Programming Parallel Computers

Jukka Suomela · Aalto University · ppc.cs.aalto.fi

Part 3B:
Reusing data in registers

Main memory is slow

CPU

- 64 additions per clock cycle
  - one addition: two floats
  - float: 4 bytes
- ***512 bytes of input data*** per clock cycle to arithmetic units

Main memory

- Bandwidth 34.1 GB/s
  - clock speed 3.3 GHz
- ***10 bytes of input data*** per clock cycle from memory

Factor 50 difference!

Main memory is slow

- If you try to read all input from the main memory, you will only use 2% of the performance of your CPU
- For full performance, you can only afford to get *2% of our input from main memory*
- Everything else has to come from:
  - **CPU registers**
  - **caches** (preferably those close to the CPU)

What registers are there?

- *16 integer registers*
  - `rax`, `rbx`, …, `r8`, `r9`, …, `r15`
  - **64 bits**
  - e.g. 1 × `long long` or 1 × pointer
- *16 vector registers*
  - `ymm0`, `ymm1`, …, `ymm15`
  - **256 bits**
  - e.g. 4 × `double` or 8 × `float`
- Accessing values stored in registers is free

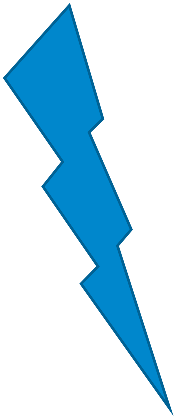
This is just the programmer's view of the CPU.

There are more physical registers, but we cannot directly refer to them.

For more details on this, search for "register renaming"

How are registers used?

- Your ***C++ compiler*** will keep all kinds of ***local variables*** and temporary values in registers whenever it can
  - it tries very hard, just make sure there are no obstacles for that
- **Main limitations:**
  - you cannot have ***“pointers to registers”***
    - `float x; float *p = &x; something(p);`
  - you cannot do ***“array indexing with registers”***
    - `float y[4]; int i = calculate(); float z = y[i & 3];`
  - there are not ***that many registers***
    - `float8_t a[1000];`



How would we like to use registers?

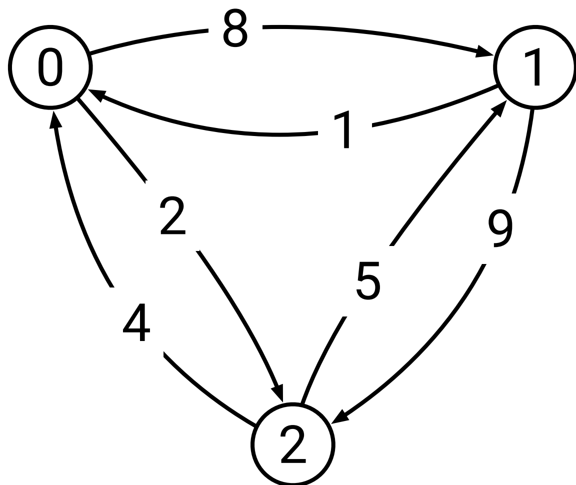
- Manipulating data in memory is slow,
  manipulating data in registers is free
- Read as little as possible from memory,
  keep as much useful data in registers as possible

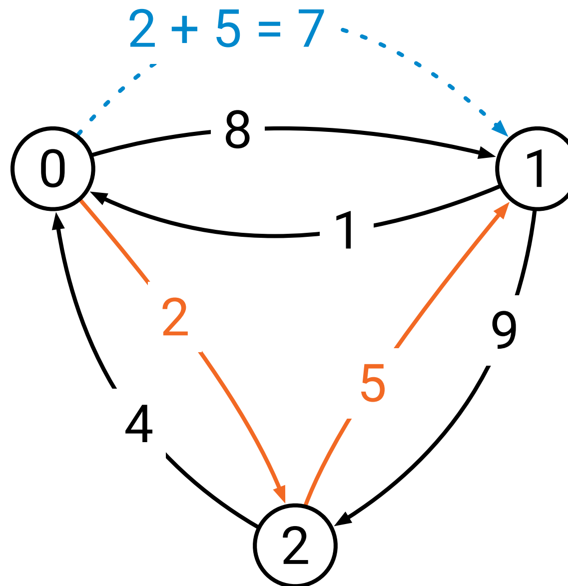
How would we like to use registers?

