A RISC-V Extension for the Fresh Breeze Architecture

Jack B. Dennis

Willie Lim

MIT CSAIL

First Workshop on
Computer Architecture Research with RISC-V (CARRV 2017)
Boston, MA, USA, October 14, 2017

Massachusetts
Institute of
Technology

CSAIL

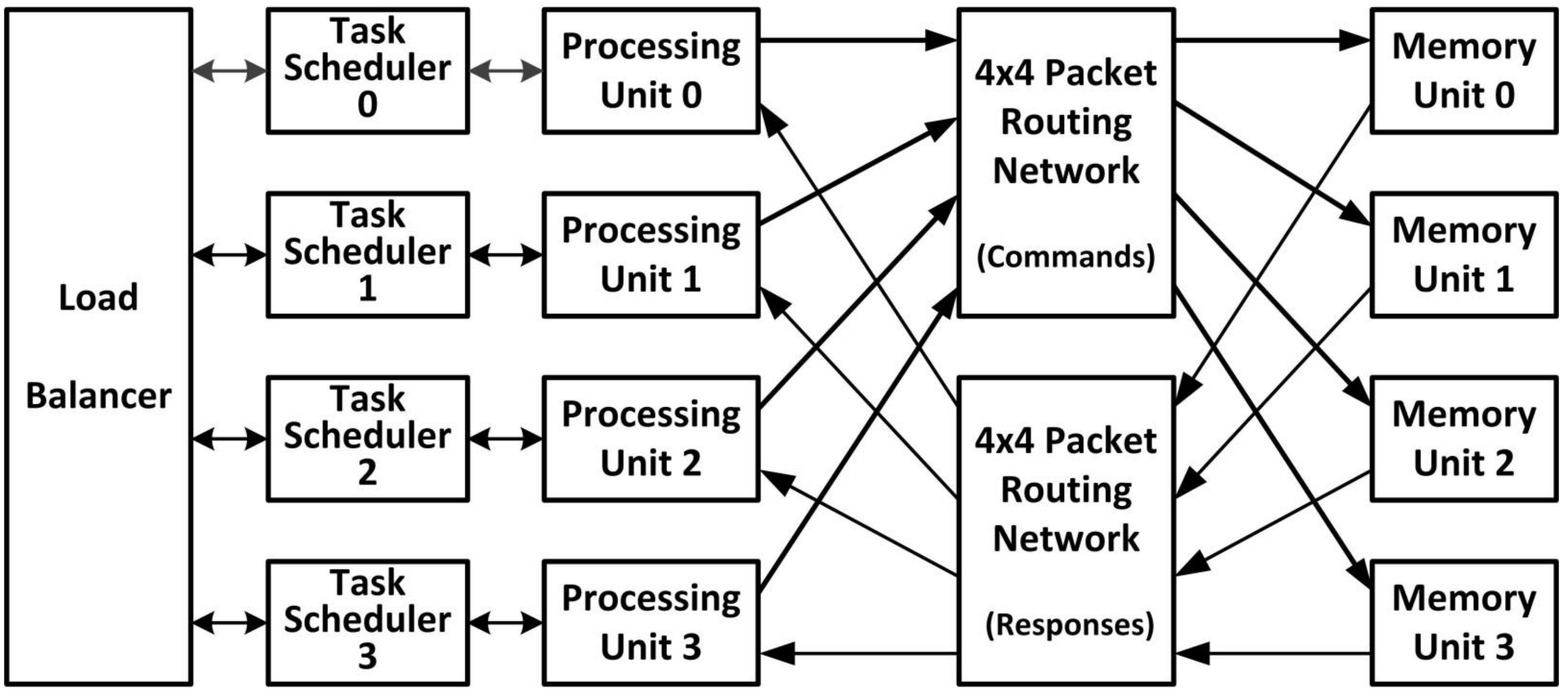
Talk Outline

- Fresh Breeze – Architecture, Programming Model, Codelets, and Memory Model
- RISC-V Extensions
  - AutoBuffer Implementation
  - Trees of Chunk Based Memory Structure
  - Garbage Collection
  - Instructions for Tasking
- Observations
- Current System Limitations
- Future Work

Fresh Breeze

- A Programming Model and System Architecture
- Two Key Concepts:
  - Tree-of-chunks Memory Model
  - The Codelet Model for fine-grain tasking
- Goals:
  - Energy-efficient architecture for exascale computing
  - Satisfy requirements for modular software construction

A 4-Core Fresh Breeze System



The Fresh Breeze Memory Model

- A chunk (1024-bit) – holds 16 64-bit scalars or handles (to chunks)
- All data and program objects represented by trees of fixed size memory chunks.
- Memory chunks are write once.
- Distributed memory hierarchy without consistency issues.
- Low overhead memory management with reference count garbage collection.
- Global name space for all data and programs.
- Supports capability based security.

