SESSION 3

ROOFLINE MODELS

MASSIMILIANO FASI

Durham University

Types of resources

Scalable

Scales linearly

- private resources
- floating-point units
- CPU cores

Saturating

Scales sublinearly

- shared resources
- L3 memory
- RAM

Types of resources

Scalable

Scales linearly

- private resources
- floating-point units
- CPU cores

Saturating

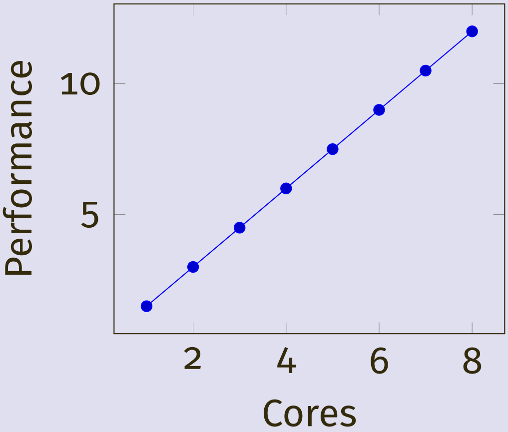
Scales sublinearly

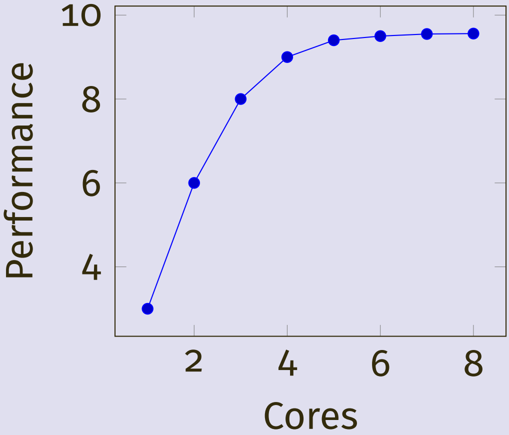
- shared resources
- L3 memory
- RAM

Bottlenecks

Saturating resources are the limiting factor.

Scalable vs. Saturating





More realistic memory benchmark

The `clcopy` benchmark we used

- **only** touches one byte in each cache line
- **only** provides upper bounds
- is not a realistic workload

State-of-the-art alternative

- STREAM benchmark[1]
- most commonly used is TRIAD
- available in `likwid-bench` as `stream_triad_XXX`

[1] https://www.cs.virginia.edu/stream/

The TRIAD loop

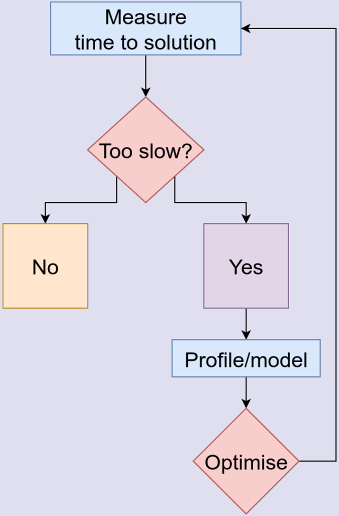
```
double *a, *b, *c;
double alpha = 1;
...
for (int i = 0; i < N; i++)
   a[i] = b[i]*alpha + c[i];
```

The TRIAD loop

```
double *a, *b, *c;
double alpha = 1;
...
for (int i = 0; i < N; i++)
    a[i] = b[i]*alpha + c[i];
```

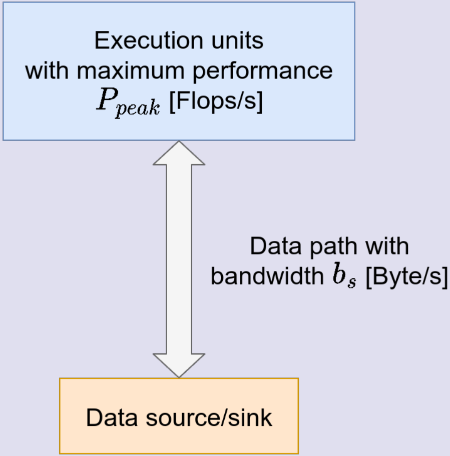
- 2 floating point operations
- 2 loads
- 1 store

Code optimisation



Simple model for loop heavy code

Simple view of **hardware**



Simple view of **software**

```
// Possibly nested loops
for (i = 0; i < ...; i++)
  // Complicated code doing
  // N Flops causing
  // B bytes of data transfer
```

Operational intensity [Flops/B]

$$I_c = \frac{N}{B}$$

The roofline model

What is the performance P of a code?

How fast can work be done? P measured in Flops/s

The roofline model

What is the performance P of a code?

How fast can work be done? P measured in Flops/s

The bottleneck is either:

- execution of work P_{peak}
- or the data path $I_c \cdot b_s$

Therefore:

$$P = \min(P_{peak}, I_c \cdot b_s)$$

The roofline model

What is the performance P of a code?