- If your code only reads each input value once,
  there isn't much you can do
  - the number of memory accesses is already as small as possible
  - no room for reusing data in registers
- But if you need to ***use the same input value many times***,
  try to design your algorithm so that you:
  - ***read it once***
  - keep it in registers for a while
  - ***use it many times for something useful***

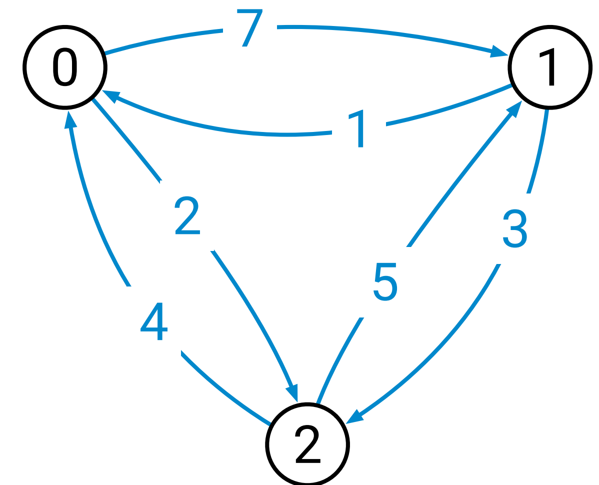
Sample application: cheapest 2-hop path

d (input):





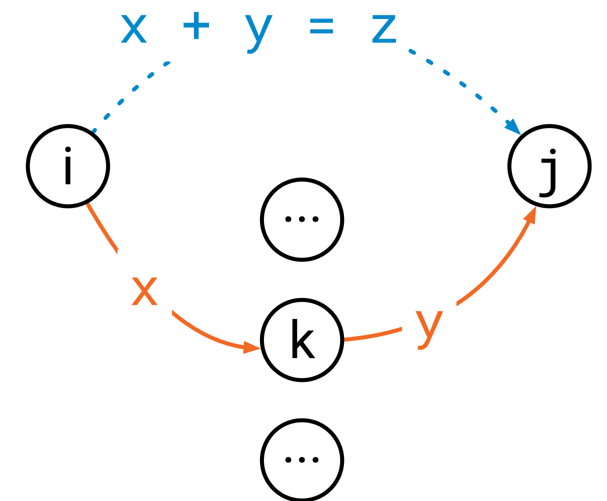
r (output):




```
d[] = { 0, 8, 2,
        1, 0, 9,
        4, 5, 0 }
```

```
r[] = { 0, 7, 2,
        1, 0, 3,
        4, 5, 0 }
```

```
void step(float* r, const float* d, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            float v = infinity;
            for (int k = 0; k < n; ++k) {
                float x = d[n*i + k];
                float y = d[n*k + j];
                float z = x + y;
                v = min(v, z);
            }
            r[n*i + j] = v;
        }
    }
}
```



2

k

?

6

Cheapest 2-hop path from 2 to 6?

d (input):

	0	1	2	3	4	5	**6**	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	**6**	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2

6

d (input):

0 1 2 3 4 5 6 7 8 9

r (output):

0 1 2 3 4 5 6 7 8 9

k

0 1 2 3 4 5 6 7 8 9

2

6

d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2 → 0 → 6

d (input):

	0	1	2	3	4	5	**6**	7	8	9
0							■			
1										
2	■									
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	**6**	7	8	9
0										
1										
2							■			
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2 → 1

6

d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0
1
2
3
4
5
6
7
8
9

2

6

d (input):

	0	1	2	3	4	5	**6**	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	**6**	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0
1
2
3
4
5
6
7
8
9

2

6

d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2 → 2 → 6

d (input):

r (output):

k

0
1
2
3
4
5
6
7
8
9

2

6

d (input):

	0	1	2	3	4	5	**6**	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	**6**	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0
1
2
3
4
5
6
7
8
9

2
6

d (input):

0 1 2 3 4 5 6 7 8 9

0
1
2
3
4
5
6
7
8
9

r (output):

