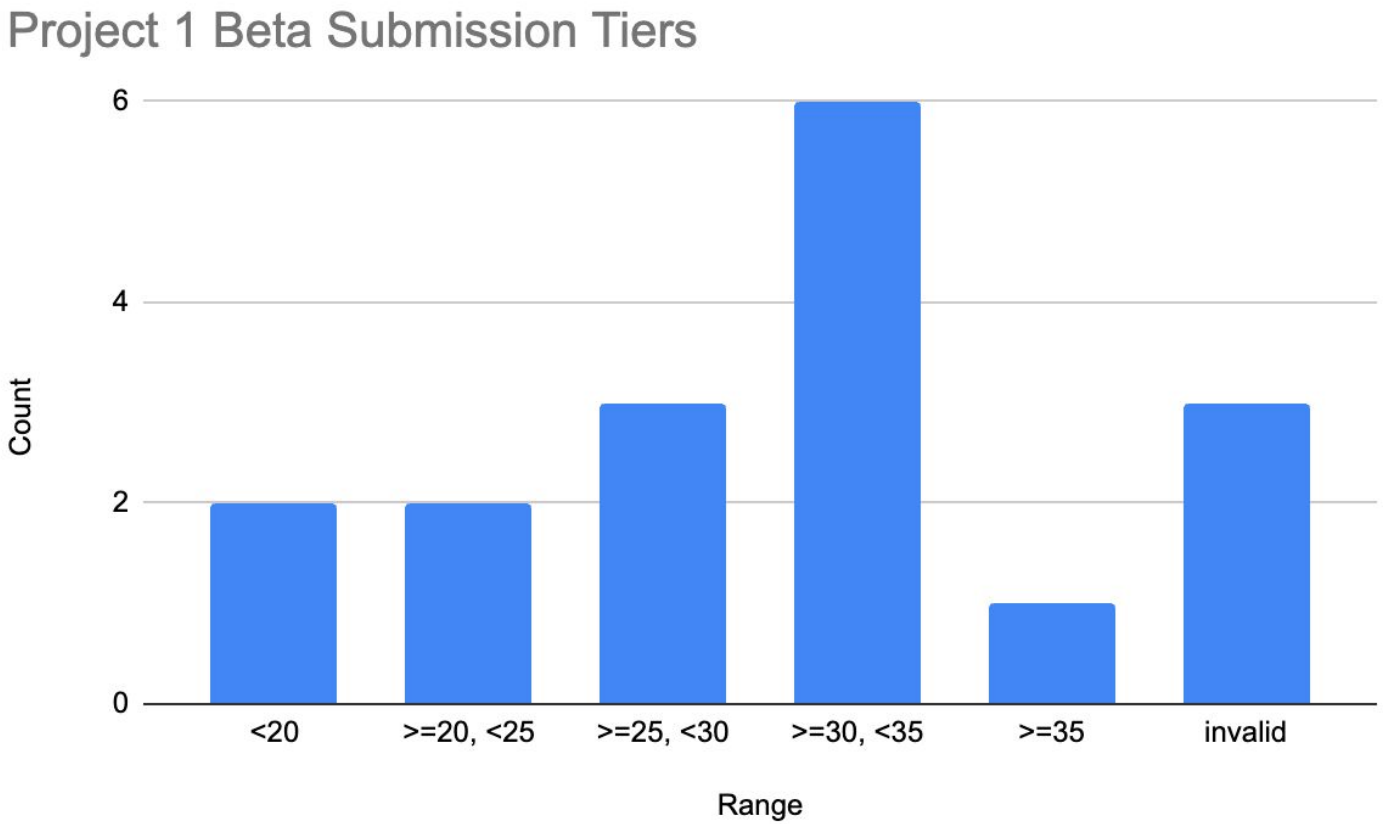
Software Performance Engineering

Recitation 1.5

Sophia Sun
Tuesday, Sep 30, 2025

Project 1 Beta Submission Tiers



Some due dates

- Homework 3 is due Sep 29 yesterday
- Homework 4 will release this Thursday

- Project 1 final is due October 9
- The final writeup:
  - make sure to include it in your GitHub repo.
  - your design, your current performance
  - what you've improved since the beta submission

- Reach tier 35 to get at least a B grade for final

VECTORIZATION

Basic Vector Instructions

- Vector Register Sizes: xmm: 128, ymm: 256, ~~zmm: 512 bits~~
- Packing: Fit many values in one register
  - xmm fits 2x uint64_t, 4x int32_t, 4x float, etc…
  - in general = (128/8) / sizeof(T)
- Packed Arithmetic Operations
  - add, subtract, multiply, divide, reciprocate, max, min, sqrt, reciprocal of sqrt
- Packed Comparison Operations
- Packed Logical Operations
- Packed Type Conversions
  - int to float etc

