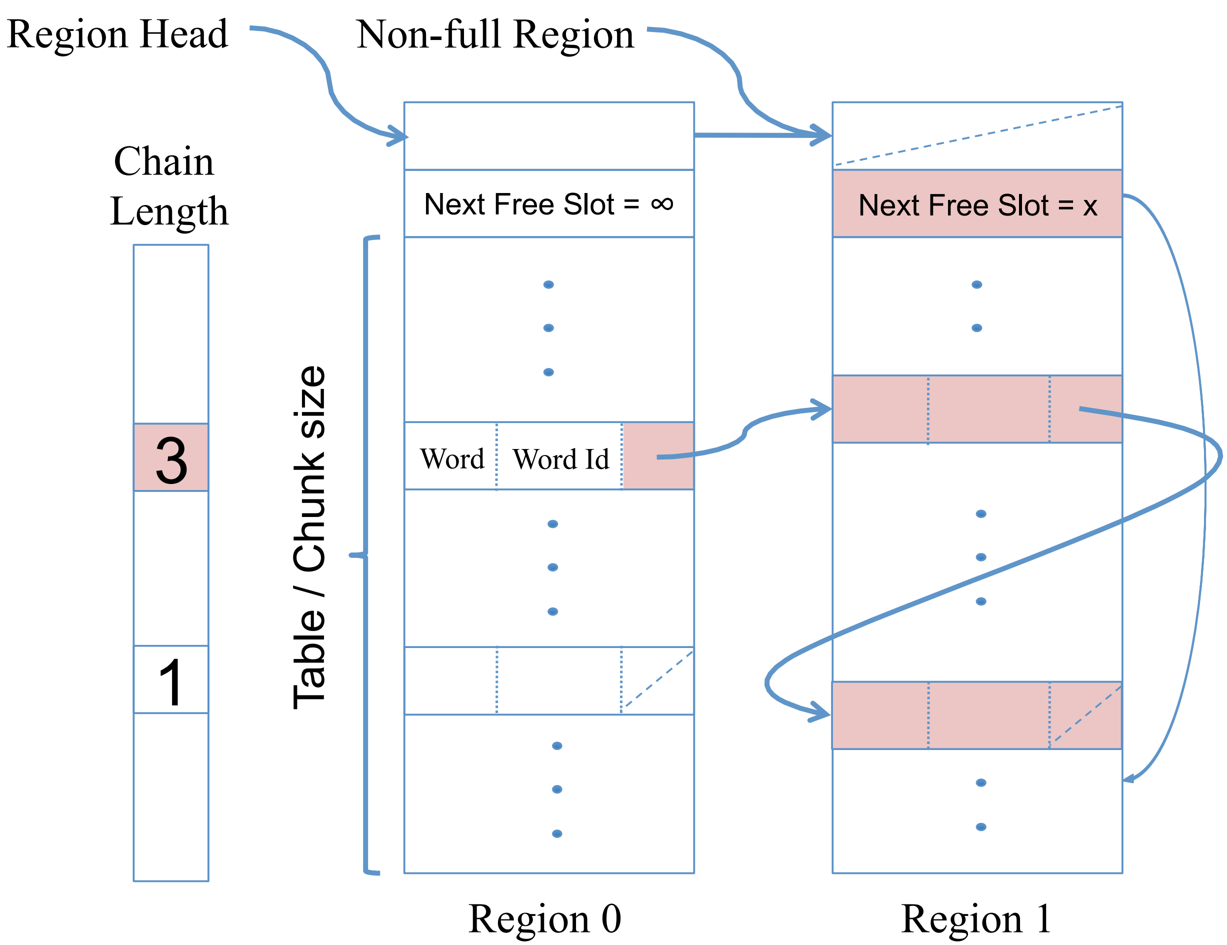
- Use "two-step acquire" on length, region linked list pointers, chain pointers.
- Use int_fetch_add on "next free slot" to allocate list node.