0 1 2 3 4 5 6 7 8 9

0
1
2
3
4
5
6
7
8
9

k

0 1 2 3 4 5 6 7 8 9

2

6

d (input):

0 1 2 3 4 5 **6** 7 8 9

0 1 **2** 3 4 5 6 7 8 9

r (output):

0 1 2 3 4 5 **6** 7 8 9

0 1 **2** 3 4 5 6 7 8 9

k

0
1
2
3
4
5
6
7
8
9

2

6
7

d (input):

	0	1	2	3	4	5	6	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

2

0
1
2
3
4
5
6
7
8
9

6
7

d (input):

	0	1	2	3	4	5	6	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2

6
7

d (input):

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

r (output):

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

k

0
1
2
3
4
5
6
7
8
9

2

6
7

d (input):

0 1 2 3 4 5 6 7 8 9

0
1
2
3
4
5
6
7
8
9

r (output):

0 1 2 3 4 5 6 7 8 9

0
1
2
3
4
5
6
7
8
9

k

d (input):

r (output):

k

0
1
2
3
4
5
6
7
8
9

2

6
7

d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

d (input):

r (output):

k

d (input):

r (output):

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

r (output):

k

d (input):

r (output):

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

r (output):

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

0 1 2 3 4 5 6 7 8 9

r (output):

0 1 2 3 4 5 6 7 8 9

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

	0	1	2	3	4	5	**6**	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	**6**	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

0 1 2 3 4 5 6 7 8 9

r (output):

0 1 2 3 4 5 6 7 8 9

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

0 1 2 3 4 5 6 7 8 9

r (output):

