GoblinCore64:

A RISC-V Extension for Data Intensive Computing

John D. Leidel, Xi Wang, Yong Chen

June 29, 2015

Data-Intensive Scalable Computing Laboratory (DISCL)

TEXAS TECH UNIVERSITY

Overview

- Why a RISC-V Extension?
- GC64 Architecture Organization
- GC64 ISA Extension(s)
- Future Development

Motivations & goals behind GoblinCore64

WHY A RISC-V EXTENSION?

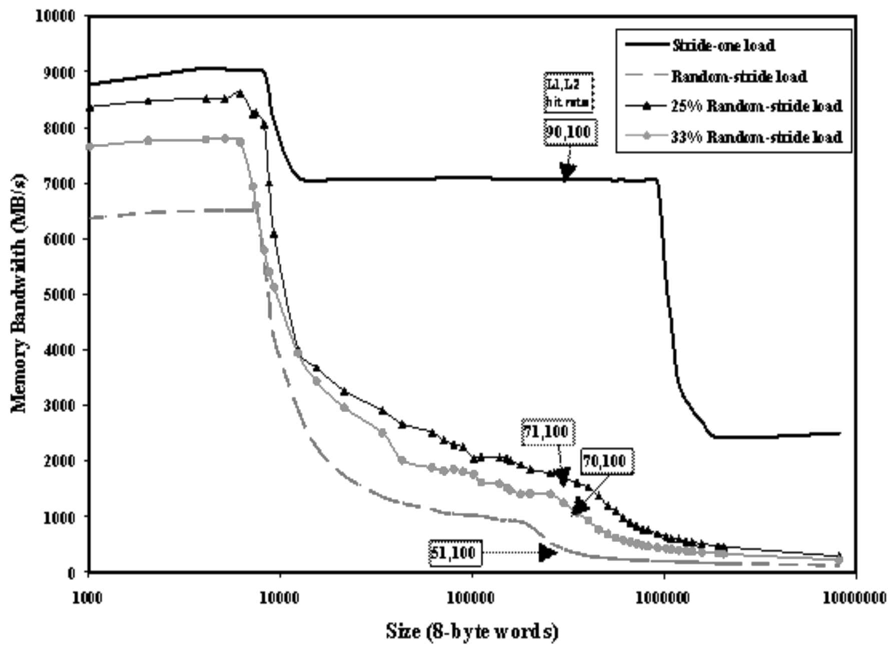
GC64 Overview

- Original project
  - Ground up effort to build an architecture and ISA for data intensive computing
- Latest Research
  - Utilize the core ISA/machine architecture from RISC-V
  - Build extensions to core RISC-V architecture for data intensive computing
- Focus on what matters:

Programmable Data Intensive Computing Performance

GC64 Requirements

- Data intensive algorithms & applications
  - Sparse data structures
  - Sparse Matrix/SPMV
  - Graph computations
  - Network Theory
- Driving characteristics
  - Non-Unit Stride Memory Access
    - *Scatters/Gathers*
  - Memory Intensive
    - *Cache unfriendly*
    - *Non deterministic*



