HW#8

Jae Lee

324-4

|  |
| --- |
| 2. Move the call to vec\_length out of the loop |
| 3. Directly access the vector data |
| 4. Accumulate results in a local variable |
| 5. Unroll the loop by 2 |
| 6. Unroll the loop by 2 with 2-way parallelism |
| 7. Unroll the loop by 2 and reassociate |
| The results are pretty close what are in the text book. I verified that the optimization techniques are very effective and real. Latency bounds is 2.7 and throughput bounds is 1.00. |

324-5

|  |
| --- |
| 1. The technique used to measure the branch misprediction penalty:  T\_avg(p) = (1-p)T\_OK + p(T\_OL + T\_MP) = T\_OK + p\*T\_MP  p: the probability of misprediction.  In a typical application, the outcome of the test x < y is highly unpredictable, and so even the most sophisticated branch prediction hardware will guess correctly approximately only 50% of the time, which makes our probability to 0.5. With knowing the value for p, we can come up with an equation for “T\_MP”:  T\_ran = T\_avg(0.5) = T\_OK + 0.5\*T\_MP  T\_MP = 2(T\_ran - T\_OK)  Thus, after measuring the values for “T\_ran” and “T\_OK”, we are able to calculate T\_MP, which is missprediction penalty.  2. Misprediction penalty:    Measuring using 131072 iterations  Cycles per function call, predictable branches  Best time = 13  Cycles per function call, unpredictable branches  Best time = 27  ran on:  Ubuntu – 64 bit  Memory: 4GB  Processor: Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz  Optimization level 1 seemed to work the best:  movl %edi, %eax  subl %esi, %eax  movl %esi, %edx  subl %edi, %edx  cmpl %esi, %edi  cmovle %edx, %eax  ret  Measuring using 131072 iterations  Cycles per function call, predictable branches  Best time = 4  Cycles per function call, unpredictable branches  Best time = 4 |