CSAIL

Fine Grain Tasking with Codelets

- Fresh Breeze Codelets.
  - The unit for scheduling processing resources.
  - Contains block of instructions executed to completion. Non pre-emptable.
  - Activated by availability of input data objects. Signals successor codelets when results are ready. (Data Flow).

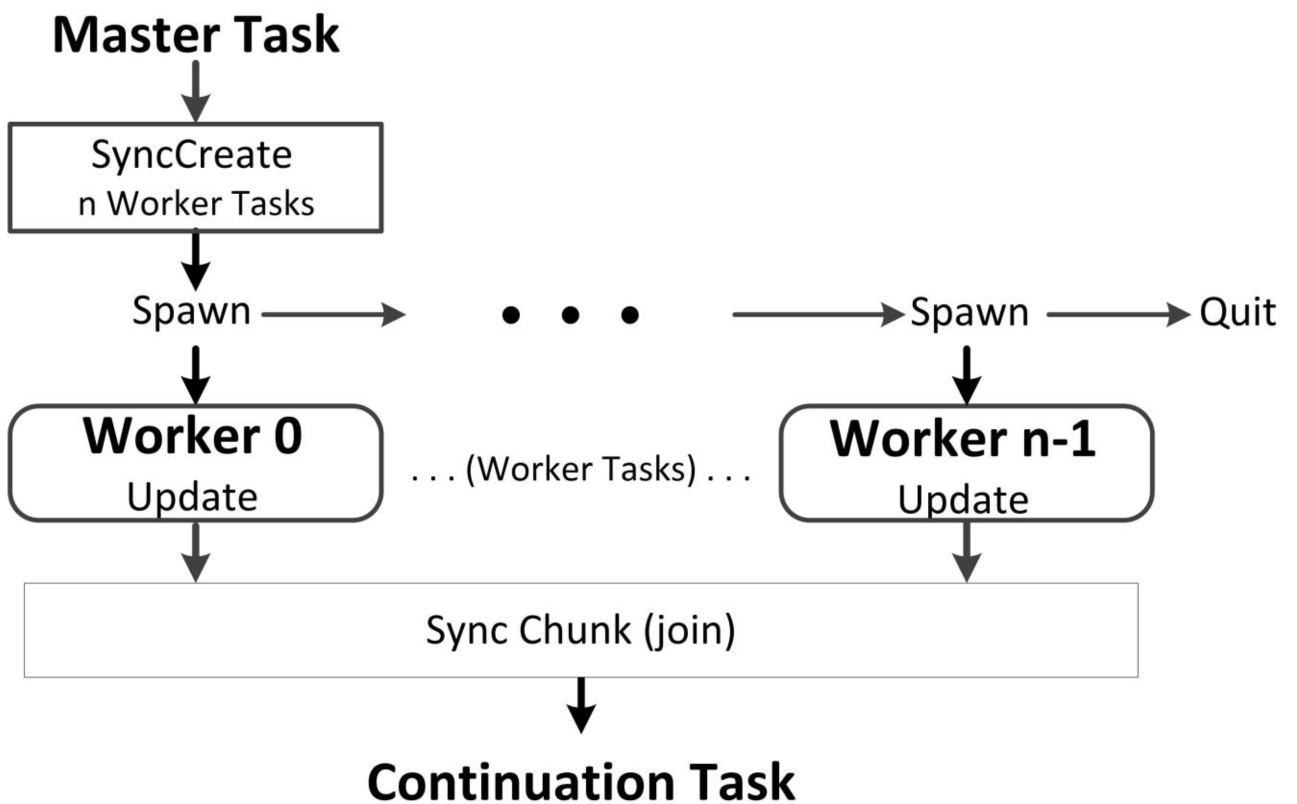
- Hardware supported scheduling and load distribution.