http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/

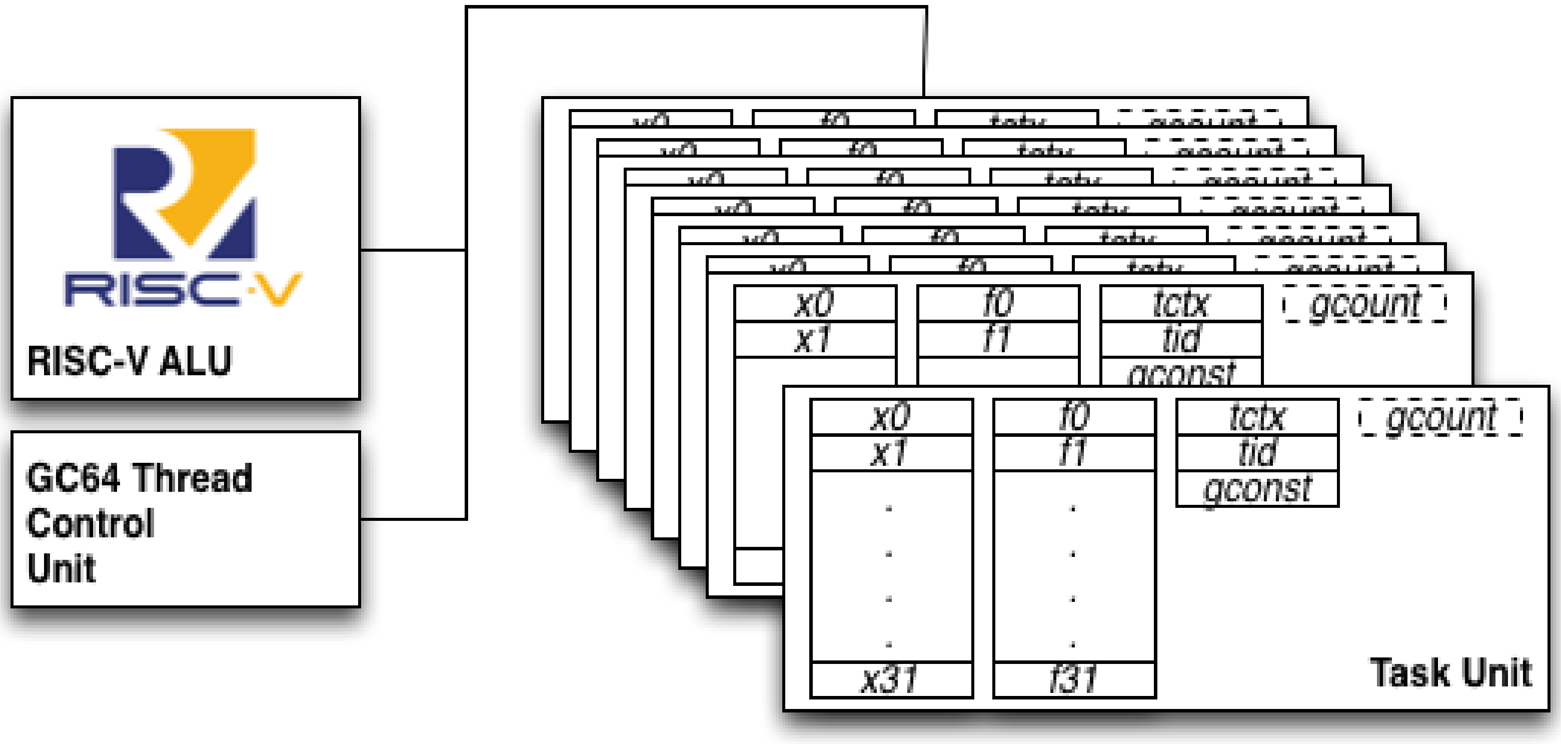
Architectural Goals

- Simple architectural components
  - Scaling the degree concurrency within an SoC is a manufacturing problem, not a design problem
- Simple ISA extensions conducive to compiler optimizations for concurrent applications
  - NOT writing the world's latest parallel compiler
- Provide low-level hardware support for mutable concurrency
  - A task lives in HARDWARE
- Provide hardware mechanisms to minimize hardware context switch latency
- Provide well-defined mechanisms on *when* and *how* context switch events occur
  - Provide a well-defined mechanism for USER SPACE code to induce a context switch

