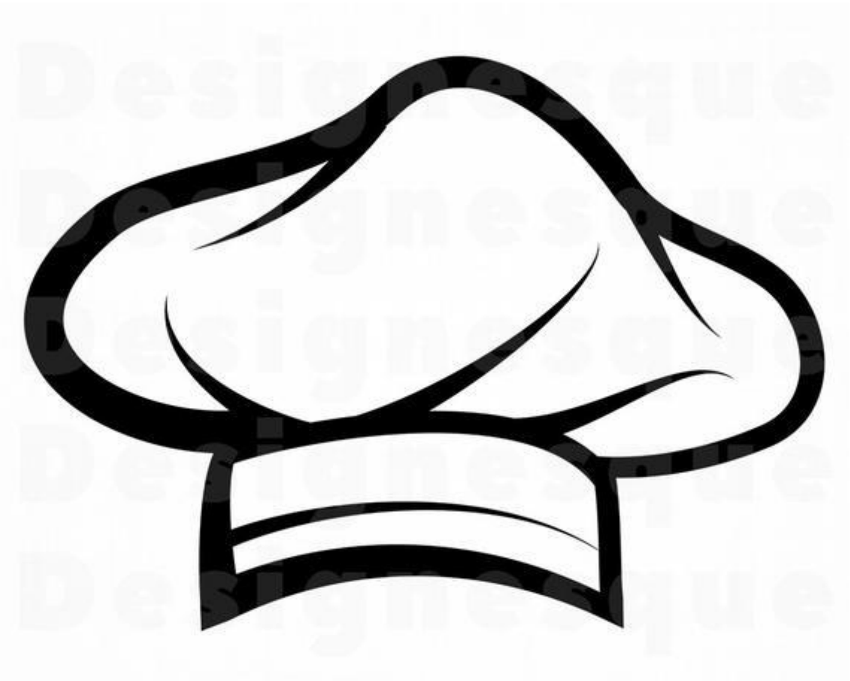


How to Cook with C++

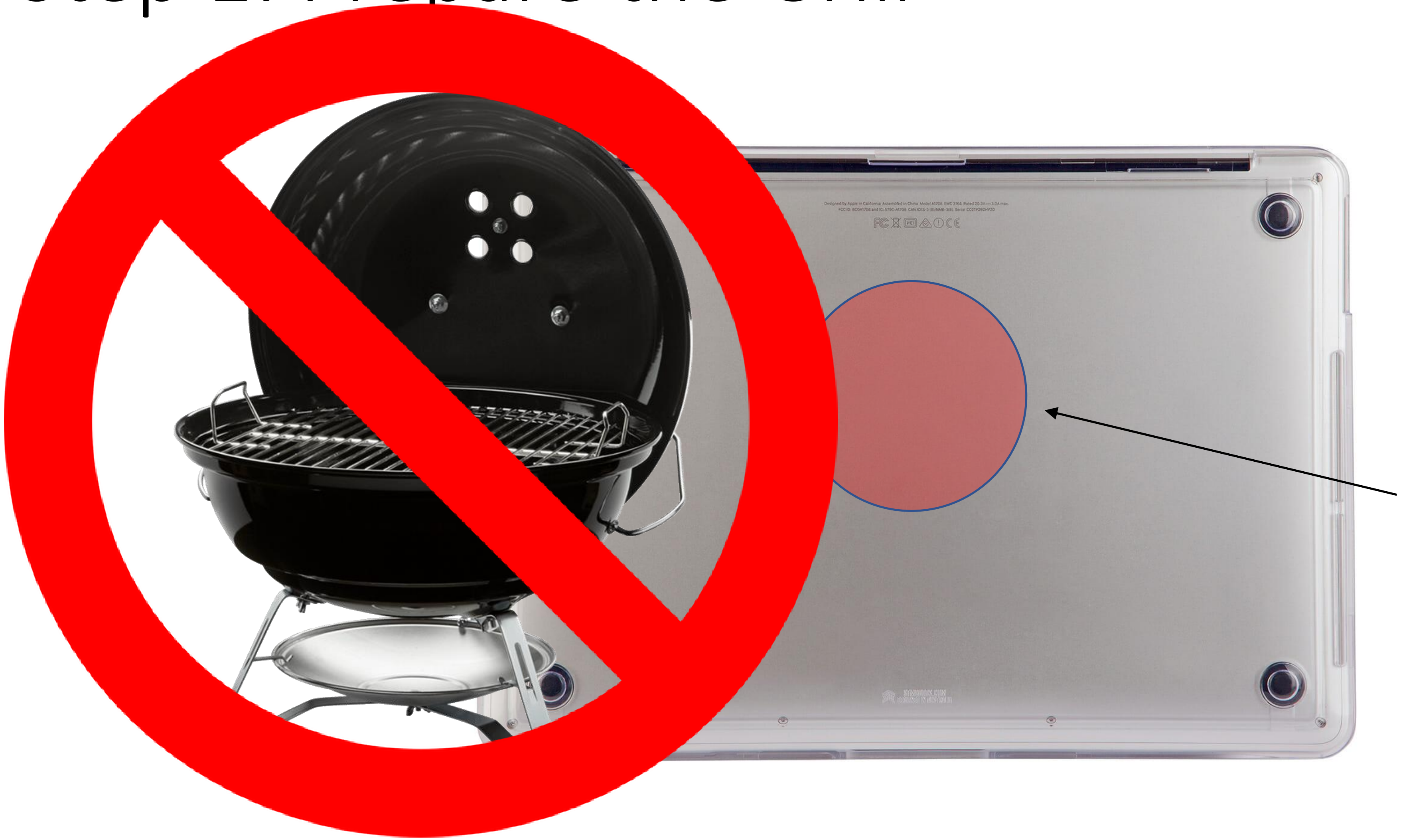
Preparation

- Have you ever wanted to do something **useful** with C++?
- Do you feel like you aren’t really using the **power** of you machine?
- Do you like eggs?

What we will Need

- A GOOD GRILL
- A GOOD GRILL PROGRAM
- A FEW EGGS

Step 1: Prepare the Grill



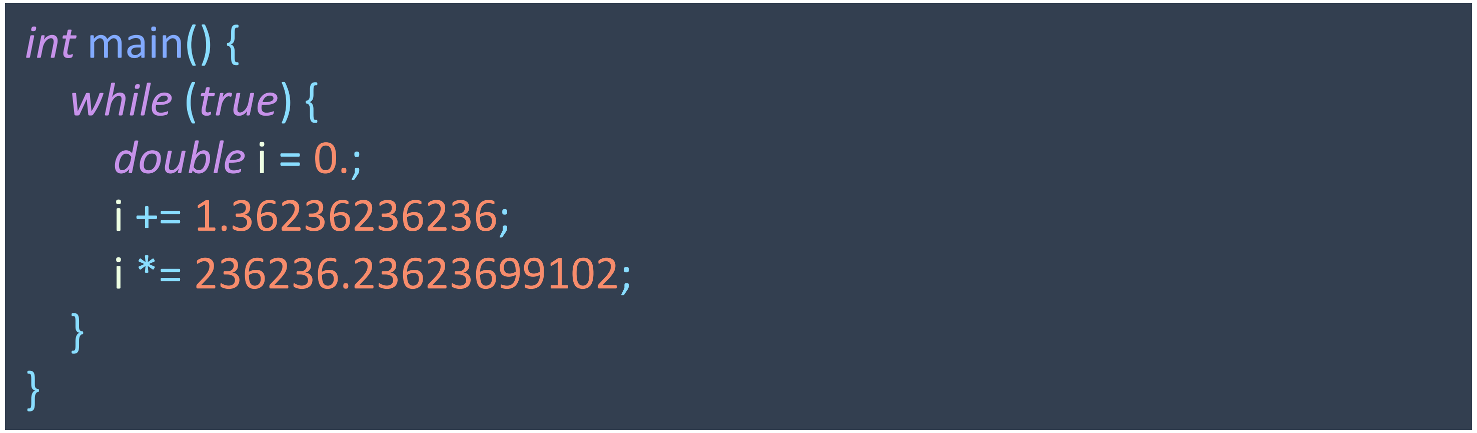
Best Grill Spot

