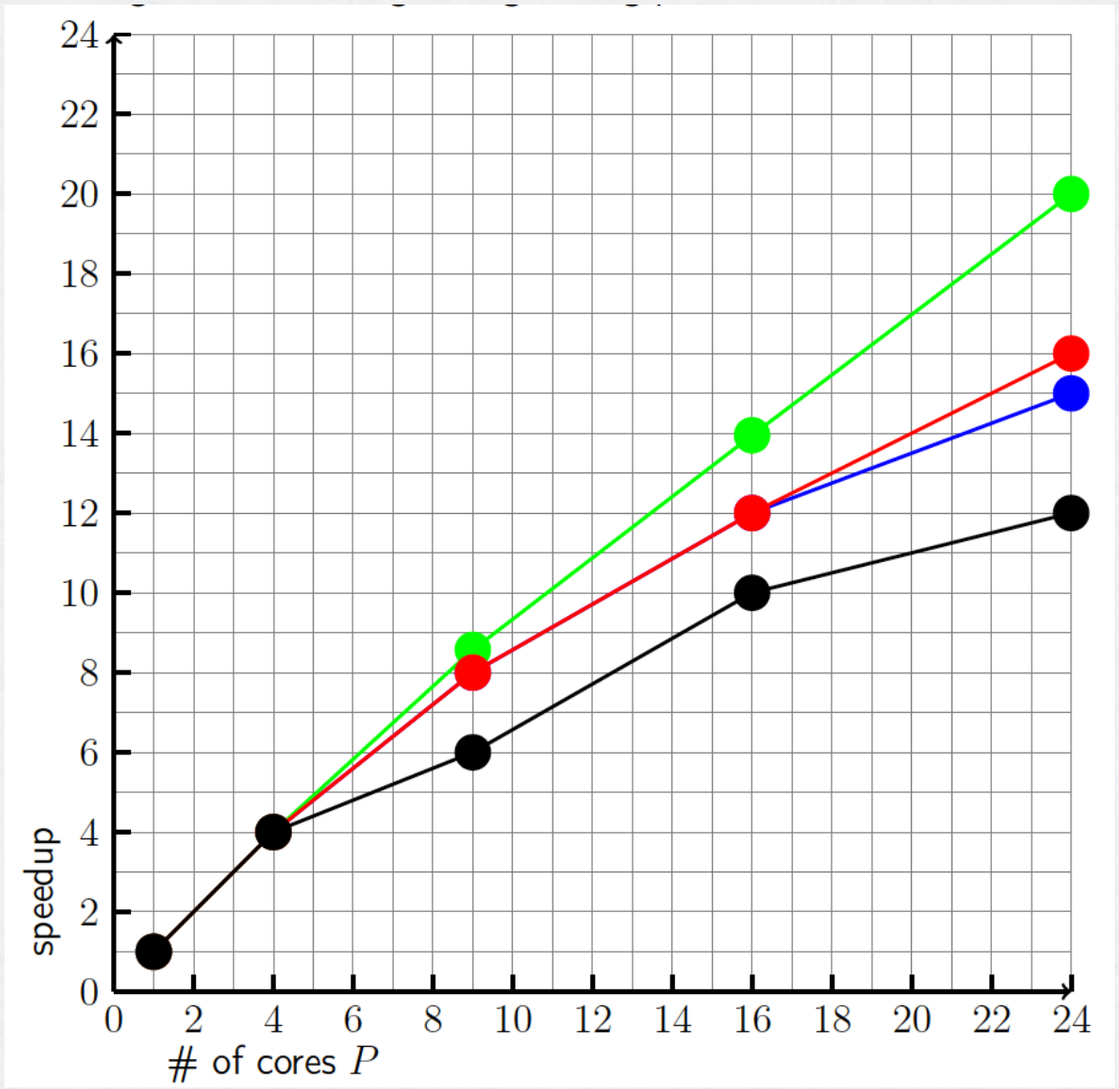
Exercises - 1a

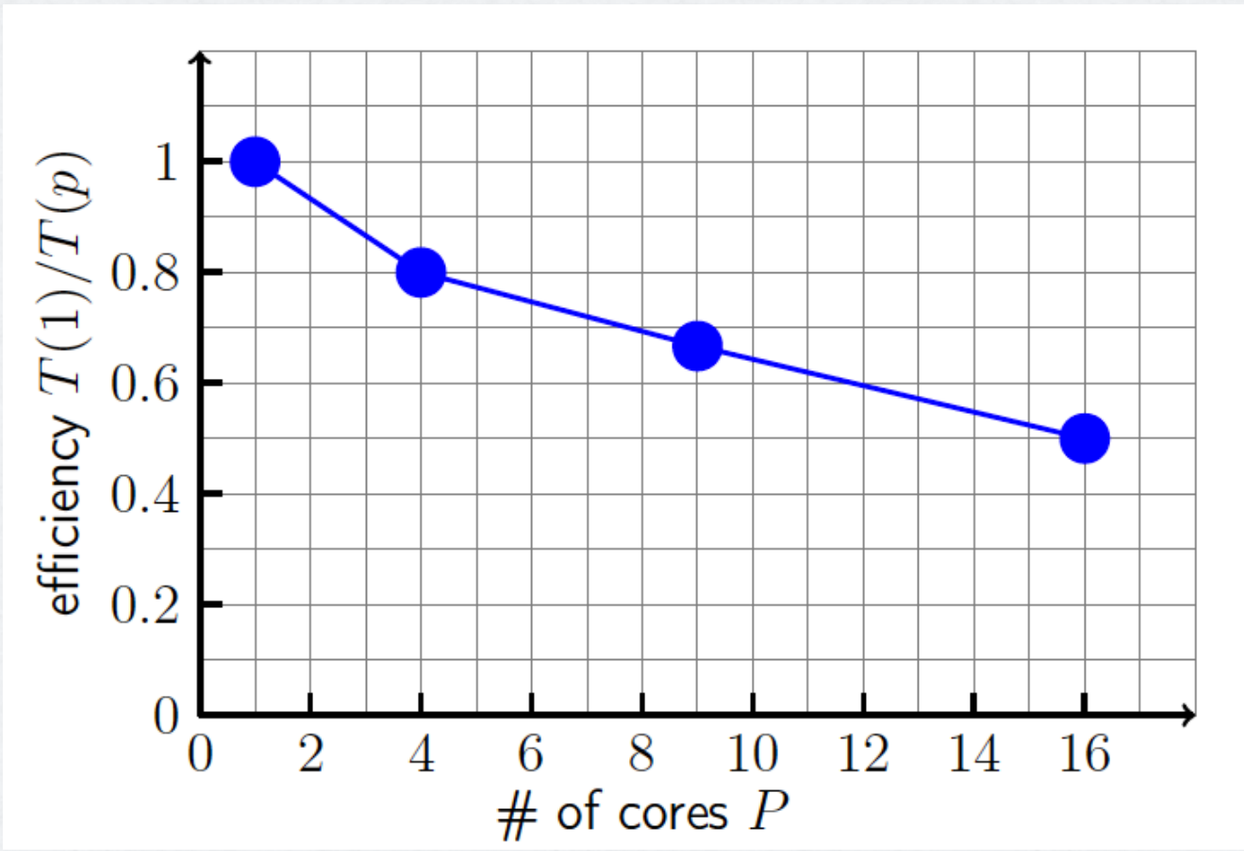
- Assume we implemented a simple N^2 solver for the N-body problem. The following table reports timing results for the solver for various number of particles N on P processor cores with a fixed number of time steps:

	runtime [s]			
P	$N = 500$	$N = 1000$	$N = 1500$	$N = 2000$
1	6.00	30.00	72.00	120.00
4	1.50	7.50	18.00	30.00
9	0.75	3.50	9.00	20.00
16	0.50	2.15	6.00	12.00
24	0.40	1.50	4.50	10.00

- Draw at least four points of the strong scaling plot for this program.
- Draw at least four points of the weak scaling plot for this program, using the value for N = 500 at P = 1 as reference to estimate the parallelization overhead.

Exercises - 1a





Exercises - 1b

- Assume you work on Euler and you have one node with 24 cores that you can use to solve a problem in parallel for which 91% of your code is parallelizable. Can you get a speedup of 8? If so, how many cores are needed at least?