Global HACHAR – Two items inserted



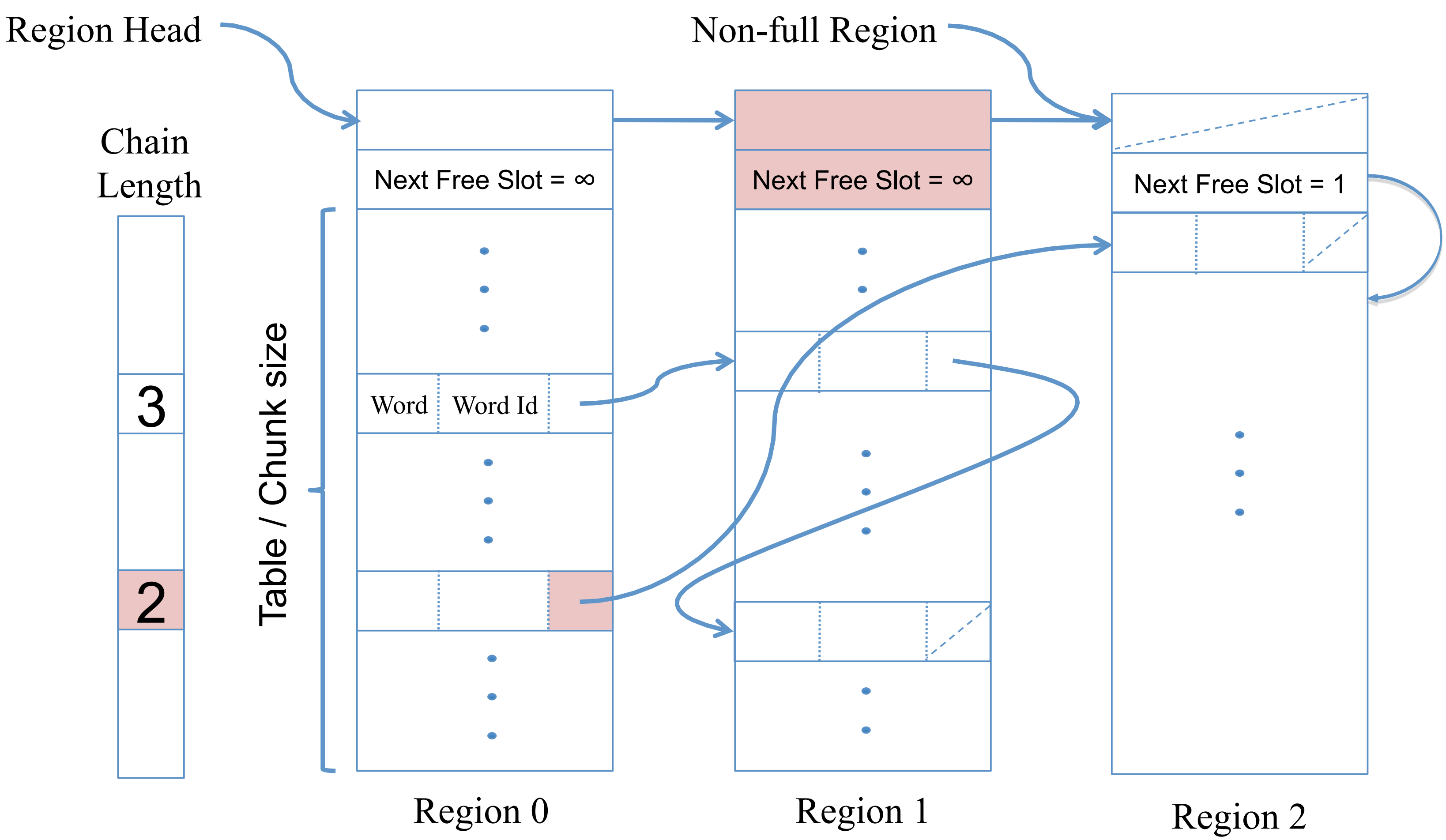
- "locked" length and inserted into "head of list"
- Potential contention only on length
- List node shows example for Bag Of Words

Global HACHAR – Collisions



- Lookup: walk chain, no locking
- Malloc and free limited to the few region buffer
- Growing a chain requires lock of only last pointer (int_fetch_add length)

