Download worksheet 8

Please mute yourself when not asking questions

CS 152: Discussion Section 9

Vector, GPU, and Lab 4 Overview

Albert Ou, Yue Dai
04/03/2020

Administrivia

- Lab 3 is due on Mon, April 6
- PS4 is extended to Mon, April 6

- Midterm 2 is on Wed, April 15 (format TBD)
- Lab 4 is due on Mon, April 20

Agenda

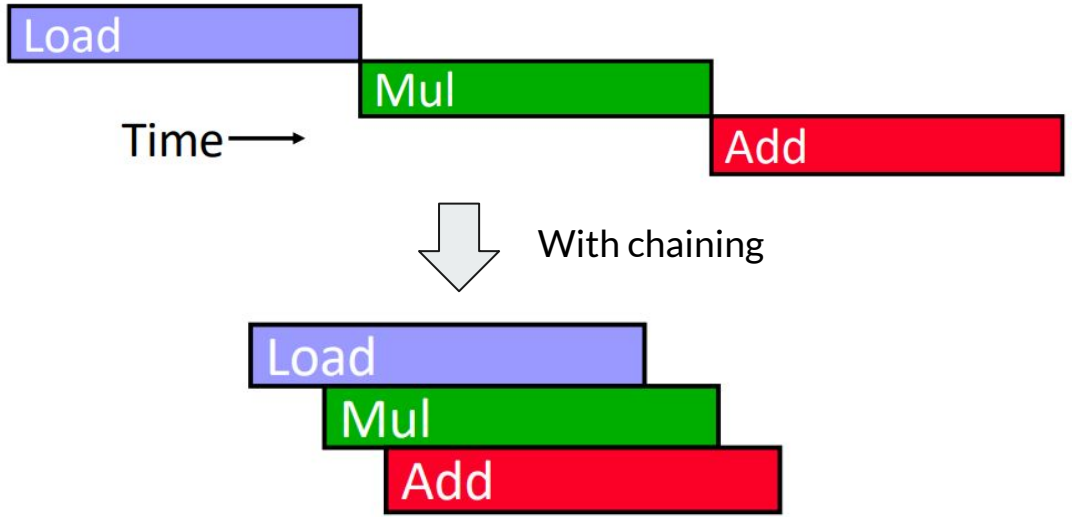
- Vector architecture
- Packed-SIMD
- GPU
- Lab 4 overview

Vector Architecture - Vectorization

- Vector stripmining: break loops into pieces that fit in vector registers
- Vector masks: help vectorize code with conditional execution
- Vector reduction: improve vectorization for loop-carried dependence
- Vector scatter/gather: vectorize loop with indirect memory accesses; need to pay attention to loop-carried dependence as well