- Profiling a serial code for Molecular Dynamics you find that 90% of the time is spent in a large loop with independent iterations (perfectly parallelizable with N threads), another 5% is spent in a region that can be parallelized with at most 2 threads and the remaining part is purely serial.
  - Given Amdahl's law, what is the strong scaling for N -> infinity ?
  - For what value of N is the speedup equivalent to 90% of the asymptotic maximum?

Exercises - 1b

$$S(p) = \frac{T(1)}{T(p)} = \frac{T(1)}{0.09T(1) + 0.91T(1)/p} = \frac{1}{0.09 + 0.91/p} \stackrel{!}{=} 8$$

$$\Rightarrow p = \frac{8 \times 0.91}{1 - 8 \times 0.09} = 26$$

$$\frac{T_1}{T_N} = \frac{1}{0.05 + \frac{0.05}{2} + \frac{0.90}{N}} = \frac{200N}{15N + 180}$$

$$\lim_{N\to\infty} \frac{T_1}{T_N} = \lim_{N\to\infty} \frac{200N}{15N + 180} = \frac{200}{15} = \frac{40}{3} \approx 13.33$$

$$\frac{200N}{15N + 180} = \frac{90}{100}\frac{40}{3} = 12 \Rightarrow N = 108$$

Exercises - 2

- What is the operational intensity of the following code?

```
1  float A[N], B[N], C[N];
2  ...
3  const int P=2;
4  for(int i=0; i<N; ++i) {
5    unsigned j = 0;
6    while(j < P) {
7      A[i] = B[i]*A[i]+0.5;
8      ++j;
9    }
10   C[i] = 0.9*A[i]+C[i];
11 }
```

- A compute node has a peak perf. of 409.7 GFLOP/s (single precision) and a memory bandwidth of 34 GB/s.
  - For which range of positive integer values P is the above code memory bound? Assume an infinite cache and state any further assumption you made.
- Draw the roofline model for the compute node.

Exercises - 2

Assumptions: we have enough registers.

```
1       A[i] = B[i] * A[i] + 0.5;
2       A[i] = B[i] * A[i] + 0.5;
3       C[i] = 0.9*A[i] + C[i];
```

memory operations: read A,B, C write A, C, (3+2)*4=20 bytes, fp operations: 3 mul + 3 add = 6 flops

OI = 6/20

Ridge point = 409.7 / 34 = 12.05 Assuming perfect caching meaning we have enough registers so A[i], B[i], C[i] are read only once from the memory and A[i], C[i] are written only once to the memory.