- What constitutes a good grill program?
  - Tons of Loads/Stores ? NO
  - Multithreading ? Yes
  - Vectorization ? Yes
  - High Flops? Yes

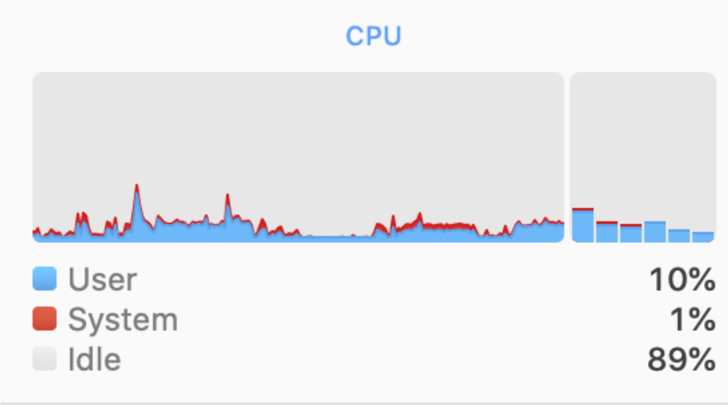
Step 2: Write a Grill Program

- Ok time for Grill_v1.cpp

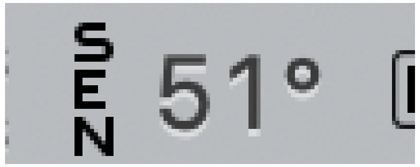
Grill_v1.cpp



```
int main() {
   while (true) {
      double i = 0.;
      i += 1.36236236236;
      i *= 236236.23623699102;
   }
}
```