Running funJava Programs

- funJava is a functional subset of Java.
- The funJava compiler has four stages:
  - *javac*: Produce bytecode files from the funJava code.
  - *Convert*: Construct a Data Flow Graph (DFG) from the bytecode for each Java method.
  - *Transform*: Identify loops amenable to data parallel implementation (map or reduce) and construct a set of DFGs representing abstract codelets for each method.
  - *Generate*: Convert each abstract codelet DFG into real codelets.
- Load and run codelets on a target machine using the *Kiva* simulator.

CSAIL

Tasking Model – Spawning Team of Workers

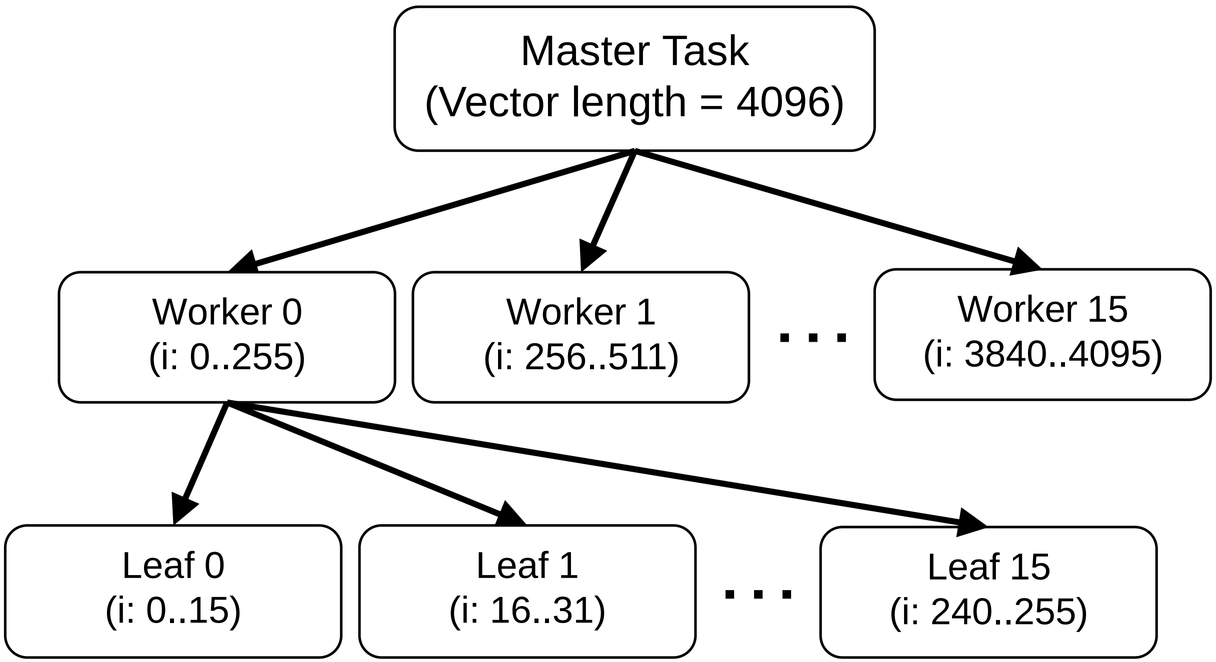


funJava Sample Code

```
long dotProduct(long[] a, long[] b, int len) {
     long sum = 0;
     for (int i = 0; i < len; i++) {
          sum = sum + a[i] * b[i];
     }
     return sum;
}
```

