Assignment 1 – Task 1

Vector triad code

```
double wcs,wce,ct;
int repeat=1,...;
double runtime=0.0;

for(; runtime<.1; repeat*=2) {
  wcs = getTimeStamp();
  for(int r=0; r<repeat; r++){
    for(int i=0; i<SIZE; i++){
      a[i]=b[i]+c[i]*d[i];
    }
    if(a[SIZE>>1]<0){
      dummy(a,b,c,d);
    }
  }
  wce = getTimeStamp();
  runtime = wce-wcs;
}
repeat /= 2;
printf("Perf: %.1lf Mflop/s\n",SIZE*(double)repeat*2.0/runtime);
```

Ensure proper time measurement

Actual benchmark loop

Prevent compiler from optimizing the r loop (or just everything) away

Do not forget to initialize your arrays with valid FP numbers!

Assignment 1 – Task 1

Vector update

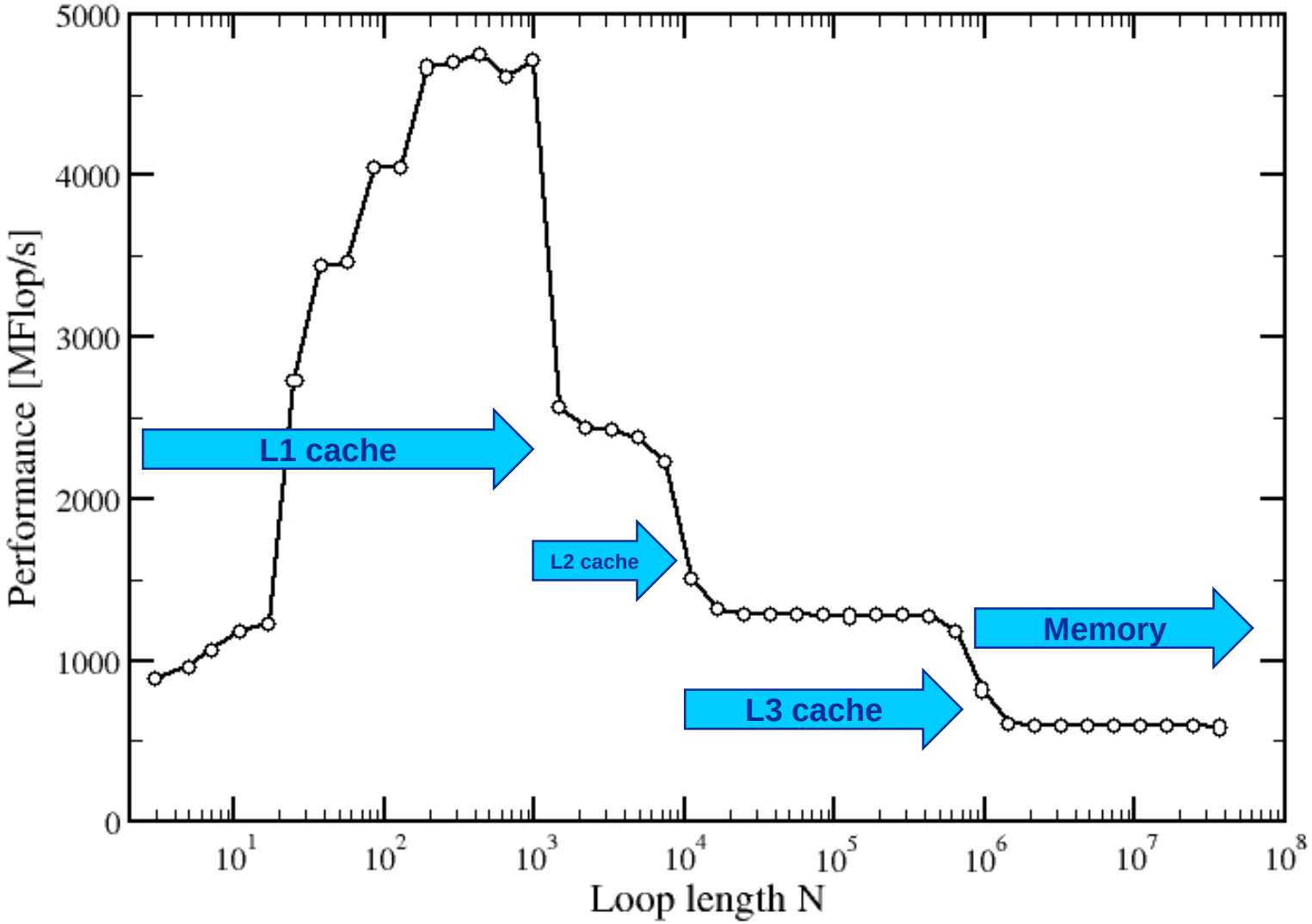
```
#define OFFSET ...
for(int r=0; r<repeat; r++){
  a[0] = a[SIZE-1]; // important
  for(int m=OFFSET; m<SIZE; m++){
    a[m] = s * a[m-OFFSET];
  }
  <DUMMY>
}
```

At OFFSET=1, we expect each iteration to take at least as many cycles as the latency of the DP MULT instruction – 5 cy in case of Emmy.