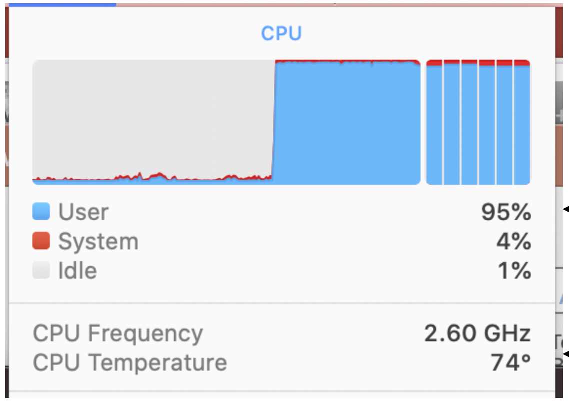
🟦 User	10%
🟥 System	1%
⬜ Idle	89%



g++ -O3 grill_v1.cpp
We are not cooking with heat at all…

Grill_v1_parallel.cpp

```
int main() {
   while(true) {
#pragma omp parallel for
      for (int i = 0; i < 1000; ++i) {
         i += 1.362362326;
         i *= 2336236.68236;
      }
   }
}
```



CPU

🟦 User	95%
🟥 System	4%
⬜ Idle	1%
CPU Frequency	2.60 GHz
CPU Temperature	74°

Utilizing all CPUs

Temp is better but still low

This Grill Program Kinda **Sucks**

Temperature to cook egg ~65°C; need higher temps to account for heat loss

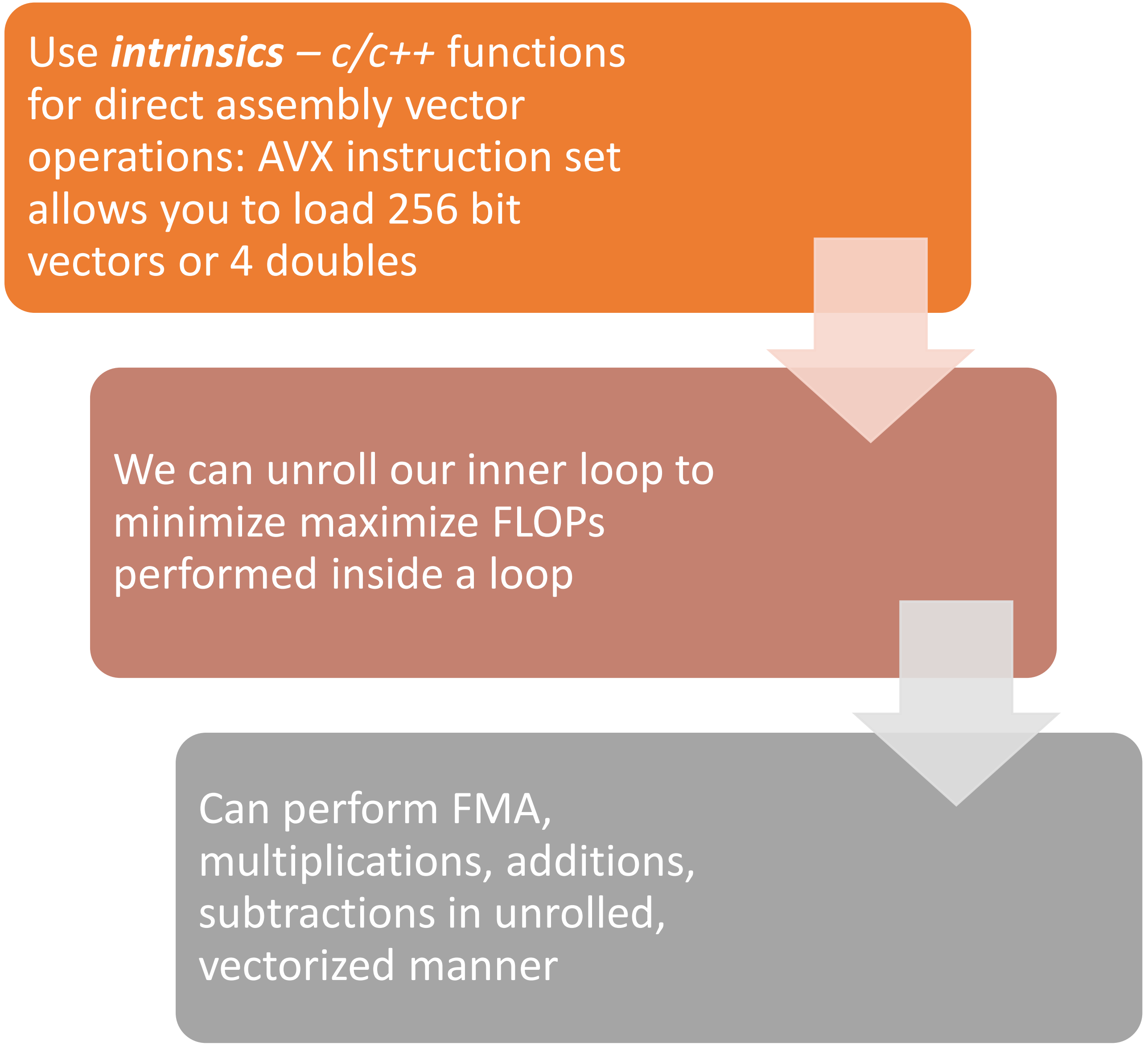
It’s Not achieving **max flops**…

Let’s do some **math**…

My Grill

- 2.6 Ghz intel processor, support for AVX 256 vector operations
- Max flops is **2.6 * 8** flops = **20.8** Gflops
- Max total flops over **6 cores**: 6 * 2.6 * 8 = 124.8 Gflops
- Let’s give it a go…

How to Cook with Gasoline