Overlapping Memory Regions

- When A[] and B[] overlap, memcpy(B, A, n) can be wrong
  - because the memcpy can overwrite data from the source, which it would need to use later
- Same thing can happen with vector operations
- To vectorize codes that deal with two arrays, need to know they don't overlap
- We can mark this using the restrict keyword
- `int* restrict A, int* restrict B`
  - The only way to obtain pointers from A's memory region is by offsetting from A
  - Thus A and B cannot overlap because there are no (non UB) offsets into each other

What can a compiler auto-vectorize?

- “Pure” (not interdependent) loop iterations
  - Dependence between loop iterations
- Reductions
  - Sum or product of contiguous memory: can detect the result variable is used for reduction
- Inductions
  - A[i] = i;
- If Statements
  - Certain if statements can be converted to branchless code
- Stride
  - A[i] += B[i * 4];
- When restrict is not known
  - Generates both vector and scalar code paths
  - Tests for overlap at runtime

Associativity of Reductions

- 1 + (2 + 3) == (1 + 2) + 3
- 0.1 + (0.2 + 0.3) != (0.1 + 0.2) + 0.3

- Floating point addition is not associative. This means order matters.
- Vector operations do not add the values in the same order as a scalar loop.
- -O3 has to produce code that behaves exactly the same as -O0, so the result might be different
- For float: Need the -ffast-math flag to tell the compiler we're ok with this.

USEFUL INTRINSICS

Alignment Hint

```
void *__builtin_assume_aligned(const void *arg,
size_t align);
```

Returns its first argument and allows the compiler to assume that the returned pointer is at least align bytes aligned.

// x is at least 16 byte aligned

```
void *x = __builtin_assume_aligned(arg, 16);
```

// (char *) x - 8 is 32 byte aligned

```
void *x = __builtin_assume_aligned(arg, 32, 8);
```

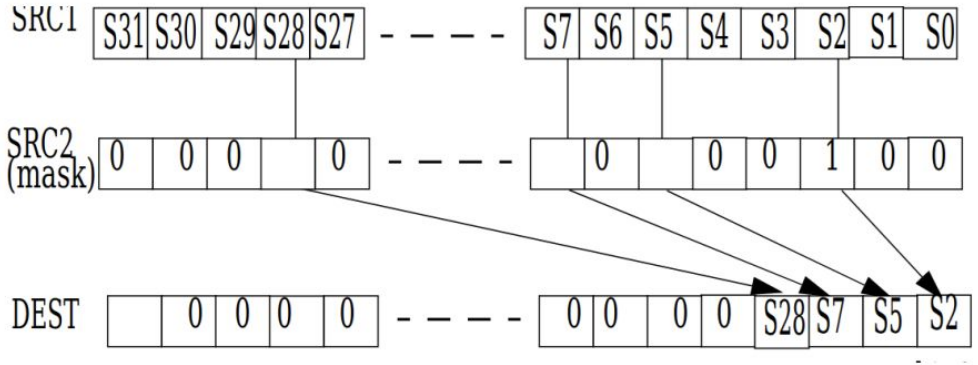
Other Bit Manipulations

```
uint64_t __builtin_rotateleft64(uint64_t x, bits_t amt);
uint64_t __builtin_rotateright64(uint64_t x, bits_t amt);

uint64_t __builtin_bitreverse64(uint64_t x);
uint64_t __builtin_bswap64(uint64_t x);

bits_t __builtin_popcountll(uint64_t x);
bits_t __builtin_clzll(uint64_t x);
bits_t __builtin_ctzll(uint64_t x);
```

Parallel Bit Extraction

```
#include <immintrin.h>
uint64_t _pext_u64(uint64_t s1, uint64_t mask);
```

Transfer either contiguous or non-contiguous bits in the first source operand to contiguous low order bit positions in the destination according to the mask values.



Packed Shift

```
#include <immintrin.h>
__m128i _mm_sll_epi64(__m128i reg, __m128i count);
```

Shift the two 64 bit numbers packed in reg left by count.

Pre Fetching!

```
__builtin_prefetch (const void *addr[, rw[,
locality]])
```

Takes:

- Addr to prefetch from
- Read mode or write mode
- Locality (L1 /L2/ L3/ auto)

Intel Intrinsics Guide

Intel® Intrinsics Guide

Coding time!