Global HACHAR – Region Overflow



Pacific Northwest
NATIONAL LABORATORY

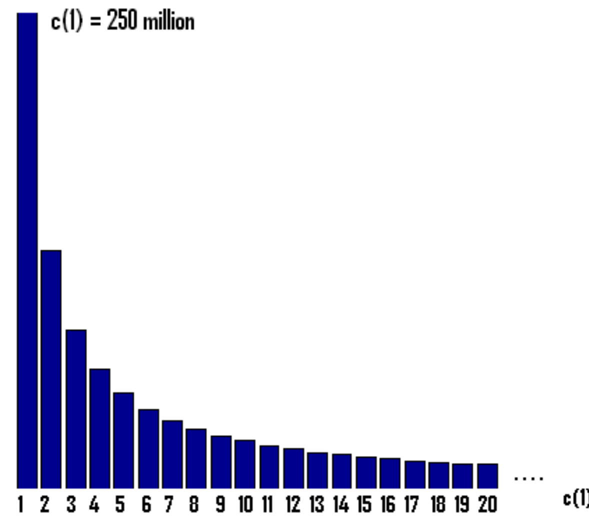
Proudly Operated by Battelle *Since 1965*

The Data Sets

- **Uniform Random Data**
  - 5 billion integers in $[-2^{63}, 2^{63}-1]$
- **Zipfian Integers**
  - Zipf's law: Element of rank k occurs 2 times more often than element of rank k/2.

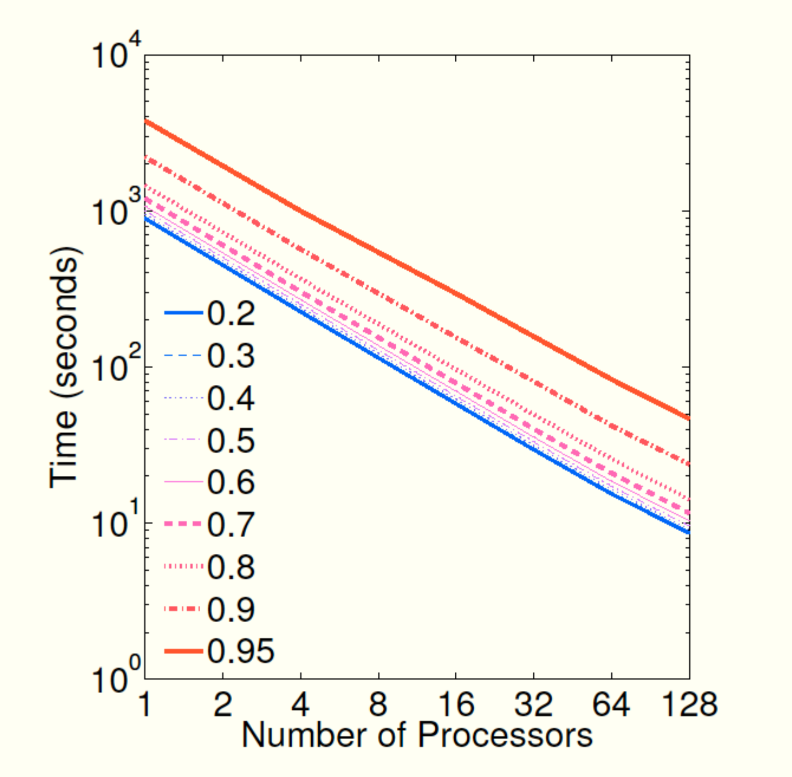
$$c(k) = \left\lfloor \frac{c(1)}{k} \right\rfloor$$

