Lecture 13: Masked Vectors

COSC 273: Parallel and Distributed Computing

Spring 2023

Announcements

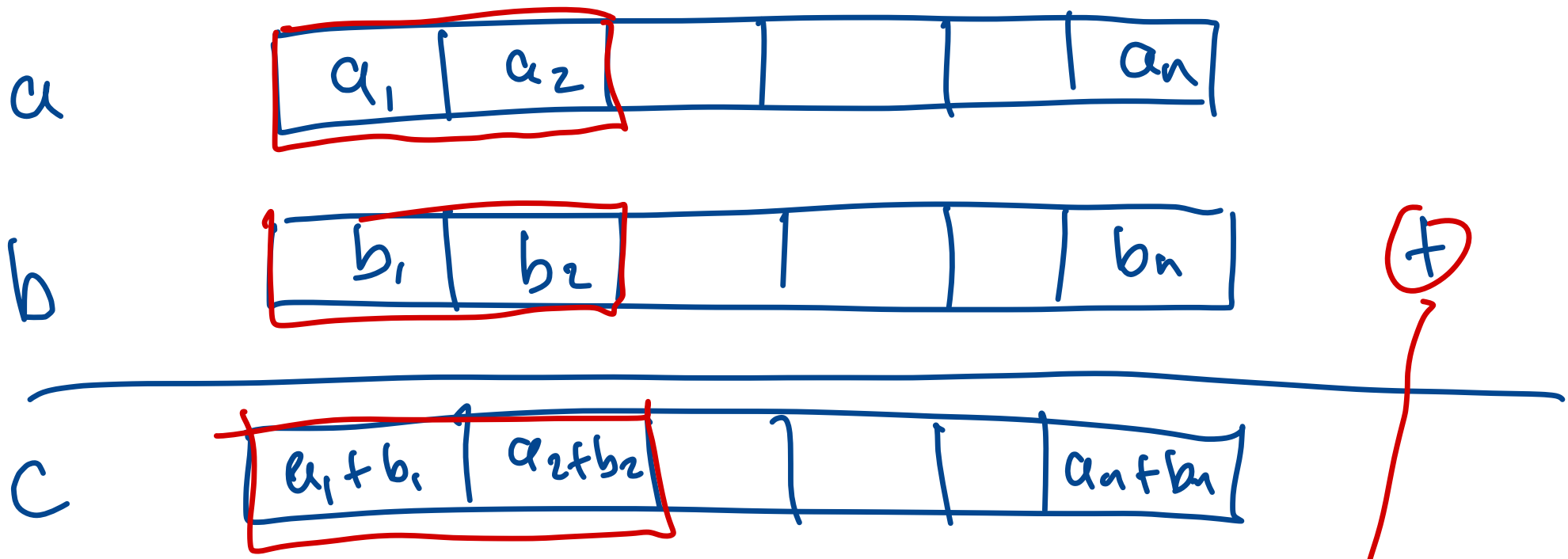
1. Homework 02: Now Due **Next** Friday (03/10)
2. Lab 03 will be due after spring break

Outline

1. Vectors and Masking
2. Benchmarking Notes
3. The Mandelbrot Set

Vector Operations in Pictures

For each `i`, set `c[i] = a[i] + b[i]`



SIMD parallelism

single instruction, multiple data

Vector module in Java

Vector Operations in Code

"width" of vector of Floats

```
int step = SPECIES.length();
int bound = SPECIES.loopBound(a.length);
int i = 0;
for (; i < bound; i += step) {
    var va = FloatVector.fromArray(SPECIES, a, i);
    var vb = FloatVector.fromArray(SPECIES, b, i);
    var vc = va.add(vb);
    vc.intoArray(c, i);
}
```

just as fast a[i]+ b[i]

store step consec. values from a, b to a vector

An Issue?

Question. What if we don't want to apply an operation to *all* entries in a `Vector`?

E.g., conditional assignment:

```
for (int i = 0; i < a.length; i++) {
    if (b[i] > 0) {
        c[i] = a[i] + b[i];
    } else {
        c[i] = a[i];
    }
}
```

Different instructions?