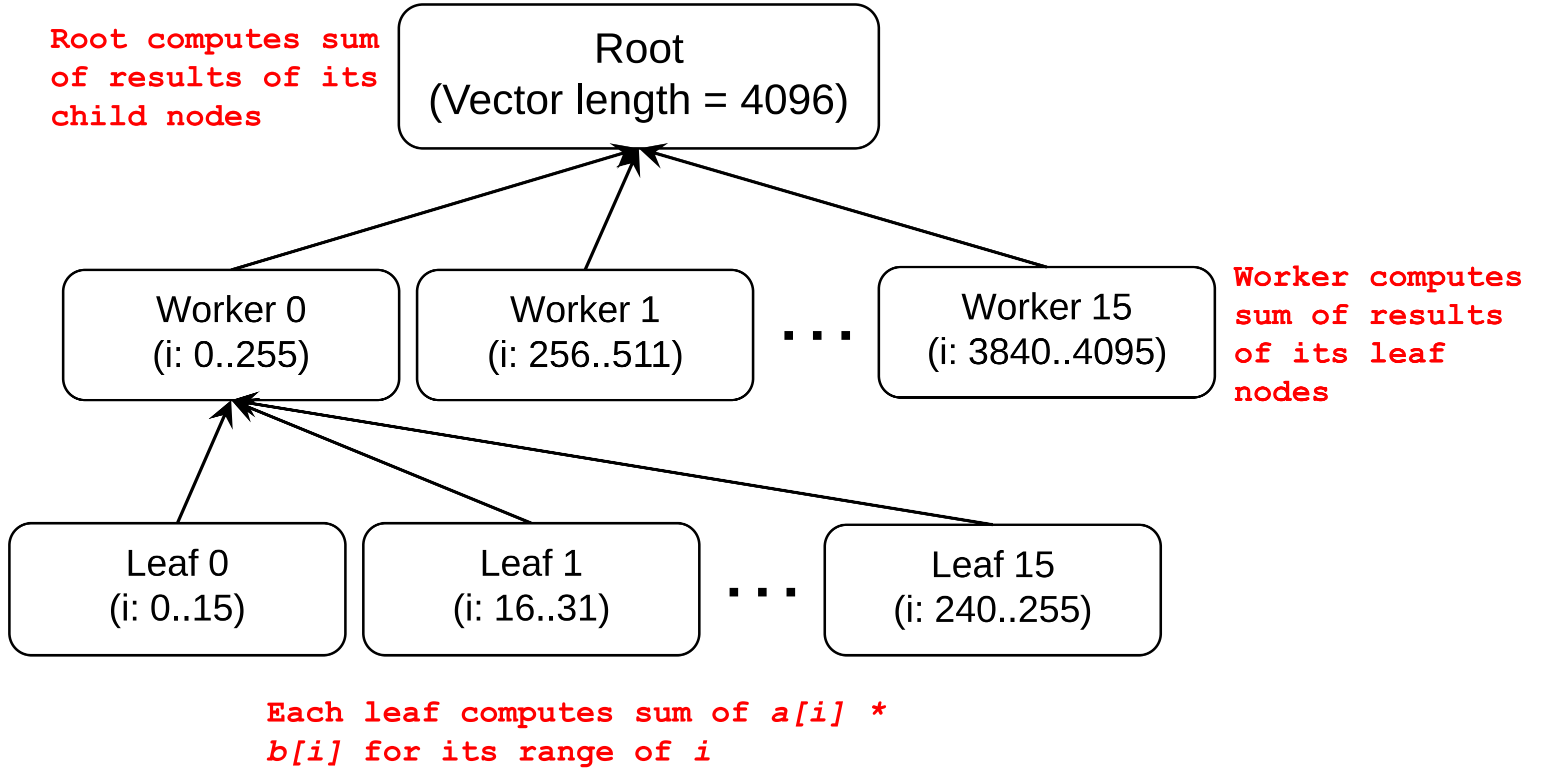

CSAIL

Tree of Tasks (Map)



Each leaf computes sum of *a[i] * b[i]* for its range of *i*.

Tree of Tasks (Reduce)



Traverse Codelet

```
Codelet 1:
   19]: LMove S0: 0; -> D: 26
   20]: IMove S0: 2; -> D: 28
   21]: IMove S0: 41; -> D: 29
   22]: SyncCreate Code: 3;
           sigCnt: 11; itemCnt: 41 -> D: 26;
           argsBase: 13 argsCnt: 2
   23]: ISet 0 -> D: 28
   24]: IfIGeq S0: 28; S1: 41; Lab: 41
   25]: ISub S0: 41; S1: 0; LV: 1; -> D: 52
   26]: IfINeq S0: 28; S1: 52; Lab: 29
   27]: IMove S0: 42; -> D: 53
   28]: Jump Lab: 30
   29]: IMove S0: 37; -> D: 53
   30]: ReadFull H: 4; Off: 28; -> D: 54
   31]: ReadFull H: 6; Off: 28; -> D: 56
   32]: IMul S0: 28; S1: 37; -> D: 58
   33]: IAdd S0: 8; S1: 58; -> D: 59
   34]: LMove S0: 54; -> D: 30
   35]: LMove S0: 56; -> D: 32
   36]: IMove S0: 59; -> D: 34
   37]: IMove S0: 53; -> D: 35
   38]: TaskSpawn Code: 2; argsBase: 13; argsCnt: 5
   39]: IAdd S0: 28; S1: 0; LV: 1; -> D: 28
   40]: Jump Lab: 24
   41]: TaskQuit
```


