**1. Code Optimization – “Beat the Compiler”**

(a)

|  |  |  |
| --- | --- | --- |
| **Argv**  **Optimization** | **10000000** | **100000000** |
| **-O2** | 14.500000ms | 150.252000ms |
| **-O3** | 16.582000ms | 169.610000ms |

\*The recorded time is the shortest time seen in a test of 5 each.

(b)

1. Processor architecture: Intel Core i7(64-bit x86-64)
2. CPU frequency: 2.6GHz
3. OS: MacOS
4. A standalone system

(c)

1. Loop Unrolling
2. Performance observations

|  |  |  |
| --- | --- | --- |
| **Argv**  **Optimization** | **10000000** | **100000000** |
| **-O2** | 14.069000ms | 154.219000ms |
| **-O3** | 14.559000ms | 154.152000ms |

1. Result analysis

It matches my expectation. Loop unrolling does improve performance, especially for O3 optimization tag.

Reason: loop unrolling can add additional instruction level parallelism and reduce the amount of loop management instructions.

1. Code

void do\_loops(int \*a, int \*b, int \*c, int N) {

int i;

for (i = N-1; i >= 1; i -= 9) {

a[i] = a[i] + 1;

a[i - 1] = a[i - 1] + 1;

a[i - 2] = a[i - 2] + 1;

a[i - 3] = a[i - 3] + 1;

a[i - 4] = a[i - 4] + 1;

a[i - 5] = a[i - 5] + 1;

a[i - 6] = a[i - 6] + 1;

a[i - 7] = a[i - 7] + 1;

a[i - 8] = a[i - 8] + 1;

}

for (i = 1; i < N; i += 9) {

b[i] = a[i + 1] + 3;

b[i + 1] = a[i + 2] + 3;

b[i + 2] = a[i + 3] + 3;

b[i + 3] = a[i + 4] + 3;

b[i + 4] = a[i + 5] + 3;

b[i + 5] = a[i + 6] + 3;

b[i + 6] = a[i + 7] + 3;

b[i + 7] = a[i + 8] + 3;

b[i + 8] = a[i + 9] + 3;

}

for (i = 1; i < N; i += 9) {

c[i] = b[i-1] + 2;

c[i + 1] = b[i] + 2;

c[i + 2] = b[i + 1] + 2;

c[i + 3] = b[i + 2] + 2;

c[i + 4] = b[i + 3] + 2;

c[i + 5] = b[i + 4] + 2;

c[i + 6] = b[i + 5] + 2;

c[i + 7] = b[i + 6] + 2;

c[i + 8] = b[i + 7] + 2;

}

}

1. Loop Unrolling