Use ***intrinsics*** – *c/c++* functions for direct assembly vector operations: AVX instruction set allows you to load 256 bit vectors or 4 doubles

We can unroll our inner loop to minimize maximize FLOPs performed inside a loop

Can perform FMA, multiplications, additions, subtractions in unrolled, vectorized manner

Red Hot Grill

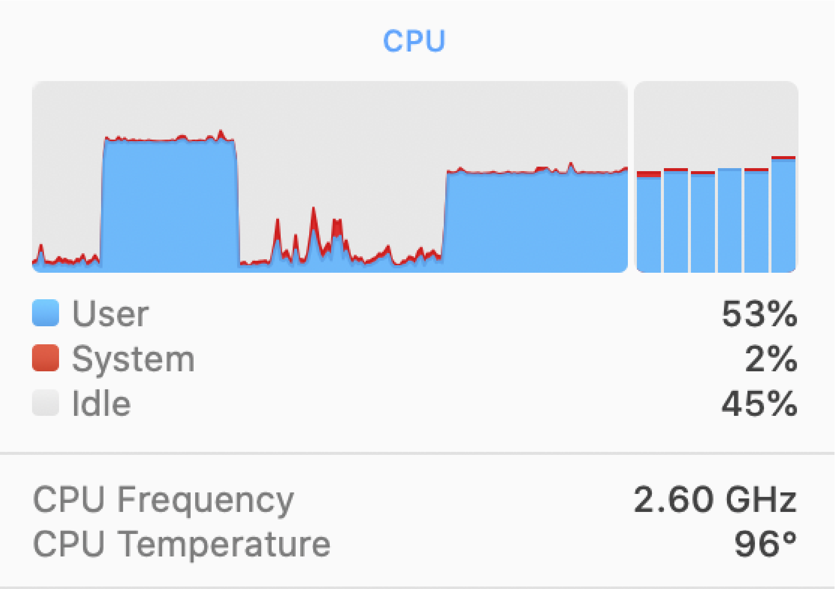
d1	d2	d3	d4

Vector Intrinsics

- **__mm256d** vector = **_mm256_set1_pd**(5) – Broadcast a double to a 4 double wide vector
- **_mm256_fmadd_pd**(__mm256d v1, __mm256d v2, __mm256d v3) – Fused Multiply Add
- **_mm256_add_pd**(__mm256d v1, __mm256d v2) – Vectorized Add
- **_mm256_mul_pd**(__mm256d v1, __mm256d v2)– Vectorized Multiply

```
_mm256_add_pd (v1, v2);
_mm256_mul_pd (v1, v2)
_mm256_mul_pd (v1, v2)
_mm256_add_pd (v1, v2);
_mm256_mul_pd (v1, v2)
        ....
```

UNROLLED



CPU

🟦 User	53%
🟥 System	2%
⬜ Idle	45%
CPU Frequency	2.60 GHz
CPU Temperature	96°

Voila