0 1 2 3 4 5 6 7 8 9

k

0 1 2 3 4 5 6 7 8 9

2 3

6 7

d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

0 1 2 3 4 5 6 7 8 9

2 3

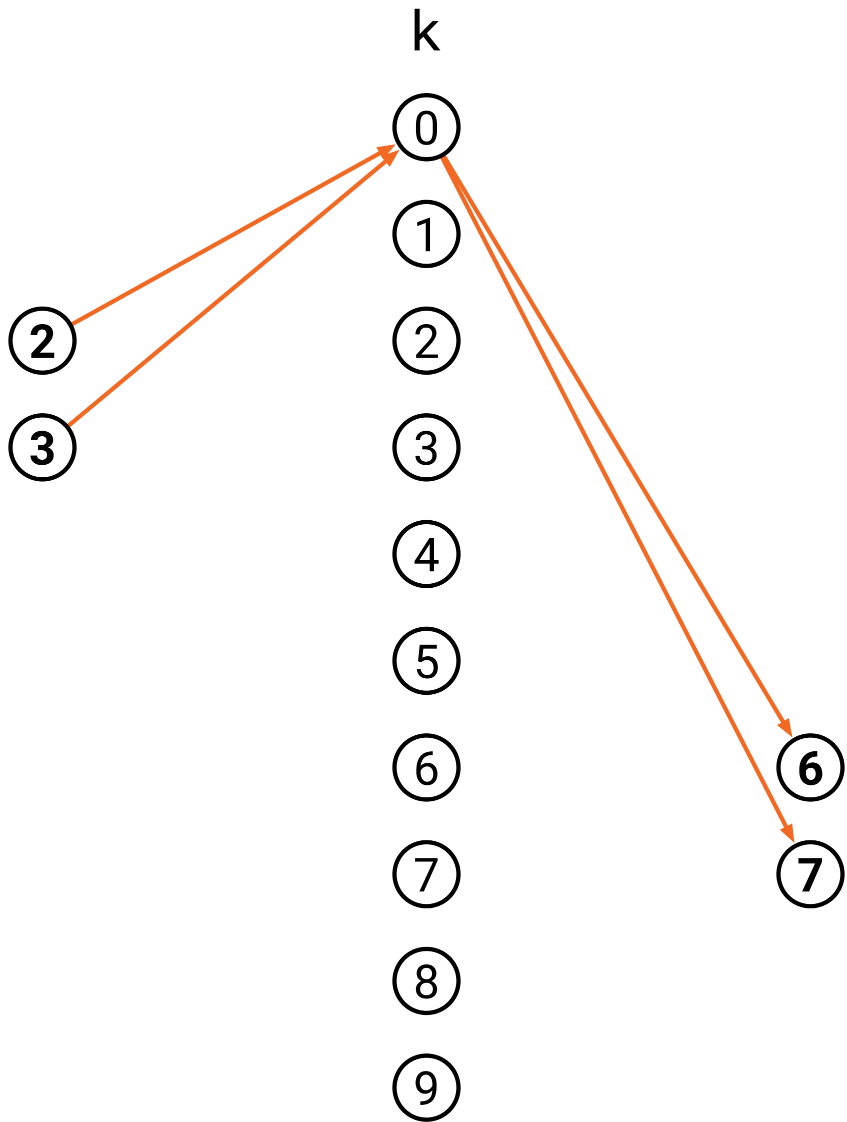
6 7

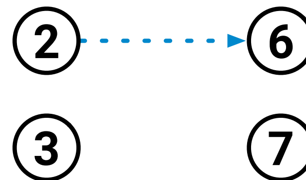
d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

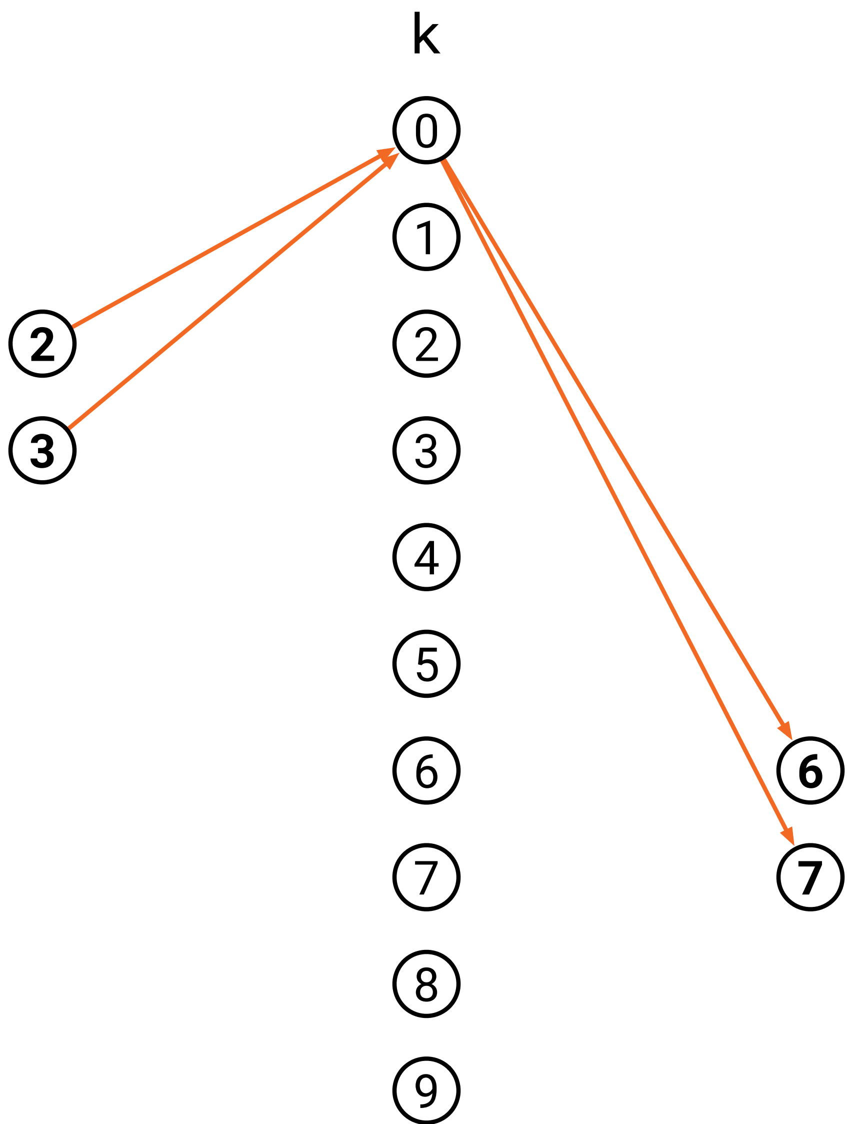
	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										