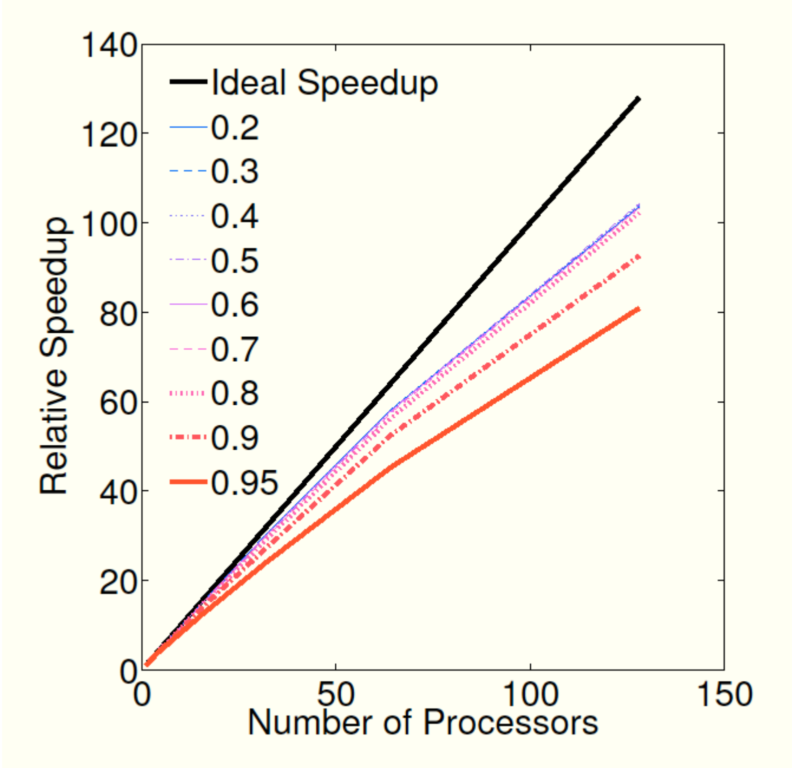
  - ~5 billion integers in [1, 250 million]
- **Wikipedia Instance**
  - 1.42 billion strings
  - 16.3 million unique strings



Sandia National Laboratories

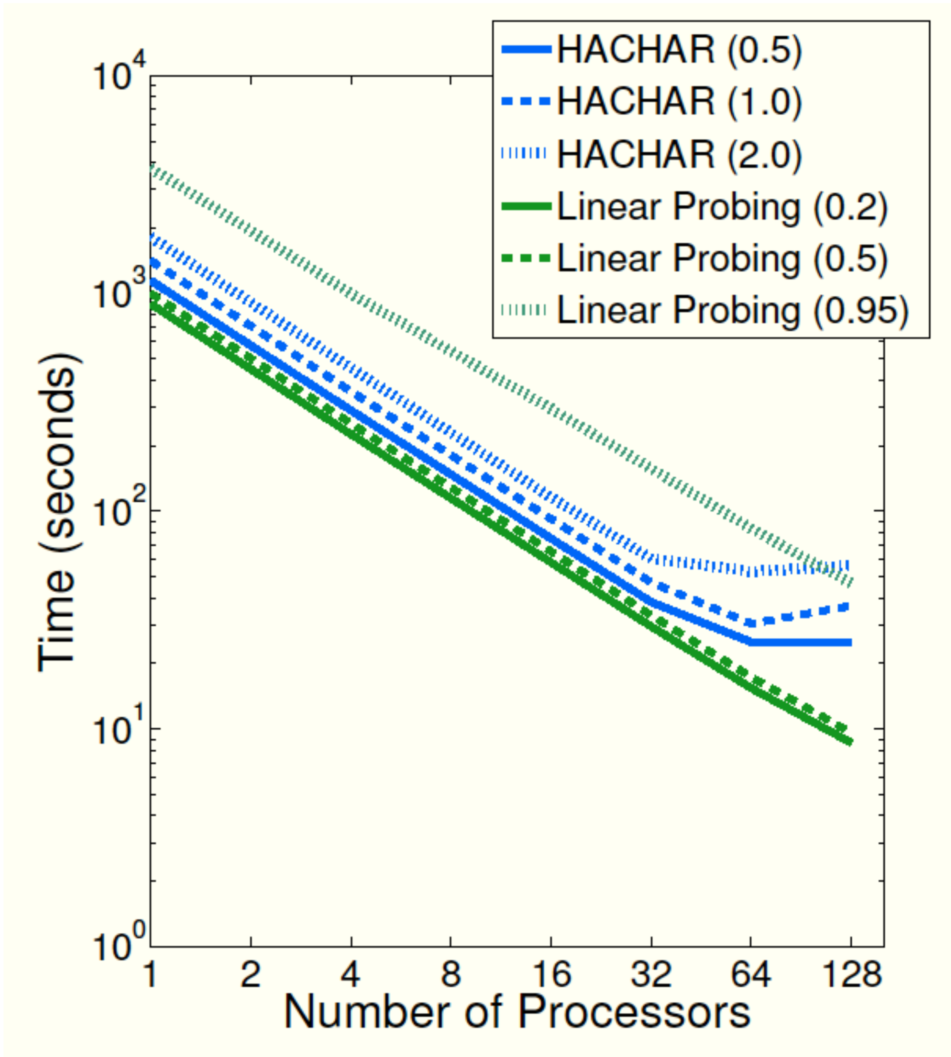
Linear Probing on Uniform Random Data (5 Billion)