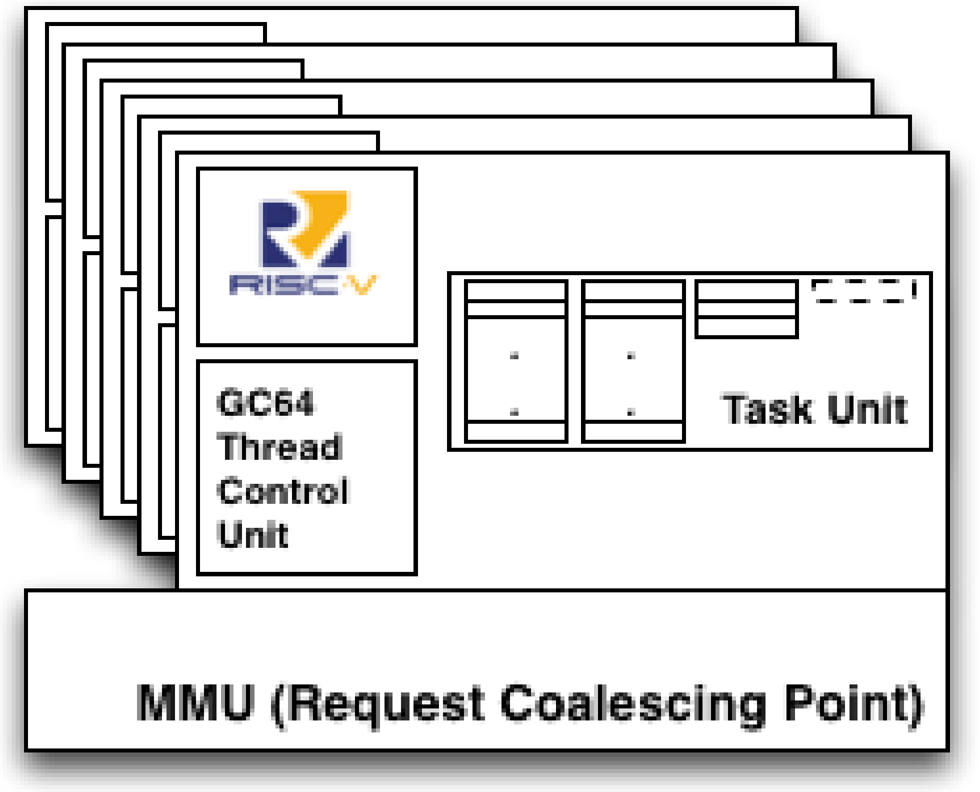
…and why its important for our ISA extension