CSAIL

Leaf Codelet

```
Codelet 2:
    0]: ISet 1 -> D: 10
    1]: ISet 0 -> D: 11
    2]: LSet 0 -> D: 12
    3]: IMove S0: 11; -> D: 14
    4]: LMove S0: 12; -> D: 16
    5]: IfIGeq S0: 14; S1: 9; Lab: 13
    6]: IAdd S0: 14; S1: 8; -> D: 15
    7]: ReadFull H: 6; Off: 14; -> D: 18
    8]: ReadFull H: 4; Off: 14; -> D: 20
    9]: LMul S0: 18; S1: 20; -> D: 22
   10]: IAdd S0: 14; S1: 0; LV: 1; -> D: 14
   11]: LAdd S0: 16; S1: 22; -> D: 16
   12]: Jump Lab: 5
   13]: SyncUpdate Sync: 0; Off: 2; Data: 16
   14]: TaskQuit
```

a[i] * b[i]

Sum of products

CSAIL

Reduce Codelet

```
Codelet 3:
  0]: ISet CV: 1 -> D: 7
  1]: ISet CV: 0 -> D: 10
  2]: LSet CV: 0 -> D: 8
  3]: IfIGeq S0: 10; S1: 5; Lab: 8
  4]: ReadFull H: 0; Off: 10;  -> D: 12
  5]: LAdd S0: 8; S1: 12; -> D: 8
  6]: IAdd S0: 10; S1: 7; LV: 1; -> D: 10
  7]: Jump Lab: 3
  8]: SyncUpdate Sync: 4; Off: 4; Data: 8
  9]: TaskQuit
```

Sum of sums of products

RISC-V Extensions

- Instructions for building and accessing data objects using the Tree-of-chunks Memory Model

- Instructions to support spawning and coordination of worker tasks.

Task Record

- Represents a task in the Core Scheduler’s queue

- Moved between Core Schedulers’ queues as directed by the Load Balancer

- Has fields:

  codeIdx (16 bits): The index of the codelet to be executed.

  argsCnt(4 bits): The number of arguments needed for the task.

  argsList (64 bits): The handle of a chunk containing the arguments.

CSAIL

The AutoBuffer

- Used in place of the usual level one cache.
- Holds several memory chunks for direct access by the processor.
- For direct access, each processor register has an extra *index* field and *valid* bit.
- *index* and *valid* are set by the ChunkCreate instruction, or when the chunk is loaded into the AutoBuffer in response to a read instruction.

CSAIL

Memory Instructions

ChunkCreate (*dest*)

WriteFull (*handle, index, value*)

WriteLeft (*handle, index, value*)

WriteRight (*handle, index, value*)

ReadFull (*dest, handle, index)*

ReadLeft (*dest, handle, index)*

ReadRight (*dest, handle, index)*

Garbage Collection

- To automatically reclaim free chunk space
- Garbage collection done by hardware
- Use the reference count (RC) scheme with RC:
  - Held as metadata for each chunk in each Memory Unit
  - Accessed using the handle of the chunk
  - Initially zero
  - Incremented by one by the ChunkCreate instruction and whenever the handle is copied
  - Decremented when a task with chunk handle terminates
  - When zero the chunk is marked free and the reference count of any chunks referenced by handles in the freed chunk are decremented

CSAIL

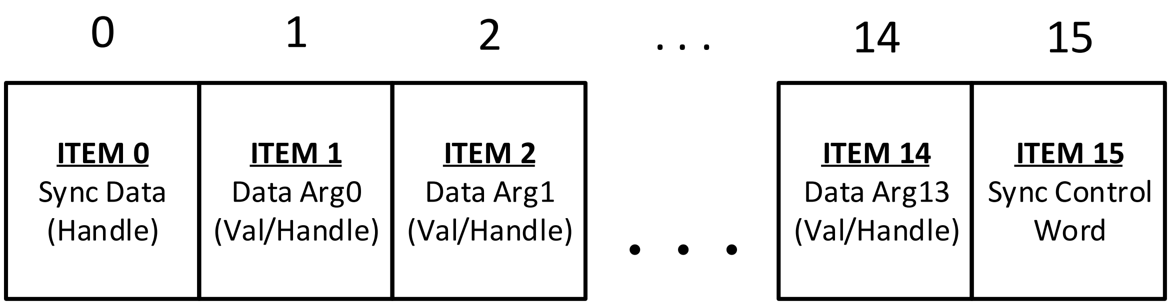
Sync Chunk

A sync chunk (1024-bit) contains 16 memory items (or 64-bit word):

Item 0: handle of the sync data chunk; null if argsCnt is zero

Items 1 - 14: hold up to 14 argument items

Item 15: Sync Control Word



CSAIL

Sync Control Word

codeIdx (16 bits): Index of Codelet

flags (16 bits): A boolean flag for each of itmCnt work tasks; used just to check that each task contributes exactly one result item to the sync data chunk.

