• Lab 8
  – A was due today
  – B due 6am this Wednesday (12/1)
  – C due 6am next Monday (12/6)

• Leave
  – Doesn’t update PC: just esp and ebp
  – Don’t need to pass in anything, just hard-code registers
void combine4(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    int i;
    for (i = 0; i < length; i++)
        t = t OP d[i];
    *dest = t;
}
void combine4(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    int i;
    for (i = 0; i < length; i++)
        x = (x OP d[i]);
    *dest = x;
}

void unroll2a_combine(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    int i;
    int limit = length - 1;
    for (i = 0; i < limit; i+=2) {
        x = (x OP d[i]) OP d[i+1];
    }
    for (; i < length; i++) {
        x = x OP d[i];
    }
    *dest = x;
}
Loop Unrolling

- Perform 2x more useful work per iteration

```c
void unroll2a_combine(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    int limit = length - 1;
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    int i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
        x = (x OP d[i]) OP d[i+1];
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
        x = x OP d[i];
    }
    *dest = x;
}
```
## Effect of Loop Unrolling

<table>
<thead>
<tr>
<th>Function</th>
<th>Method</th>
<th>Integer</th>
<th>Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>+ 2.0</td>
<td>+ 3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* 3.0</td>
<td>* 3.0</td>
</tr>
<tr>
<td>combine 4</td>
<td>accum in local</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ 3.0</td>
<td>F* 4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* 5.0</td>
<td>D* 5.0</td>
</tr>
<tr>
<td>combine5</td>
<td>unroll by x2</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.50</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.0</td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td>unroll by x3</td>
<td>1.0</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0</td>
</tr>
</tbody>
</table>
Can this change the result of the computation?
• Yes, for FP.
Reassociated Computation

\[ x = x \text{ OP } (d[i] \text{ OP } d[i+1]); \]

- What changed:
  - Ops in the next iteration can be started early (no dependency)

- Overall Performance
  - N elements, D cycles latency/op
  - Should be \((N/2+1)*D\) cycles:
    \[ \text{CPE} = \frac{D}{2} \]
  - Measured CPE slightly worse for FP
Loop Unrolling with Separate Accumulators

```
void unroll2a_combine(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    int limit = length-1;
    data_t *d = get_vec_start(v);
    data_t x0 = IDENT;
    data_t x1 = IDENT;
    int i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
        x0 = x0 OP d[i];
        x1 = x1 OP d[i+1];
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
        x0 = x0 OP d[i];
    }
    *dest = x0 OP x1;
}
```

- Different form of reassociation
Separate Accumulators

\[
x_0 = x_0 \text{ OP } d[i]; \\
x_1 = x_1 \text{ OP } d[i+1];
\]

- **What changed:**
  - Two independent “streams” of operations

- **Overall Performance**
  - \(N\) elements, \(D\) cycles latency/op
  - Should be \((N/2+1)\times D\) cycles:
    \[CPE = D/2\]
  - CPE matches prediction!
Unrolling & Accumulating

Idea

– Can unroll to any degree L
– Can accumulate K results in parallel
– L must be multiple of K

Limitations

– Diminishing returns
  • Cannot go beyond throughput limitations (execution units)
– Large overhead for short lengths
Calculating Improvement

• $\frac{\text{CPE}_{\text{old}}}{\text{CPE}_{\text{new}}}$
Calculating Improvement

- $\frac{CPE_{\text{old}}}{CPE_{\text{new}}}$
- $\frac{CPE_{\text{new}}}{CPE_{\text{old}}}$
- $100 \left( \frac{CPE_{\text{new}} - CPE_{\text{old}}}{CPE_{\text{new}}} \right)$
- $100 \left( \frac{CPE_{\text{new}} - CEP_{\text{old}}}{CEP_{\text{old}}} \right)$
Calculating Improvement

\[ \text{CPE}_{\text{new}} = 1.0 \quad \text{CPE}_{\text{old}} = 2.0 \]

- \[ \frac{\text{CPE}_{\text{old}}}{\text{CPE}_{\text{new}}} = 2.0 \]
- \[ \frac{\text{CPE}_{\text{new}}}{\text{CPE}_{\text{old}}} = 0.5 \]
- \[ 100 \left( \frac{\text{CPE}_{\text{old}} - \text{CPE}_{\text{new}}}{\text{CPE}_{\text{new}}} \right) = 100\% \]
- \[ 100 \left( \frac{\text{CPE}_{\text{old}} - \text{CEP}_{\text{new}}}{\text{CEP}_{\text{old}}} \right) = 50\% \]
Profiling Demo
Amdahl’s Law