GC64 ARCHITECTURE ORGANIZATION

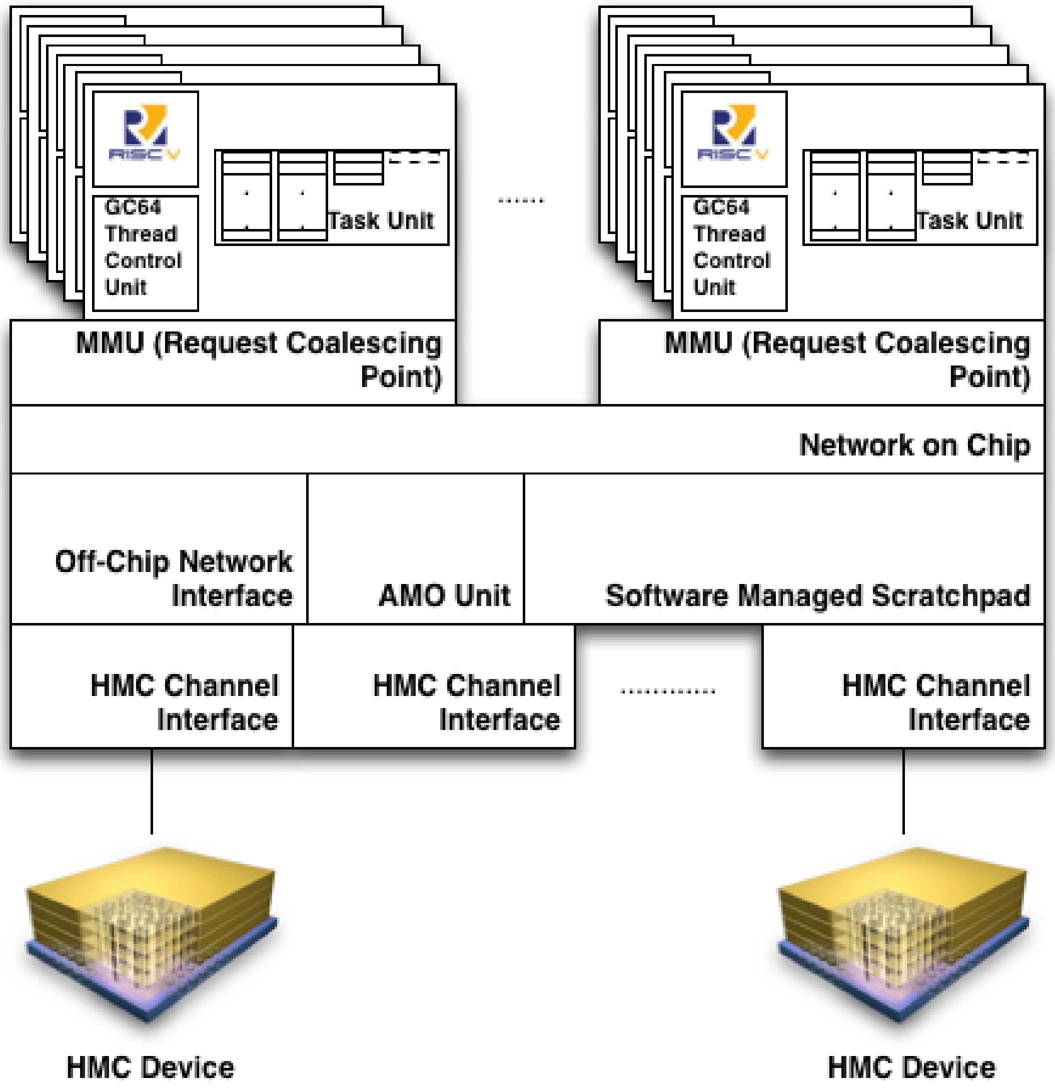
GC64 Architecture Organization



GC64 Architecture Organization cont.



GC64 SoC



RISC-V
GC64 Thread Control Unit
Task Unit
......
RISC-V
GC64 Thread Control Unit
Task Unit
MMU (Request Coalescing Point)
MMU (Request Coalescing Point)
Network on Chip
Off-Chip Network Interface
AMO Unit
Software Managed Scratchpad
HMC Channel Interface
HMC Channel Interface
....
HMC Channel Interface
HMC Device
HMC Device

User-Visible Registers

- TCTX
  - 64bit register that holds the address of the current task context
- TID
  - 64bit register that holds the current task ID
- TQ
  - 64bit register that holds the address of the task queue
- TE
  - Task exception register: exceptions specific to task operations
- GCONST
  - Constant register defining the locality of the given task unit
- GARCH
  - Architecture description register

Mnemonic	Bits	Size [Bits]	Description
TU	[7:0]	8	Task Units ID [hardware]
TP	[15:8]	8	Task Processors ID
TG	[23:16]	8	Task Group ID
SID	[31:24]	8	Socket ID
NID	[47:32]	16	Node ID
PID	[63:48]	16	Partition ID

Mnemonic	Bits	Size [Bits]	Description
NTU	[7:0]	8	Number of task units per processor
NTP	[15:8]	8	Number of task processors per group
NTG	[23:16]	8	Number of task groups per socket
NS	[31:24]	8	Number of sockets per node
NN	[47:32]	16	Number of nodes per partition
NP	[63:48]	16	Number of partitions

Supervisor Registers

- GKEY
  - The “gkey” register contains a 64-bit key loaded by the kernel
  - The key determines whether a task may spawn and execute work on neighboring task processors
  - Can only be written from privileged code
  - This provides a very rudimentary protection mechanism in order to prevent:
    - *Task resource starvation from other process spaces*
    - *Memory bandwidth utilization from outside process address spaces*
    - *Well defined mechanism to constrain the bounds of highly concurrent applications*

Currently being revised based upon the latest supervisor specification

Machine State Registers

- GCOUNT
  - Tracks the current state of a task's context pressure
  - Every instruction increments this value
  - Once the value reaches the overflow threshold, a context switch event is injected
  - Cannot be read or written directly from any instructions
  - One instruction, *ctxsw*, implicitly forces an overflow

Current GC64 RISC-V specification

GC64 RISC-V EXTENSION

RISC-V ISA Requirements

- What are we using from the core RISC-V architecture spec?
- ISA:
  - RV64I: 64bit integer arithmetic and addressing support
  - M-Extension: 64-bit integer multiplication and division
  - A-Extension: Atomic instructions
- Optional Support:
  - F-Extension: Single-precision floating point arithmetic support and storage
  - D-Extension: Double-precision floating point arithmetic support and storage
  - RC128I: Extended (scalable) 128-bit addressing support
    - *Physical addressing is architected to accept 128-bit extension*

