Of each would you suggest:

Load/store, branch, and ALU units, what number

building a superscalar based on this with

Intel provided this benchmark. If they are

Review Problem 55
void max(int vals[], int ten) {
    int i = 0;
    for (; i < ten; i++) {
        int max = vals[i];
        for (int j = i + 1; j < ten; j++) {
            if (vals[j] > max) {
                max = vals[j];
            }
        }
        if (max > result) {
            result = max;
        }
    }
    return result;
}

Example: Uniprocessor MAX

Using a multiprocessor is significantly more complex.
<table>
<thead>
<tr>
<th>GPU</th>
<th>SIMD</th>
<th>SISD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-</td>
<td>Multi-</td>
<td>Multi-</td>
</tr>
<tr>
<td>Stream</td>
<td>Stream</td>
<td>Instruction</td>
</tr>
<tr>
<td>Single</td>
<td>Single</td>
<td>Single</td>
</tr>
<tr>
<td>Data Streams</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Categories models of parallel computation:

Readings: 6.3, 6.6

Flynn's Taxonomy
Example: Compute \( y = ax + v; x = a, x_{19} = 8x, x_{20} = 8y \)

Add vector operations. E.g. \( \text{ADDV V0, V1, V2} \)

Add vector registers; each holding say 32 ints: \( V0, V1, V2 \)

**Early SIMD: Vector Processors**
Can't accelerate non-vectorized code

Scatter/Gather instructions

Working on data that isn't "packed" together

Predicated instructions, but still wasteful

"Jumpy" code: if-then case

Bad:

Easily parallelized

Remove loop overhead

Block-based LDR/STUR moves (like cache blocks)

Repetitive computations

Good:

Vector Considerations
Large cache: 0.75MB (comparable CPU might be 8MB)

8-16 lanes/SIMD Processor

8-16 SIMD Processors

Example

Multicore

Provide many parallel processors (SIMD), that are internally SIMD.

Hide latency via multi-threading

Skip multilevel caches, use high-bandwidth memory

GPU datasets smaller than CPUs (4-6GB vs. 32-256GB)

CPU coprocessor, so don't have to support everything.

GPUs, vs. CPUs

Processor optimized for graphics processing, coprocessor to main CPU