Coreographer

Micro-task Scheduled Multi-core RISC-V Architecture

Revised Scope Document

E. Mihir Divyansh
EE23BTECH11017

Executive Summary

This project implements a **heterogeneous multi-core RISC-V architecture** inspired by GPU streaming multiprocessors, featuring clustered cores with local register sharing and a hardware-based micro-task scheduler. The design targets moderately parallel workloads where task-level parallelism can be exploited without the overhead of a full OS.

Architecture Overview

1. **Neighbor-based register sharing** replaces global shared register file
   - Each core has its own private 32-register file (RV32I standard)
   - Cores can access registers from neighboring cores via extended instructions
   - Neighbor topology is configurable (And is optionally a performance affecting parameter)
   - **Exploration aspect: (Optional)** Different topologies can be implemented and compared
2. **Parallel hardware scheduler** as dedicated dispatch unit
   - Maintains instruction cache and ready queue for worker cores
   - Broadcasts tasks on shared dispatch bus
   - Cores pull tasks when available
3. **Micro-task programming model**
   - Input programs structured as self-contained micro-tasks. (To be defined)
   - Tasks declare dependencies and resource requirements in headers

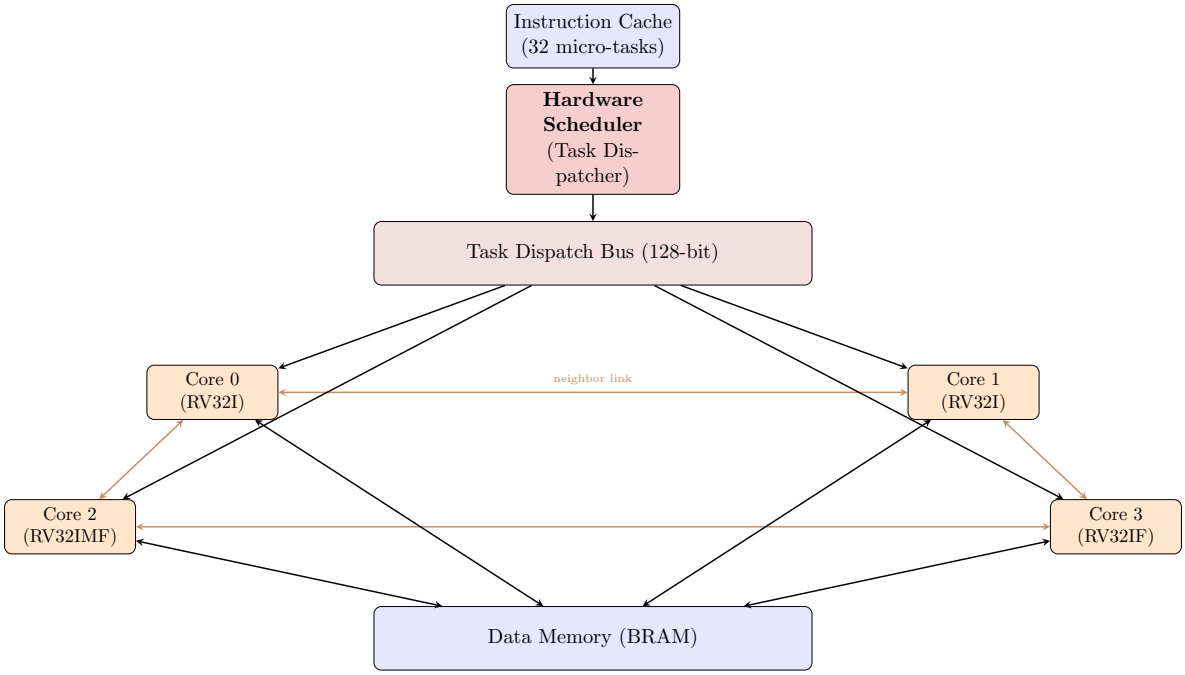

Figure 1: Example Block Diagram of System

System Block Diagram

One topology will be chosen for initial implementation, with potential to compare alternatives in stretch goals.

Micro-Task Format

Each micro-task consists of a **header** (metadata) and a **body** (instructions).

Task Header Format (32 bits):

```
[31:24] Task ID (8 bits)
[23:16] Dependency vector (8 bits) - IDs of prerequisite tasks
[15:12] Capability flags (4 bits):
        bit 3: Requires multiply (→ must go to Cluster 1)
        bit 2: Requires division
        bit 1: Memory-intensive (hint)
        bit 0: Reserved
[11:8]  Cluster preference (4 bits): 0=any, 1=Cluster0, 2=Cluster1
[7:0]   Instruction count (8 bits): number of instructions in body
```

Task Body:

- Sequence of 4-16 standard RV32I(M) instructions

- Last instruction must be task completion marker (custom: `TASK_DONE`)
- No branches outside task boundary (tasks are atomic)
- Example:

```
.task 42 # Task ID
.depends 39, 40 # Wait for tasks 39 and 40
.needs_mul # Requires Cluster 1
.instructions
lw r1, 0(r10) # Load A[i]
lw r2, 0(r11) # Load B[i]
mul r3, r1, r2 # Multiply
sw r3, 0(r12) # Store C[i]
TASK_DONE # Signal completion
.end_task
```

Benchmark Selection

Selection Criteria:

- Exhibit moderate task-level parallelism (not embarrassingly parallel)
- Fit in on-chip memory (32 KB data limit) (Optionally stream from host)
- Simple enough to manually decompose into micro-tasks
- Demonstrate heterogeneous core utilization

Benchmark 1: Vector Dot Product

- **Size:** Two 64-element vectors (512 bytes total)
- **Why:** Simple, verifiable, demonstrates basic dispatch.
- **Task decomposition:** Each task computes one element: `result[i] = A[i] * B[i]`

Benchmark 2: Small Dense Matrix Multiply

- **Size:** 8×8 or 16×16 matrices (depending on memory constraints)
- **Task decomposition:** Each task computes one output element via dot product

Benchmark 3: Streaming Accumulator

- **Size:** 256-element stream, 8-tap accumulator window
- **Parallelism:** Pipeline with overlapping windows
- **Why:** Tests scheduler's ability to maintain steady-state throughput

Evaluation Metrics

Performance Metrics

- **Throughput**
- **Speedup**
- **Core utilization:** % of cycles each core is executing instructions
- **Scheduler efficiency:** Dispatch stalls / total dispatch attempts
- **CPI per core**
- **Neighbor access metrics:**
  - Neighbor register read/write counts per core
  - Average latency per neighbor access
  - Neighbor access efficiency (successful / attempted)

Resource Utilization

- LUT count, BRAM, DSP usage
- Maximum clock frequency
- Power consumption estimate

Scope of Future Work

- Topology comparison: Topology vs performance on different workloads.
- Vary core count (2, 4, 8 cores) to study neighbor network scaling
- Compare homogeneous (all RV32I) vs heterogeneous (RV32I + RV32IM) configurations
- Analyze neighbor access latency sensitivity (1-cycle vs 2-cycle access times)

Deliverables

1. **RTL codebase:** Synthesizable Verilog/VHDL for all system components
2. **Benchmark Results:** At least 2 working micro-task programs with verification datasets
3. **FPGA demonstration:** Live demo showing parallel execution and performance counters