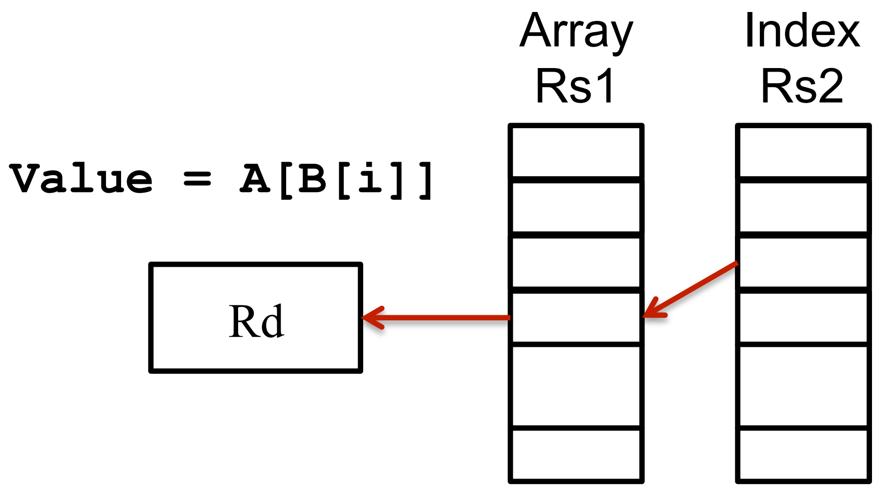
GC64 Instruction Extensions

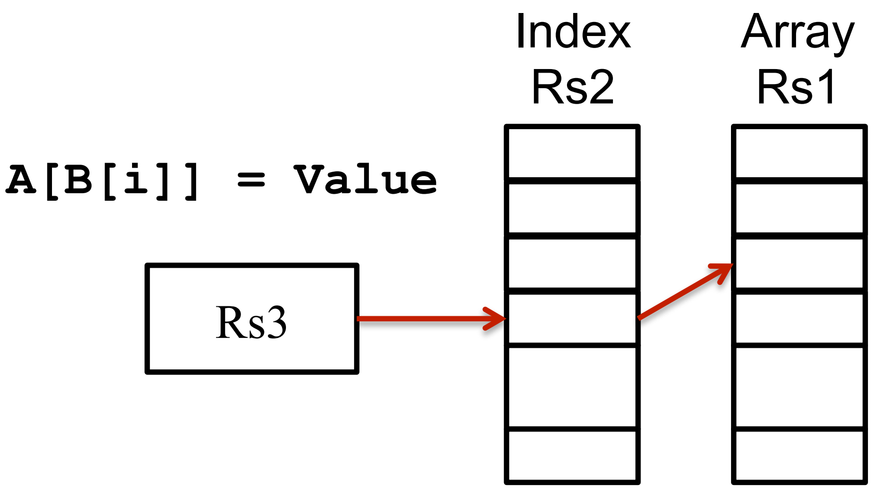
- Integer load/store instructions
- Single Precision load/store instructions
- Double Precision load/store instructions
- Concurrency instructions
- Task control instructions
- Environment instructions
- Supervisor instructions
- Extended addressing instructions

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type

GC64 Integer Load/Store

- Gathers (indexed load)
  - LBGTHR Rd, Rs1, Rs2
  - Rd = Rs1[Rs2]
  - Effective Base Address = addr(rs1 + (rs2 * size_in_bytes))
- Scatters (indexed store)
  - SBSCATR Rs1, Rs2, Rs3
  - Rs2[Rs3] = Rs1
  - Effective Base Address = addr(rs1 + (rs2 * size_in_bytes))





GC64 Concurrency Instructions

- IWAIT Rd, Rs1, Rs2
  - Pend the execution of the next instruction until the register hazard on the register index at Rd has been cleared
  - The pend is upheld while:
    - *rs2 < rs1*
- CTXSW
  - Set the *gcount* register to an overflow state, thus forcing the current task to context switch