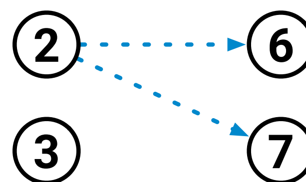
d (input):

r (output):



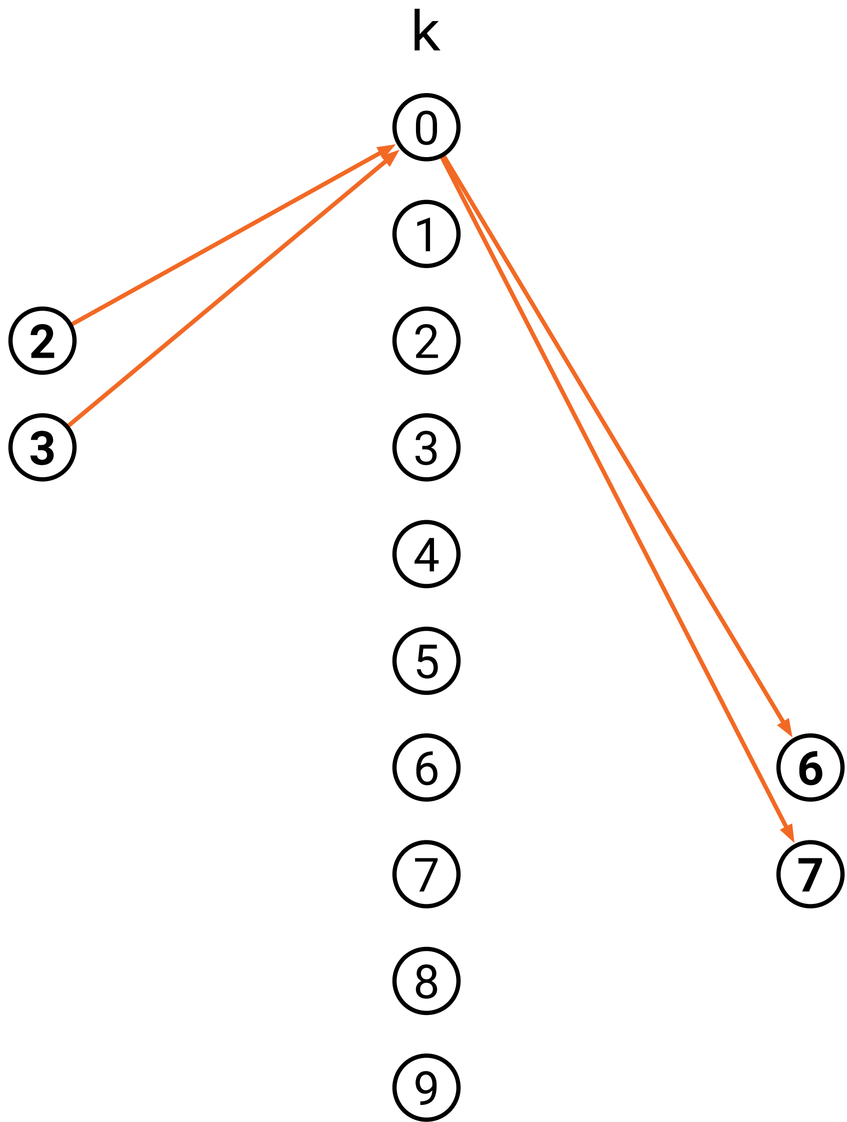
d (input):

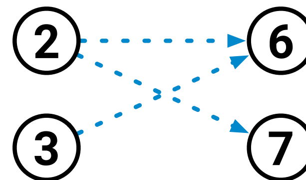
	0	1	2	3	4	5	**6**	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										



r (output):

