GoblinCore-64: A Scalable, Open Architecture for Data Intensive High Performance Computing

John D. Leidel, Yong Chen

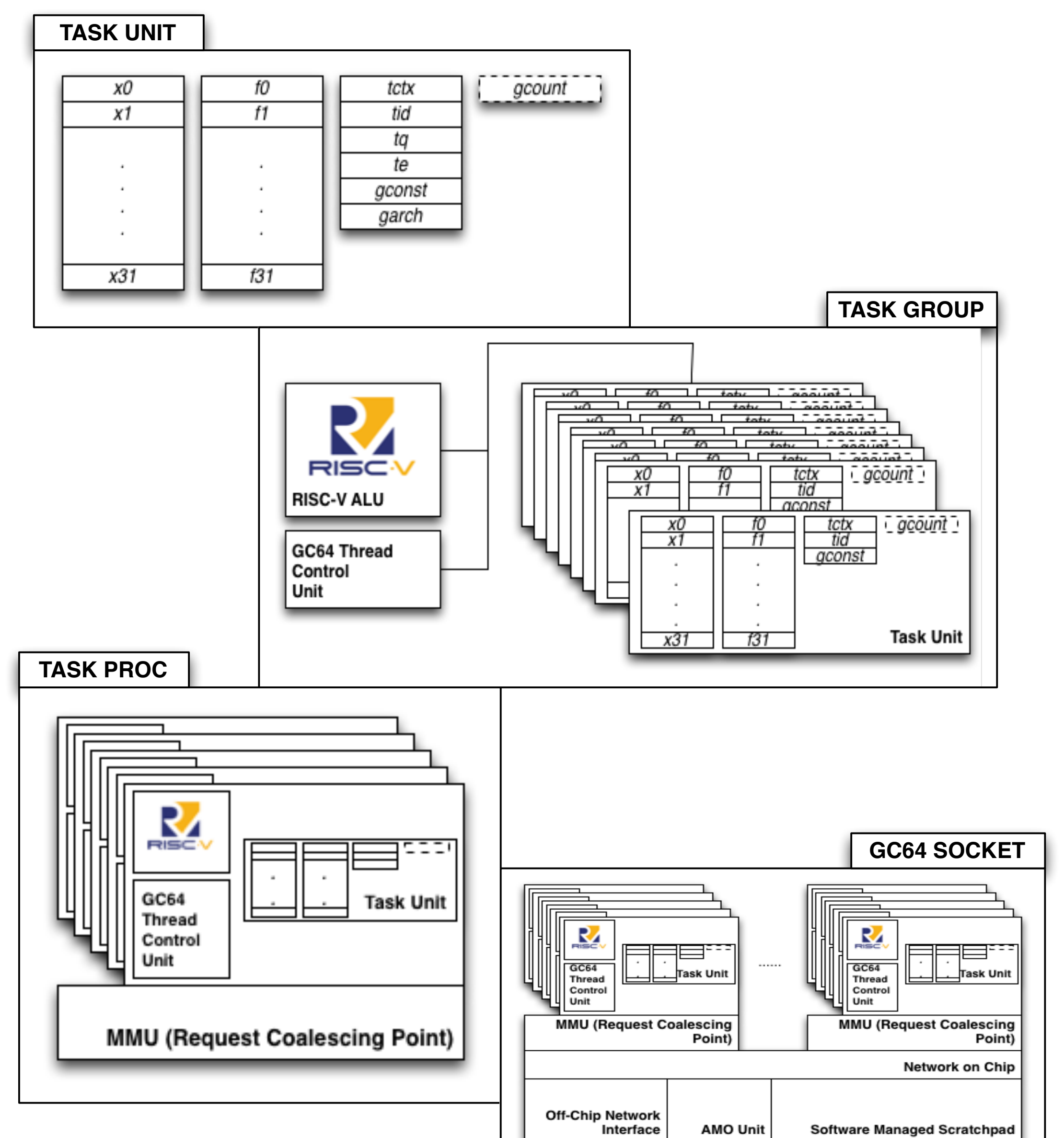
Abstract

The current class of mainstream microprocessor architectures rely upon multi-level data caches and relatively low degrees of concurrency to solve a wide range of applications and algorithmic constructs. The mainstream architectures are well suited to efficiently executing applications that are generally considered to be cache friendly. These may include applications that operate on dense, linear data structures or applications that make heavy reuse of data in cache.

However, applications that are generally considered to be data intensive in nature may access memory with irregular memory request patterns or access such large data structures that they cannot reside entirely in an on-chip data cache. The goal of GC64 is to provide a scalable, flexible and open architecture for efficiently executing data intensive computing applications and algorithms.

The GC64 infrastructure is built upon a hierarchical set of hardware modules designed to support scalable concurrency with explicit support for latency hiding. The infrastructure's memory hierarchy is constructed using software-managed scratchpad memories for local, application-managed memory requests and Hybrid Memory Cube devices for main memory storage. The instruction set is based upon the RISC-V instruction set specification with additional extensions to support scatter/gather memory requests, task concurrency and task management. The result is a simple, effective, and scalable architecture that can be easily adapted to efficiently execute data intensive applications using commodity programming models such as OpenMP, MPI and MapReduce.