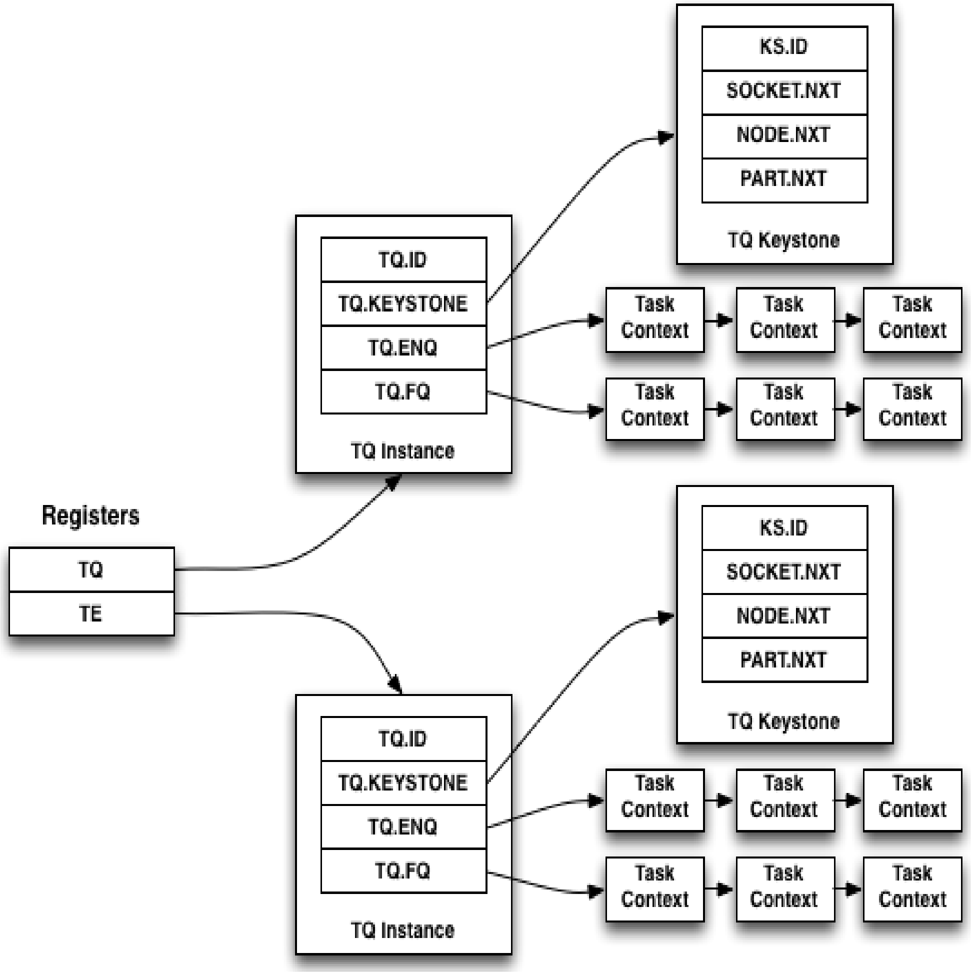
```
Data: rd is the target integer register, rs1 is the max unsigned value to
      be, rs2 is the unsigned start value
Result: The encountering task unit pends until rs2 is greater than or
        equal to rs1
while rs2 less than or equal to rs2 do
  | rs2++;
end
clear iwait state
```

GC64 Task Control Instructions

- SPAWN Rd, Rs1
  - Spawn a new task using the context at Rs1
- JOIN Rd, Rs1
  - Join a task using the task context at Rs1
- GETTASK RD, TCTX
  - Retrieve the task context value for the encountering task unit
  - "Where do I get my new tasks from?"
- SETTASK TCTX, RD
  - Set the task context value for the encountering task unit
  - "This is where I get my new tasks from!"
- GETTID RD, GTID
  - "What is my {task,thread,etc} id?
- Modifying Task Queue Values
  - GETTQ RD, TQ
  - SETTQ TQ, RS1
  - GETTE RD, TE
  - SETTE TE, RS1

GC64 Task Queue Structure

- Task control instructions manipulate a well-defined task queuing construct
  - Instructions are implemented via small RISC-V cores and embedded microcode
- Permits deterministic placement of task/ thread work via a low-level runtime library



Next steps in GC64 development

FUTURE DEVELOPMENT

Future Development

- The core architectural specification is documented
  - See http://discl.cs.ttu.edu/gitlab/gc64/gc64-doc
- The next steps in hardware design/implementation:
  1. Complete the design and implementation of the memory coalescing unit!
  2. Complete the design of the scratchpad memory layout
  3. Begin design and path-finding efforts on the scalable network architecture
  4. Demonstrate the initial architectural spec on an FPGA!
- The next steps in software design/implementation:
  1. Harden the current simulator implementation(s)
  2. Rework portions of the the software scratchpad runtime (named address spaces)
  3. Complete the low-level runtime library

Questions/Comments?

John Leidel

john.leidel@ttu.edu

http://discl.cs.ttu.edu/gitlab/groups/gc64

TEXAS TECH UNIVERSITY™