A Vector Solution

To apply an operation (say, `add`) only to *some* lanes:

1. store a vector of Boolean "flags"
   - this vector is a **vector mask**
2. only apply the operation for the lanes where the mask is `true`
   - hardware supports vector masking!
   - avoids conditional statements (which tend to slow execution)

Masking Example, In Pictures

Set `c[i] = a[i] + b[i]` if `b[i] > 0` and `c[i] = a[i]` otherwise

a : [4, 1, 3, 2]

b : [1, -2, 3, -4]

mask : [1, 0, 1, 0]

"b[i] < 0"

don't add

don't add

a.add(b, mask)

c[i] = a[i] + mask[i] * b[i]

want : c : [5, 1, 6, 2]

Masking in Java

- `VectorMask<Float>` datatype: think `Vector` of `Booleans`
- masked arithmetic operations:

```
public final FloatVector add(Vector<Float> v,
                             VectorMask<Float> m)
```

"Adds this vector to a second input vector, selecting lanes under the control of a mask. This is a masked lane-wise binary operation which applies the primitive addition operation (+) to each pair of corresponding lane values. For any lane unset in the mask, the primitive operation is suppressed and this vector retains the original value stored in that lane. This method is also equivalent to the expression `lanewise(ADD, v, m)`."

Creating and Using a Mask

A `VectorMask<Float>` that is `true` when `b[i] > 0`:

```
var va = FloatVector.fromArray(SPECIES, a, i);
var vb = FloatVector.fromArray(SPECIES, b, i);
var bMask = vb.compare(VectorOperators.GT, 0);
var vc = va.add(vb, bMask);
```

val in lane j is

a[j] if bBash is 0 in lane j

a[j] + b[j] — 1 —

Examle: Hamming Weights

Definition. Given an `int` a, the **Hamming weight** of a is the number of 1s in the binary representation of a.

a	binary	Hamming
0	0	0
1	1	1
2	10	1
3	11	2
7	111	3
15	1111	4
16	10000	1

a_2: 1011001 $a >> 1$ 0101100 ~~X~~

Examle: Hamming Weights

Definition. Given an `int a`, the **Hamming weight** of `a` is the number of 1s in the binary representation of `a`.

Question. How to compute Hamming weight of `int a`?

int a:

→ find largest power of 2 less than/eq to a, subtract from a, add 1 to count, repeat

Alt: if a % 2 == 1, add to count
a /= 2, repeat until a == 0

Hamming Weights via Bitwise &

Bitwise & operator, `a & b`:

1 & 1 = 1

else = 0

a:	0	1	1	0
b:	1	0	1	1
a&b:	0	0	1	0

How to determine if bit `i` is `0` or `1`?

	4	3	2	1
a	0	1	1	0
idx	0	0	1	0

i

& a & idx == 0 if ith bit of a is 0

and != 0 otherwise

Computing the Hamming Weight

Idea. For bits `i = 1...32`, check if bit `i` is 1

- if so, increment a count

idx = 1, 2, 4, 8, ..., 2^{32}

count starts @ 0

<u>if</u> a & idx ≠ 0, increment count

Computing the Hamming Weight

Idea. For bits `i = 1...32`, check if bit `i` is 1

- if so, increment a count

In code:

```
int val;
int idx = 1;
int weight = 0;
for (int j = 0; j < INT_LENGTH; j++) {
    if ((val & idx) != 0) {
        weight++;
    }
    idx = idx << 1;
}
// weight is the Hamming weight of val
```

Question

Want to compute Hamming weight of an array of `ints`...

```
for (int j = 0; j < INT_LENGTH; j++) {
    if ((val & idx) != 0) {
        weight++;
    }
    idx = idx << 1;
}
```

How could we vectorize this method?