GC64 Architecture Hierarchy



Introduction

This work introduces the ***GoblinCore-64*** architecture and infrastructure. GoblinCore-64, herein also referred to as ***GC64***, combines a hierarchical system infrastructure and memory infrastructure to provide a scalable architecture designed to efficiently support data intensive computing applications. The system infrastructure hierarchy provides efficient concurrency mechanisms to hide the latency of accessing memory in irregular patterns. The memory infrastructure combines software-managed scratchpad memories on chip to high bandwidth, ***Hybrid Memory Cube*** [HMC] devices in order to provide significant bandwidth to concurrent applications. The core machine model and instruction set is based upon the RISC-V instruction set architecture with the addition of an extended set of instructions designed to support task concurrency, task management and scatter/gather memory operations. Finally, every attempt has been made to build the hardware modules, infrastructure and software tools under BSD-like licenses such that both academic and commercial organizations may make use of GoblinCore-64.

Construct a truly scalable and programmable data intensive computing architecture

RISC-V Requirements

Requirement	Extension
RISC-V Required ISA Extensions	RV64I, M, A
RISC-V Optional ISA Extensions	RV128I, F, D, Q

GC64 ISA Extension

Instruction(s)	Description
Scatter/Gather Instructions	
lbgthr, lhgthr, lwgthr, ldgthr, lbugthr, lhugthr, ldugthr, lqugthr	Integer Gather Instructions
sbscatr, shscatr, swscatr, sdscatr, sqscatr	Integer Scatter Instructions
flwgthr, fldgthr	Floating Point Gather Instructions
fswscatr, fsdscatr	Floating Pointer Scatter Instructions
Concurrency Instructions	
iwait	Instruction Wait (Hazard)
ctxsw	Forced Context Switch
Task Control Instructions	
spawn, join	Task Spawns and Joins
gettask, settask	Get/Set the Task Context
gettid	Get the Task ID
gettq, settq	Get/Set the Task Queue Address
Supervisor Instructions	
sgetkey	Get the Supervisor Task Key
ssetkey	Set the Supervisor Task Key

GC64 Scalability

Hardware Unit	Scalability
Task Units Per Task Processor	256
Task Processors Per Task Group	256
Task Groups Per Task Socket	256
Sockets Per Node	256
Nodes Per Partition	65,536
Partitions Per System	65,536

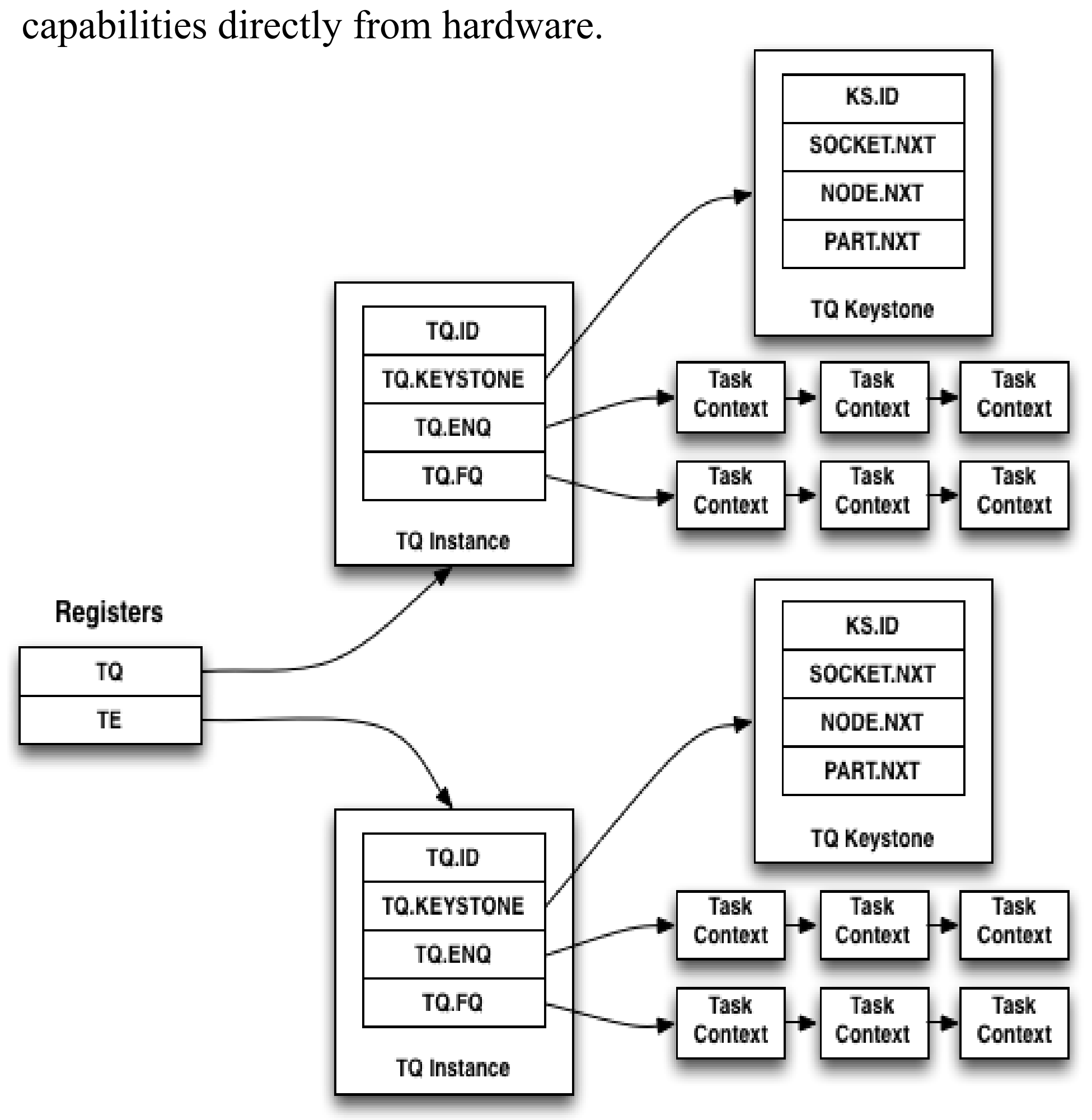
18,446,744,073,709,551,616 Maximum Concurrent Tasks