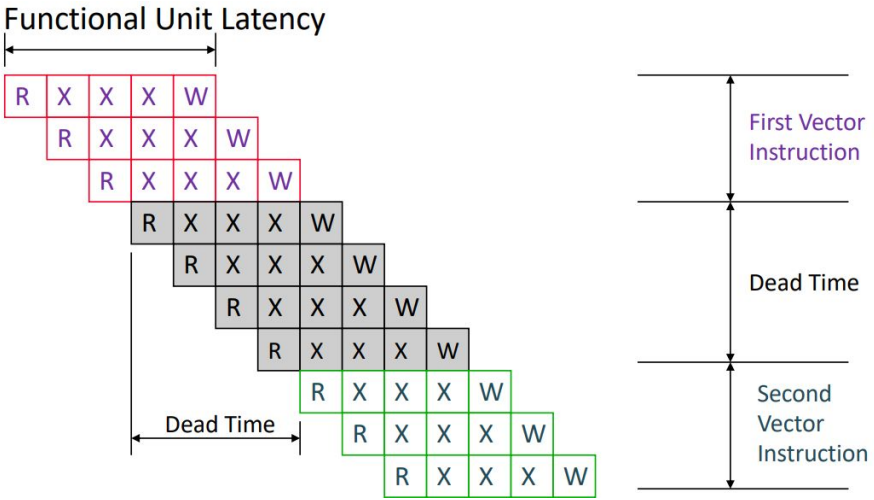
Vector Architecture

- Vector chaining: start dependent instruction as soon as first result appears



Vector Architecture

- Dead time: time before another vector instruction can start down pipeline



Packed SIMD

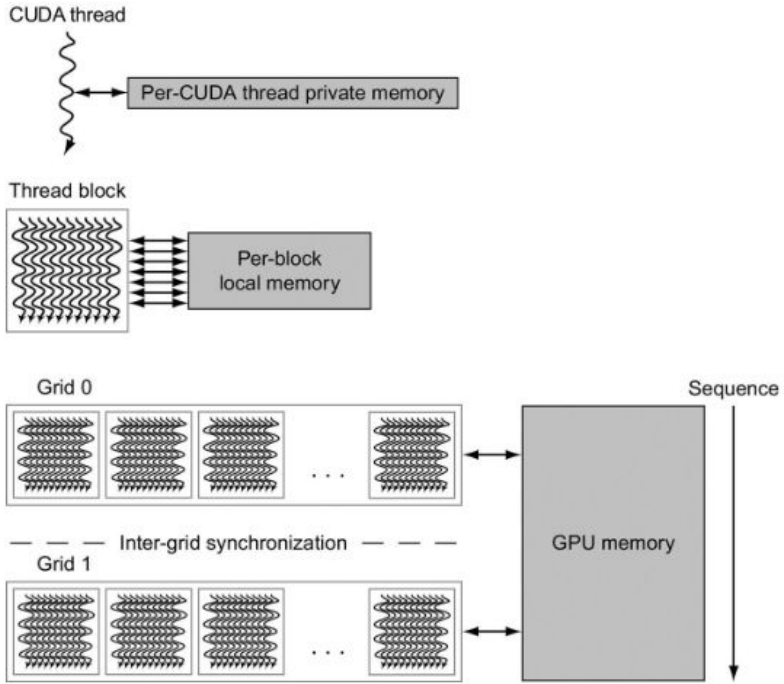
- Very short vectors added to existing scalar ISAs for microprocessors
- Q: Compare and contrast packed-SIMD and vector architectures

GPU

- SIMT: individual scalar instruction streams for each CUDA thread are grouped together for SIMD execution on hardware
- All memory operations are scatter/gather; rely on coalescing to detect access patterns
- Branch divergence managed in hardware

GPU - Memory

- Q: What are the ways for vector architectures and GPU to hide memory latency?
- A:
  - Vector - deeply pipelined memory access, memory-level parallelism with vector accesses patterns (CS252: access/execute decoupling)
  - GPU - Multithreading



Conditional SAXPY: Vectorizable Loop

```
// y = a*x + y

void csaxpy(size_t n, bool *cond, float a, float *x, float *y) {
    for (size_t i = 0; i < n; i++)
        if (cond[i])
            y[i] = (a * x[i]) + y[i];
}
```

Conditional SAXPY: SPMD kernel

```
csaxpy_spmd<<<((n-1)/32+1)*32>>>;   // 1-D thread launch

void csaxpy_spmd(size_t n, bool *cond, float a, float *x, float *y) {
    if (tid.x < n)
        if (cond[tid.x])
            y[tid.x] = (a * x[tid.x]) + y[tid.x];
}
```

CSAXPY: Vector Assembly

a0	size_t n
a1	bool *cond
fa0	float a
a2	float *x
a3	float *y

```
csaxpy_rvv:
stripmine_loop:
    vsetvli t0, a0, e8, m2       # 8-bit elements
    vle.v v8, (a1)               # load cond[i]
    vmsne.vi v0, v8, 0           # set mask
    vsetvli zero, zero, e32, m8  # 32-bit elements
    vle.v v8, (a2)               # load x[i]
    vle.v v16, (a3)              # load y[i]
    vfmacc.vf v16, fa0, v8, v0.t # a*x[i] + y[i]
    vse.v v16, (a3)              # store y[i]
    sub a0, a0, t0               # decrement n
    add a1, a1, t0               # bump cond
    slli t0, t0, 2
    add a2, a2, t0               # bump x
    add a3, a3, t0               # bump y
    bnez a0, stripmine_loop
    ret
```

CSAXPY: SIMT Assembly

a0	size_t n
a1	bool *cond
fa0	float a
a2	float *x
a3	float *y

```
csaxpy_simt:
    mv      t0, tid      # get thread ID
    bgeu    t0, a0, skip
    add     t1, a1, t0   # compute cond + i
    lbu     t1, (t1)     # load cond[i]
    beqz    t1, skip
    slli    t0, t0, 2
    add     a2, a2, t0   # compute x + i
    add     a3, a3, t0   # compute y + i
    flw     ft0, (a2)    # load x[i]
    flw     ft1, (a3)    # load y[i]
    fmadd.s ft0, fa0, ft0, ft1
    fsw     ft0, (a3)    # store y[i]
skip:
    stop
```

Lab 4 Overview

Write/optimize RISC-V vector code in assembly and run on ISA simulator

- **cmplxmult**: Complex multiply
- **dgemv**: Double-precision generalized matrix-vector multiply
- **dgemm**: Double-precision generalized matrix-matrix multiply
- **imax**: Index of maximum

- **spmv**: Sparse matrix-vector multiply (open-ended)
- **rsort**: Radix sort (open-ended)