How fast can work be done? P measured in Flops/s

The bottleneck is either:

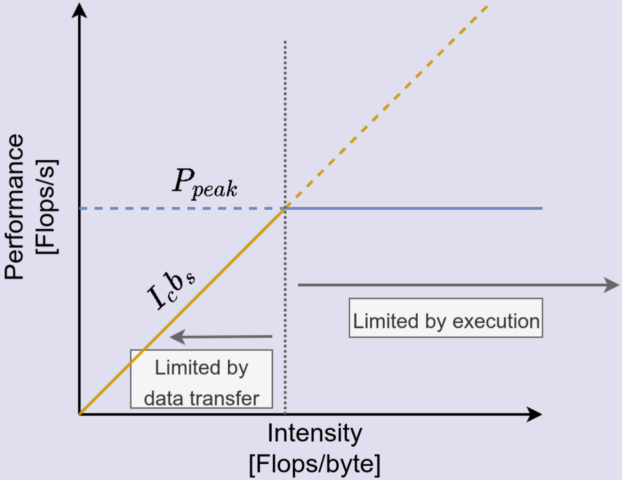
- execution of work P_{peak}
- or the data path $I_c \cdot b_s$

Therefore:

$$P = \min(P_{peak}, I_c \cdot b_s)$$

Optimistic model: everything happens at “light speed”.

Roofline diagram



Applying roofline

Roofline characterises performance using three numbers:

HW1. P_{peak}: peak floating point performance

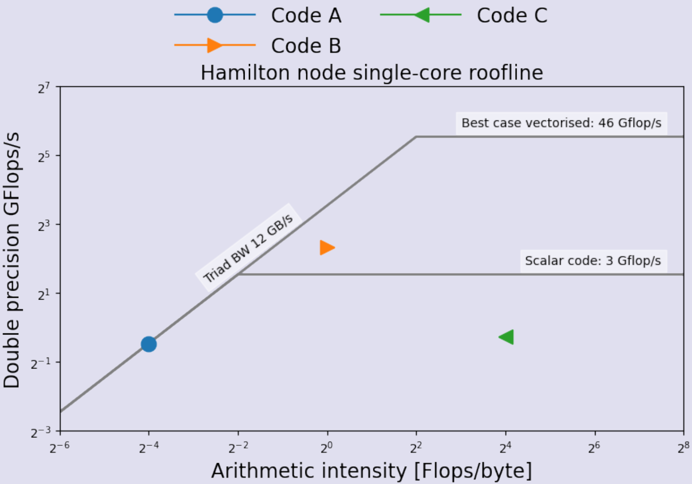
HW2. b_s: streaming memory bandwidth

SW1. I_c: operational intensity of the code

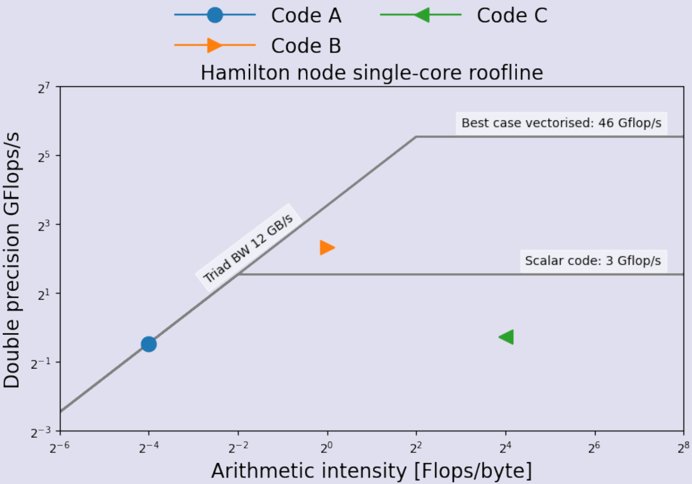
Process

1. Measure these numbers
2. Draw diagram
3. Use diagram to chose optimisations likely to pay off

Guide for optimisation choices

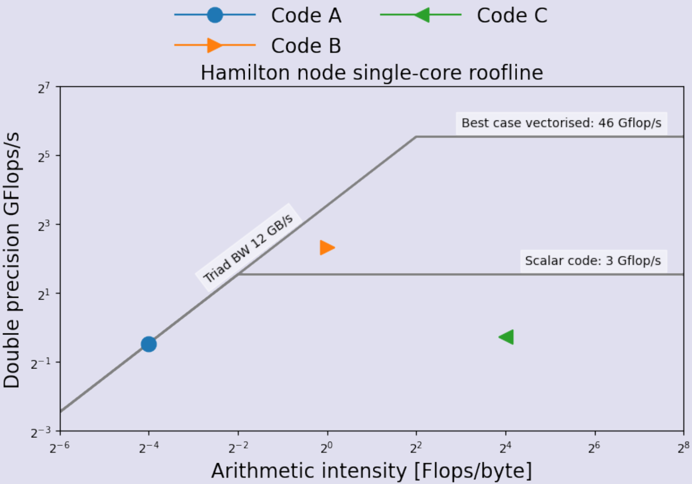


Guide for optimisation choices



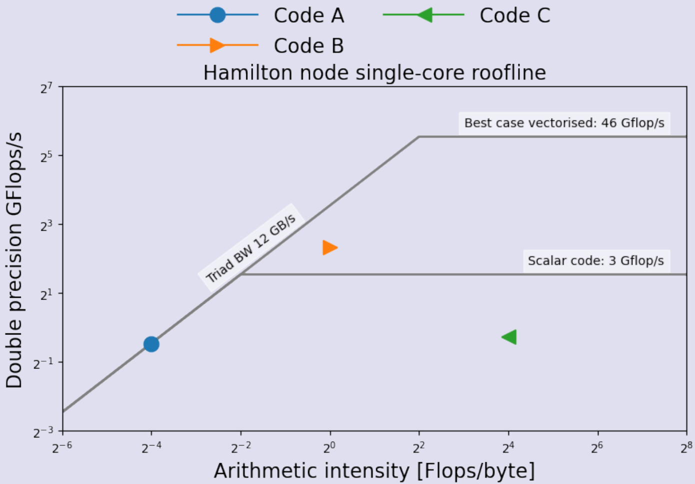
Which codes might benefit from vectorisation?

Guide for optimisation choices



How much improvement could we expect?

Guide for optimisation choices



Which codes might benefit from refactoring to increase I_c?

Determining the memory bandwidth

Data transfers are modeled with **streaming memory bandwidth**

Estimating streaming memory bandwidth (STB)

1. **Computation**
   - find out speed of memory M_s
   - find out number of memory channels C
   - STB in B/s is $C \times M_s \times 8$
   - speed of memory often unknown in practice
2. **Measurement** using STREAM
   - typical solution (see exercise 4)

Determining floating point throughput

Absolute peak can be estimated from

- specification sheet frequency
- knowledge of hardware architecture

AMD Zen 2 architecture **?**

- Floating point instructions execute on 4 ports
- Up to 4 "μops" issued per cycle
- up to 2 floating point instructions per cycle
- MUL and FMA ($y \leftarrow a + b \times c$) are issued on ports 0 and 1
- ADD are issued on ports 2 and 3
- DIV are only issues on port 3

Example

Assuming a maximum clock speed of 3.35GHz

Example: best case

For code with only double precision SIMD FMAs, peak throughput is

$$\overbrace{3.35}^{\text{clock speed}} \times \underbrace{2}_{\text{dual issue}} \times \overbrace{4}^{\text{vector width}} \times \underbrace{2}_{\text{FMA}} = 53.6\text{GFlops/s}$$

Example

Assuming a maximum clock speed of 3.35GHz

Example: only DIVs

Code only does double precision SIMD DIVs

$$\overbrace{3.35}^{\text{clock speed}} \times \underbrace{1}_{\text{single issue}} \times \overbrace{4}^{\text{vector width}} = 13.4\text{GFlops/s}$$

Determining machine characteristics

- Sometimes multiple “roofs” for different instruction mixes
- Calculations are complicated by frequency scaling as well

More details

- `https://wikichip.org` for spec sheets
- `https://uops.info` for μops execution throughput
- Travis Down’s discussion on finding limiting factors in (simple) code

Computing arithmetic intensity

Two options:

1. measurement using performance counters
2. pen-and-paper method
   - count floating point operations
   - count data accesses
   - use formula $I_C = N/B$

Assessing operational intensity

```
double *a, *b, *c, *d;
...
for (i = 0; i < N; i++) {
  a[i] = b[i]*c[i] + d[i]*a[i];
}
```

Counting operations

- 3 double-precision Flops/iteration
- 3N total double-precision Flops
- Notice we don't care what operations these are

Assessing operational intensity

```
double *a, *b, *c, *d;
...
for (i = 0; i < N; i++) {
  a[i] = b[i]*c[i] + d[i]*a[i];
}
```

Counting data accesses

- Load counts as one access, write as two (one load, one store).
- 3 reads, 1 write per iteration.
- $8 \times 5N$ total bytes

A model of cache

Perfect cache

- Lower bound
- Data moved to cache once
- Counts *unique* memory accesses
- $8 \times 2M + 8 \times 3N$ total bytes

Pessimal cache

- Upper bound
- Each array access misses cache
- Counts *total* memory accesses
- $8 \times 2MN + 8 \times 3MN$ total bytes

- These bounds are typically not tight
- Better bounds normally require more work in the analysis
- Best employed in combination with measurement of operational intensity

Exercises 4: roofline for matrix-vector multiply

1. Split into small groups
2. Make sure one person per group has access to Hamilton
3. Benchmark memory bandwidth as a function of vector size
4. You can use the bash script from last week.
5. Ask questions!

$$y = Ax = \sum_{j=1}^{n_{col}} A_{ij} x_j$$