Vectorization Idea

1. create vector `va` from array of values
2. create vector `vb` initialized to all `0` ← count for each lane
   - this will store Hamming weights
3. `int idx` has `1` in exactly one bit position
4. iterate over bits `idx`:
   - create mask that is `1` if `i`th bit of `va` lane is `1`
   - use mask to add `1` to corresponding lanes

Vectorized Code

```
var va = IntVector.fromArray(SPECIES, a, i);
var vb = IntVector.broadcast(SPECIES, 0);
int idx = 1;
for (int j = 0; j < INT_LENGTH; j++) {
    var bitMask = va.and(idx).eq(0).not();
    vb = vb.add(1, bitMask);
    idx = idx << 1;
}
vb.intoArray(b, i);
```

Create vectors

000 ... 01

32

va in lane k gets incremented if jth bit is 1

`HammingWeight.java` Demo

Benchmarking Notes

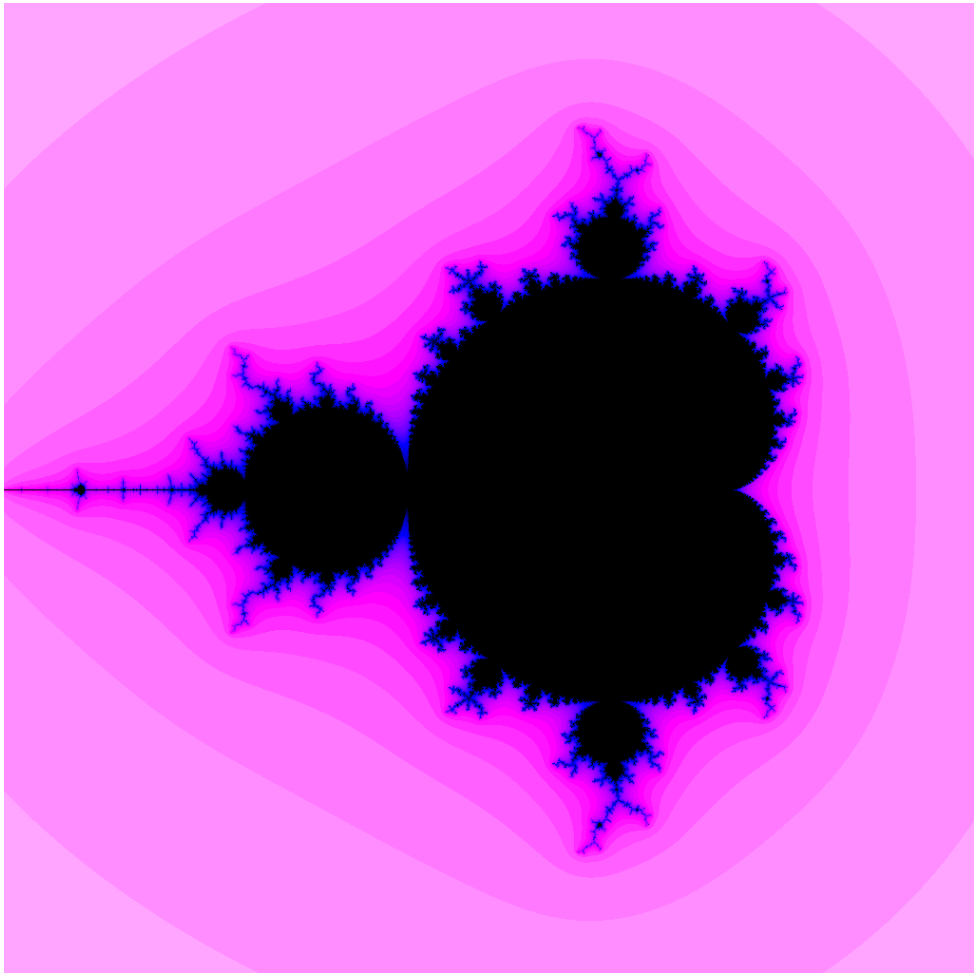
To give “accurate” measure of efficiency:

- test running time of method for **many** invocations
- run several invocations before starting timing
  - “warm up” primes hardware with correct instructions

Benchmarking Demo

Lab 03: Mandelbrot Set

Draw this picture as quickly as possible!



Next Week

1. Mandelbrot set definition
2. Thread pools and executers
    - handling threads more elegantly