	0	1	2	3	4	5	**6**	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										



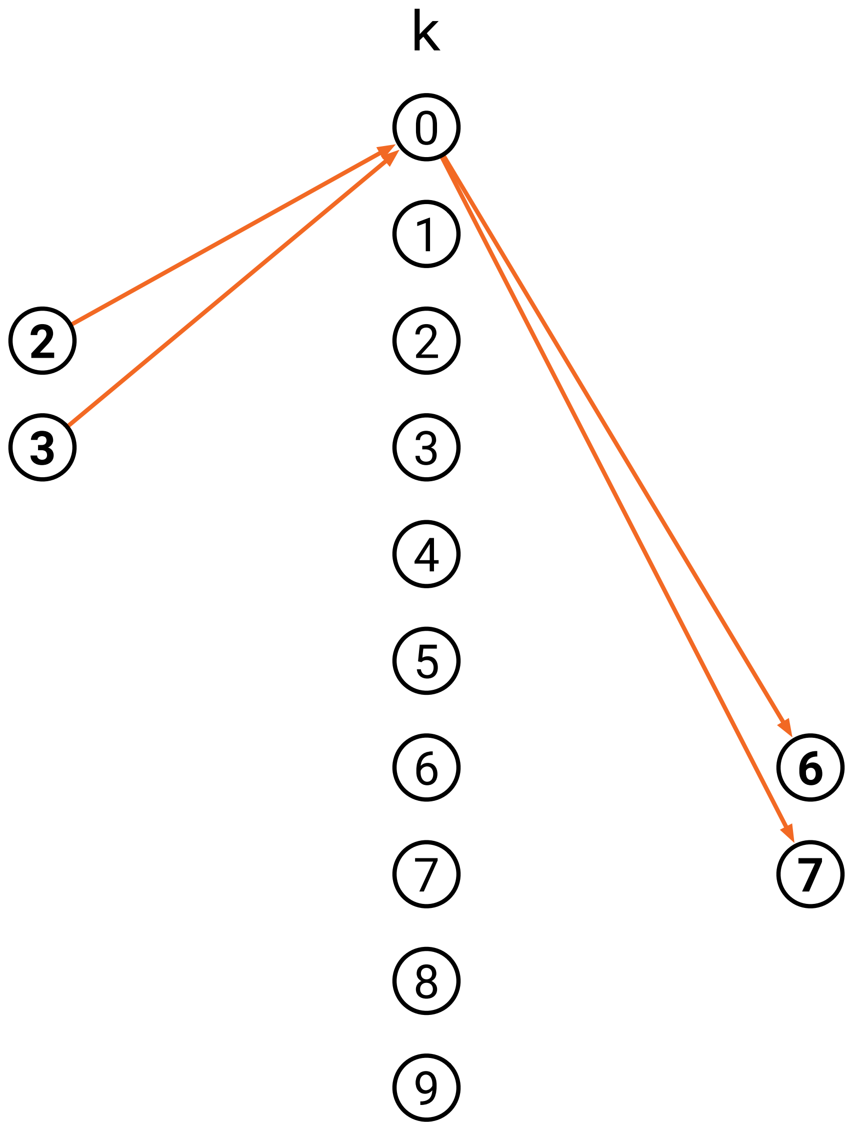


d (input):

	0	1	2	3	4	5	**6**	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	**6**	**7**	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