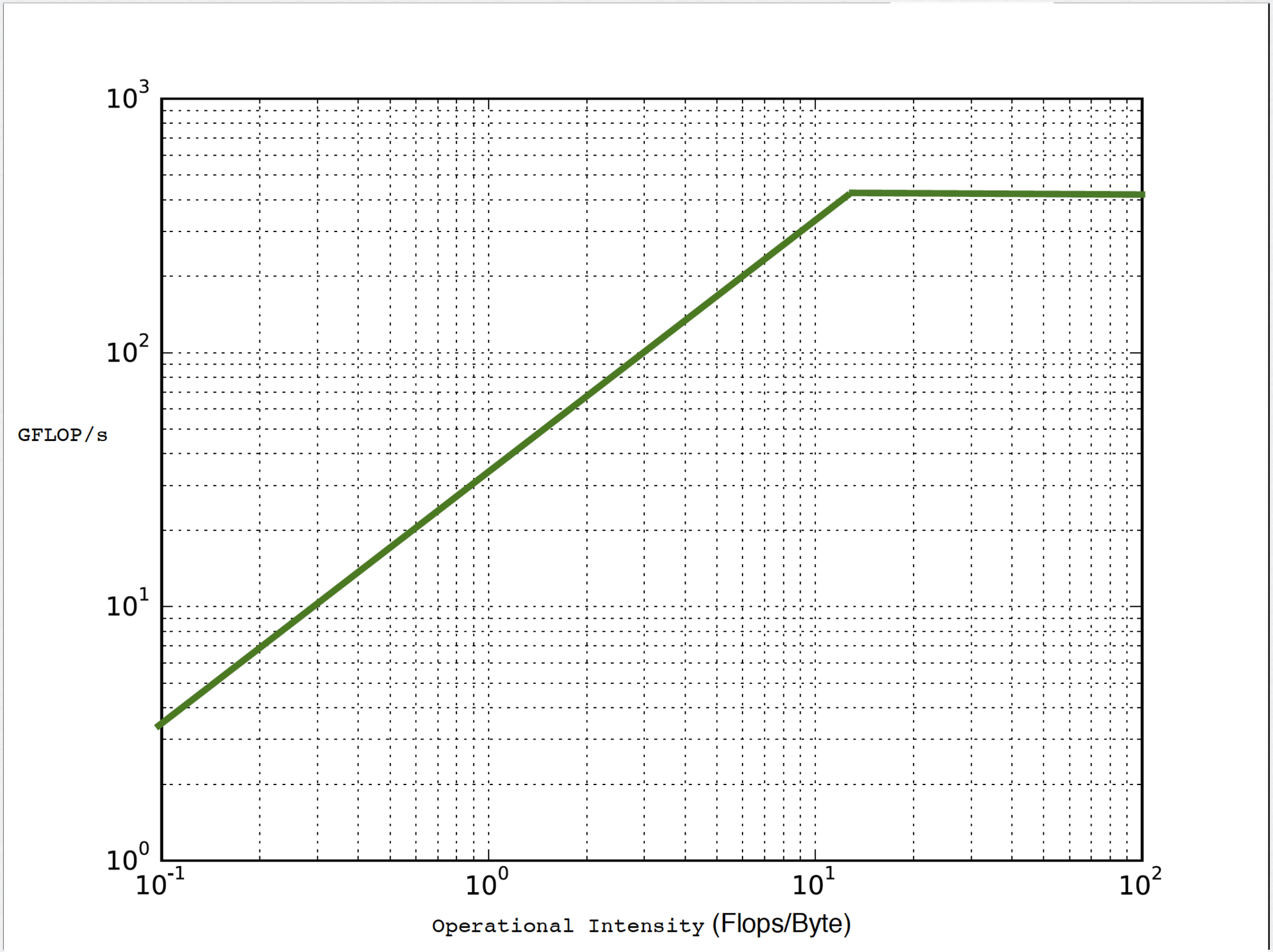
```
1       A[i] = B[i] * A[i] + 0.5;       \\ 2 FLOP
2       ...
3       A[i] = B[i] * A[i] + 0.5;       \\ 2 FLOP
4       C[i] = 0.9*A[i] + C[i];         \\ 2 FLOP
```

OI > Ridge point = 12.05

Flops = (2 + 2P), Bytes = 5*4 = 20

$(2 + 2P)/20 < 12.05$
$2 + 2P < 20 * 12.05 = 241$
$2P < 239$
$P < 239/2 = 119.5$
$P < 119$

Exercises - 2


10^3
10^2
10^1
10^0
GFLOP/s
10^{-1}
10^0
10^1
10^2
Operational Intensity (Flops/Byte)

Exercises - 3

In the following MPI code, rank 0 distributes a number ($= M * size$) of input values to all ranks of the MPI application. Each input value is processed by the `do_work()` function, which has uniform but significant execution time.

1. Identify and explain possible issues in the MPI code.
2. Explain how you can address the above issues.

```
1  // void do_work(int);
2  // rank: MPI process id
3  // size: number of MPI processes
4
5  int M = 2;  //  any value > 1
6  int input;
7
8  if (rank == 0) {
9      srand48(time(0));
10
11     int N = M*size;
12     for (int i = 0; i < N; i++) {
13         input = lrand48() % 1000;    // some random value
14         MPI_Send(&input, 1, MPI_INT, i%size, 100, MPI_COMM_WORLD);
15     }
16 }
17
18 MPI_Status status;
19 for (int i = 0; i < M; i++) {
20     MPI_Recv(&input, 1, MPI_INT, 0, 100, MPI_COMM_WORLD, &status);
21     do_work(input);
22 }
```