- (2 bits): (not used)

argsCnt (6 bits): # of args

sigCnt (6 bits): (Not used - for streams

sigIdx (6 bits): (Not used - for streams)

itmCnt (6 bits): # of data

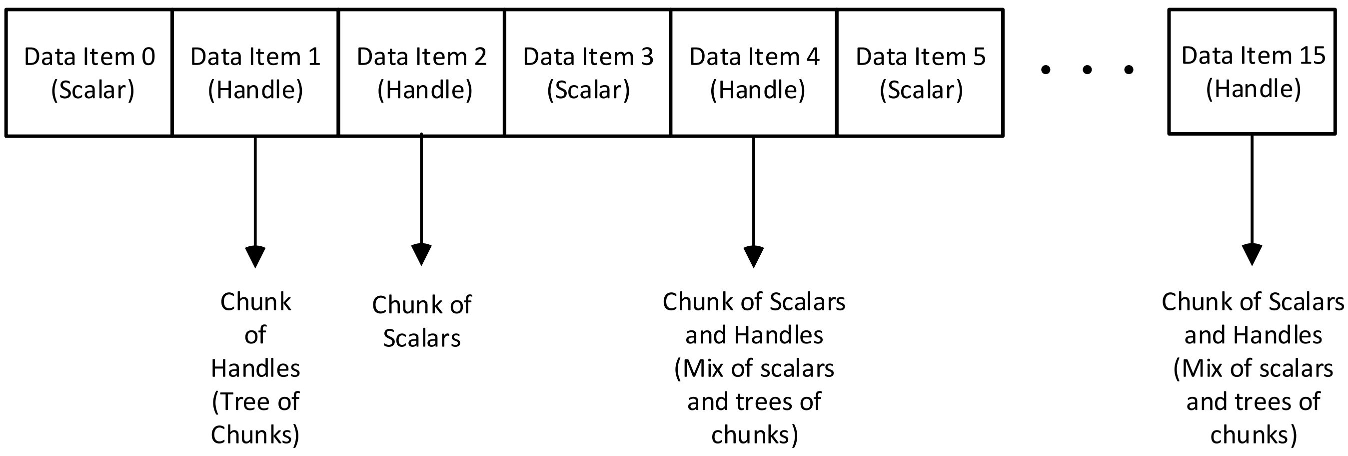
itmIdx (6 bits): # of items counter (increment counter)

codeIdx	flags	-	argsCnt	sigCnt	sigIdx	itmCnt	itmIdx
(16)	(16)	2	(6)	(6)	(6)	(6)	(6)

64 48 32 30 24 18 12 6 0

Sync Data Chunk

- A chunk to hold at most 16 handles to data items
- Example: N data item handles with M data items (scalars & handles) total



Instructions for Tasking

SyncCreate (*dest, code, count*)

Create a *SyncChunk* for a task to be executed upon completion of the current task.

TaskSpawn (*code, args, sync, index)*

Puts a *TaskRecord* in the scheduler queue.

SyncUpdate (*sync, index, result*)

Puts a worker result in the data chunk of a *SyncChunk*.

TaskQuit ()

Observations

- From machine learning and linear algebra performance studies – for sufficiently large input data, computation performance scales linearly as the number of cores increases due to:
  - The ability to decompose computations into many parallel data driven tasks.
  - Efficient load balancing using hardware
- Similar observation for experiments involving running several different funJava programs at the same time.
- Task can be from the same or different computation

Current System Limitations

- Running out of JVM heap space.
- Long simulation time.
- The need for garbage collection by the Java VM between successive simulation runs.

Future Work

- Multi-host version of Kiva
- Compiler enhancements to support streams and transactions
- Build an FPGA-based prototype system
  - Model a Fresh Breeze multi-core processor with extended RISC-V Processing Cores.
  - Use BlueDBM facility in the Computation Structures Group (CSG) of MIT CSAIL

CSAIL