→ $P_{\max} = \frac{1\ \text{flop}}{5\ \text{cy}} \times 2.2\ \text{Gcy/s} = 440\ \text{Mflop/s}$ (light speed limit!)

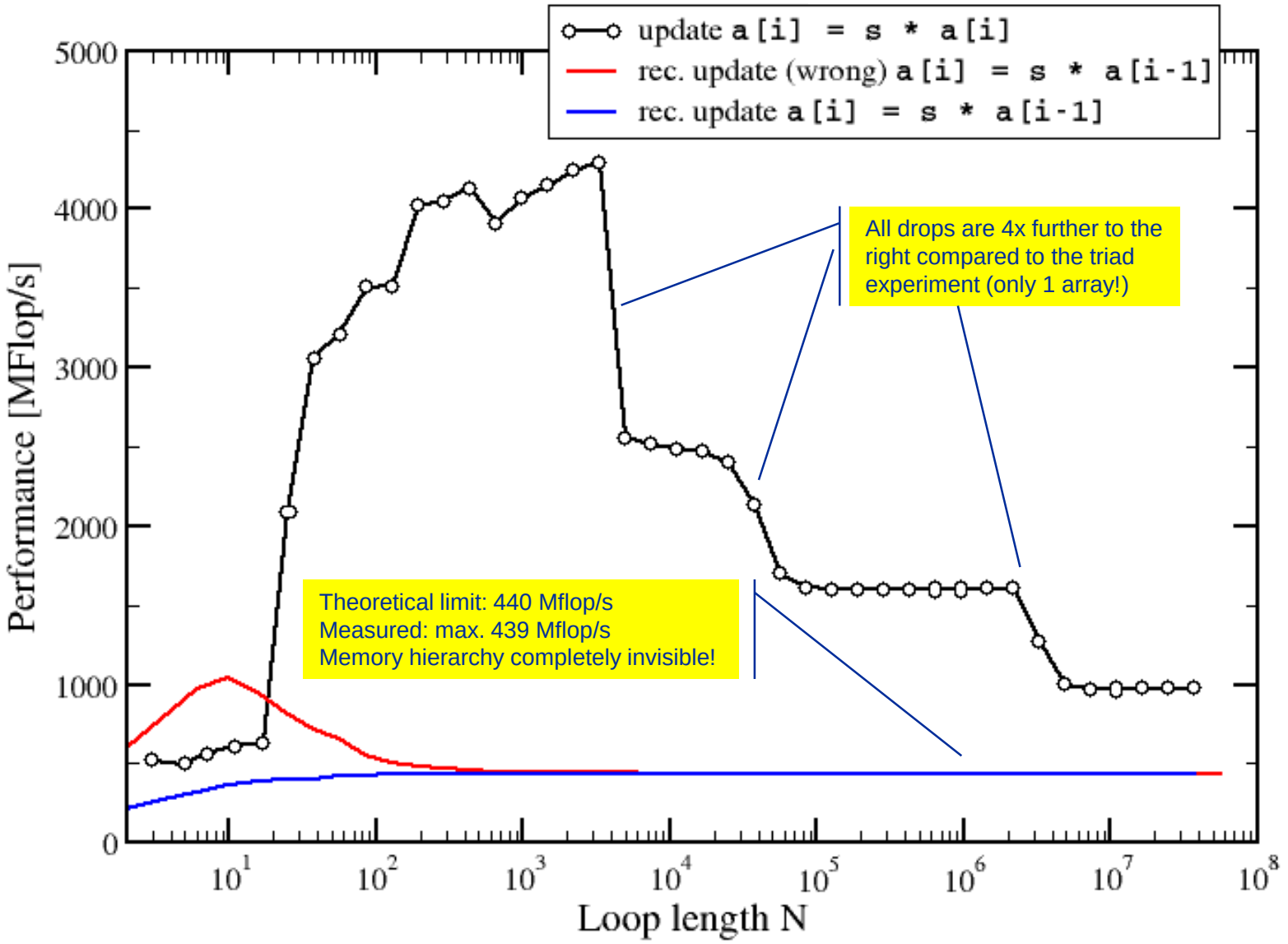
Vector Triad on Emmy @ 2.2 GHz

`A(:)=B(:)+C(:)*D(:)`



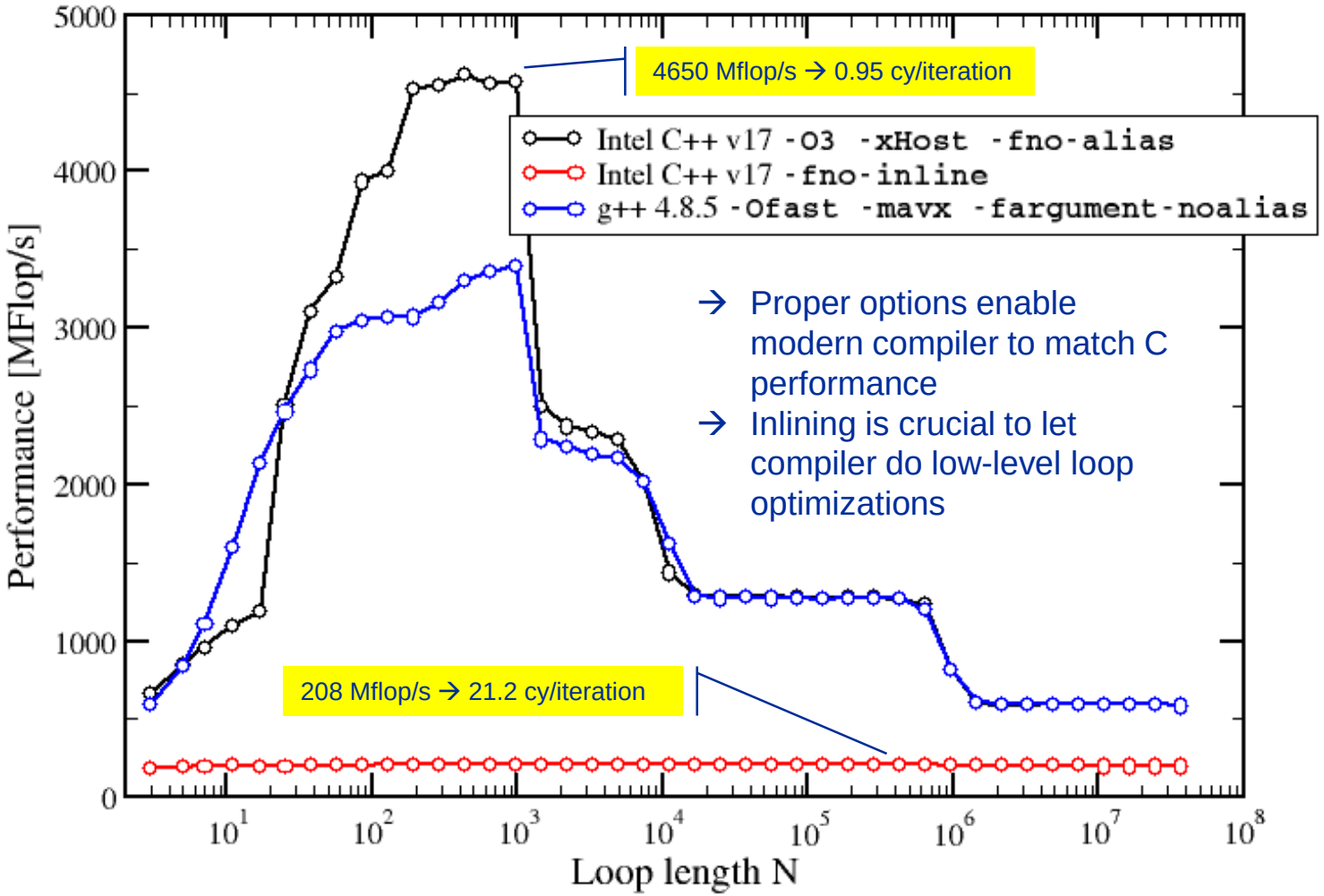
Vector update on Emmy @ 2.2 GHz

```
for(int m=OFFSET; m<SIZE; m++) {
    a[m] = s * a[m-OFFSET];
}
```



Assignment 1 – Task 2: Vector Triad C vs. C++ (Emmy @ 2.2 GHz)

```
icpc –O3 -xHost -fno-alias ...
gcc –Ofast –mavx –fargument-noalias …
```



Assignment 1 – Task 2: Vector Triad C vs. C++

- Triad source code is the same as in C and C++:

```
for(int i=0; i<SIZE; i++){
   a[i]=b[i]+c[i]*d[i];
}
```

- C++
  - `a[i]` is a call to `std::vector<double>::operator[](size_type)`
  - Not inlining the operator incurs one function call per iteration
  - … and who knows what happens *inside* `operator[](size_type)`
  - On top of this, automatic SIMD vectorization is ruled out for loops with function calls → scalar code