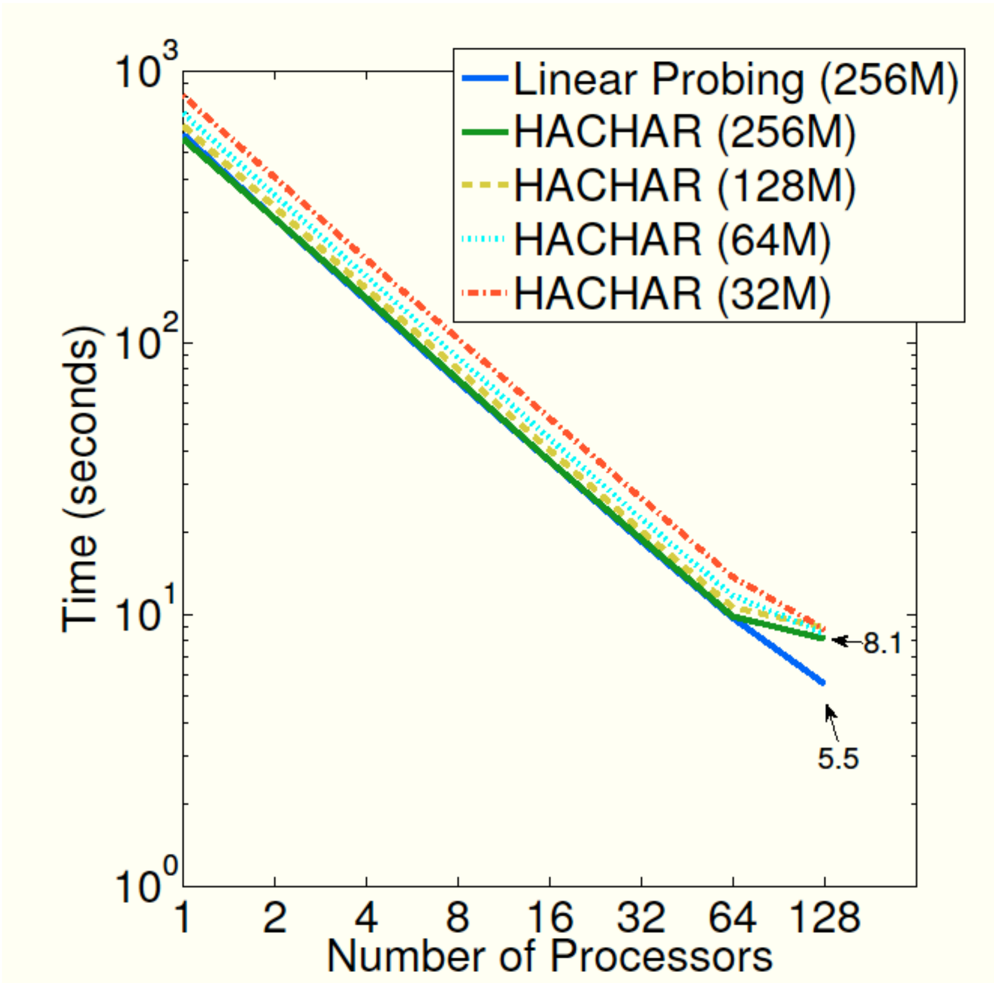
Sandia National Laboratories

Comparison with HACHAR, Uniform Random Data



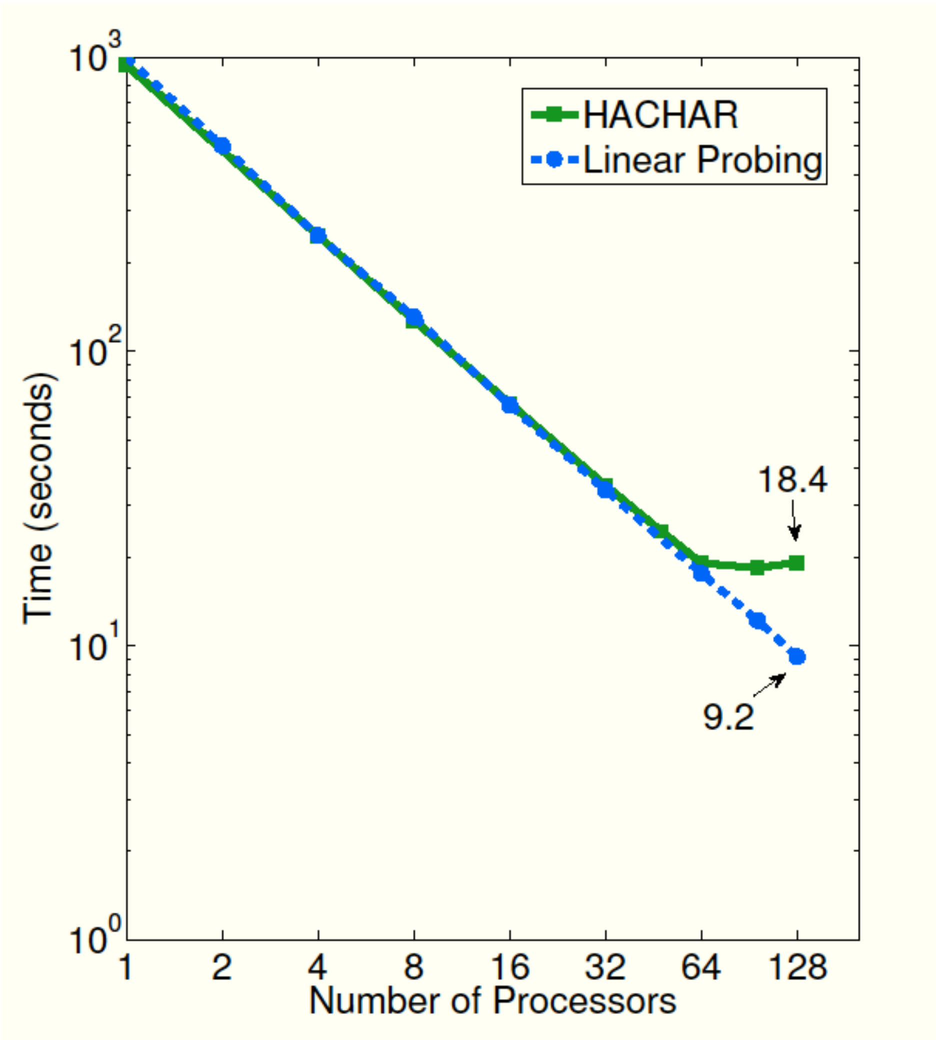
Sandia National Laboratories

Comparison on Zipfian Integers (~5 Billion)



Sandia National Laboratories

Wikipedia Instance



- 1.42 billion tokens
- 16.3 million unique keys
- Linear Probing used table with 64 million slots
- HACHAR used 32 million

Hashing Conclusions and Future Work

- **Two robust and fast solutions for hashing**
  - Works well on both uniform random and power law data
- **Linear Probing best option when number of keys is known**
- **HACHAR best option when number of keys is not known**
  - Performs well even with large load factors
- **Two-step acquisition process main contributing factor behind performance**
  - May work well in other contexts
- **Hash-reduce strategy**
  - May scale better