RISC-V Register Extensions

Mnemonic	Access	Description
TCTX	User, Supervisor	Task Context Address
TID	User, Supervisor	Task/Thread ID
TQ	User, Supervisor	Task Queue Address
TE	User, Supervisor	Task Exception Queue (debugging)
GCONST	User, Supervisor	Physical Locality Constant
GKEY	Supervisor	Security Key
GCOUNT	Arithmetic Machine State	Context Switch Pressure

GC64 Task Queuing

Tasks are managed via the ***Task Control*** instruction extensions. The base address found in the task queue (*TQ*) register contains the address of the task queuing structures. Task queues are linked across hardware partitions using ***Task Keystones***. This permits us to express task locality and provide work stealing capabilities directly from hardware.



OpenMP Task Yield

```
#include <omp.h>


void solver_function( void );
void halo_exchange( void );

void driver( omp_lock_t *lock,
             int n ) {

  int i;

  for( i = 0; i<n; i++ ){
    #pragma omp task
    {
      solver_function();

      while(!omp_test_lock(lock)){
        #pragma omp taskyield
      }

      halo_exchange();
      omp_unset_lock( lock );
    }
  }
}
```

RISC-V

```
driver . _omp_fn .0:
  add sp,sp,−16
  sd s0,0(sp)
  sd ra,8(sp)
  ld s0,0(a0)
  call solver_function
  j .L3
.L2:
  call GOMP_taskyield
.L3:
  move a0,s0
  call omp_test_lock
  beq a0,zero,.L2
  call   halo_exchange
```

GC64

```
driver . _omp_fn .0:
  add sp,sp,−16
  sd s0,0(sp)
  sd ra,8(sp)
  ld s0,0(a0)
  call solver_function
  j .L3
.L2:
  ctxsw
.L3:
  move a0,s0
  call omp_test_lock
  beq a0,zero,.L2
  call   halo_exchange
```

Task yield resolves to a runtime call!

GC64 uses a single instruction

Future Exploration

Our current research and exploration focuses on the development of an individual socket. While we have the necessary micro-architectural optimizations in place for a scalable system, several items need to be addressed for future scalability of GC64 systems. These include:

- ***Scalable Memory Interconnect:*** The scalable system version of GC64 relies upon a high performance interconnect well suited to injecting small message traffic. We shall explore adapting the HMC memory protocol as the basis for a scalable system interconnect.
- ***Global Synchronization/Barrier Coalescing:*** Providing the ability to coalesce raw local and global memory requests has inherent advantages. However, coupling this with additional memory logic to uniquely identify global synchronization primitives may prove to very useful for nondeterministic applications.

We also seek to port additional programming models to our architecture. This requires additional tool chain and library support from external sources. Additional work may be required to optimize the runtime infrastructure for each programming model. These include:

Cray Chapel	*MapReduce*
Unified Parallel C	*Apache Spark*

References

[1] GoblinCore-64. http://gc64.org/

[2] Data Intensive Scalable Computing Laboratory. http://discl.cs.ttu.edu/

[3] RISC-V ISA Specification. http://riscv.org/specifications/

[4] Hybrid Memory Cube Specification 2.1. Technical Report, Hybrid Memory Cube Consortium, Accessed March 2016.

Contacts:

John D. Leidel [john.leidel@ttu.edu]

Yong Chen [yong.chen@ttu.edu]