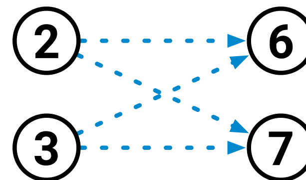


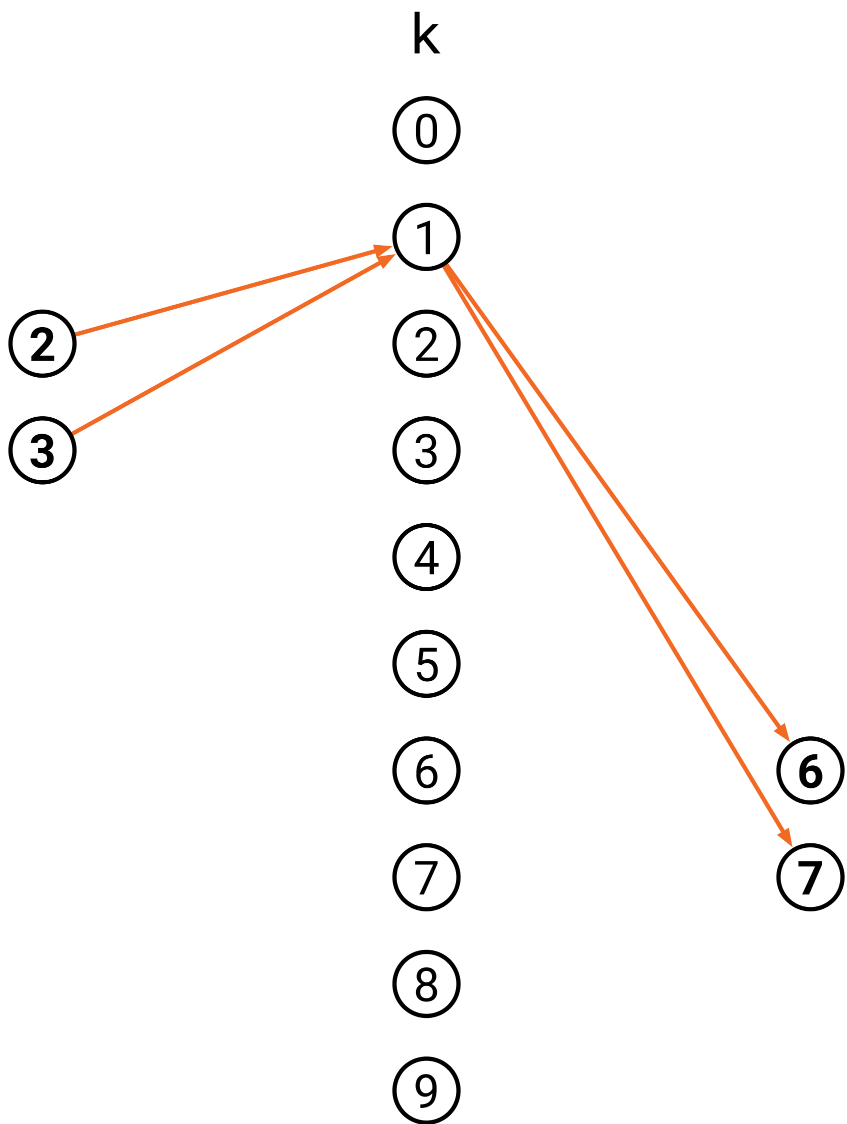
d (input):

0 1 2 3 4 5 6 7 8 9

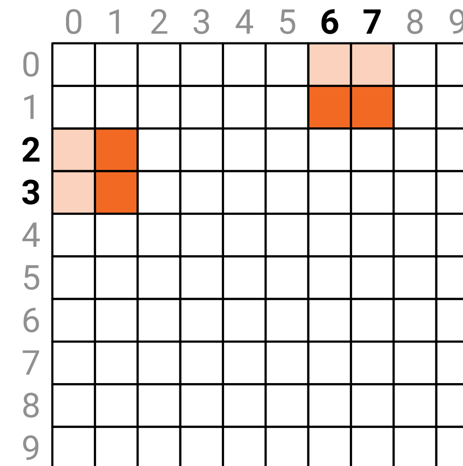
r (output):

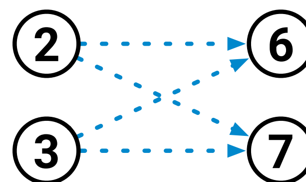
0 1 2 3 4 5 6 7 8 9



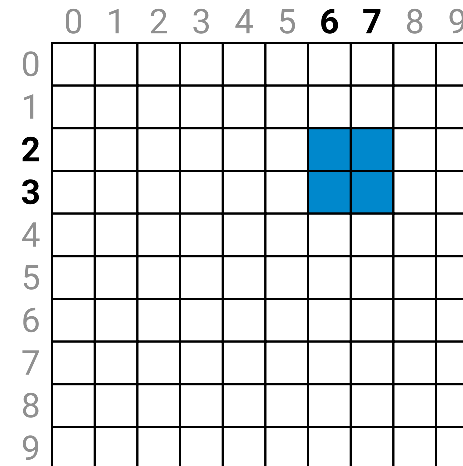


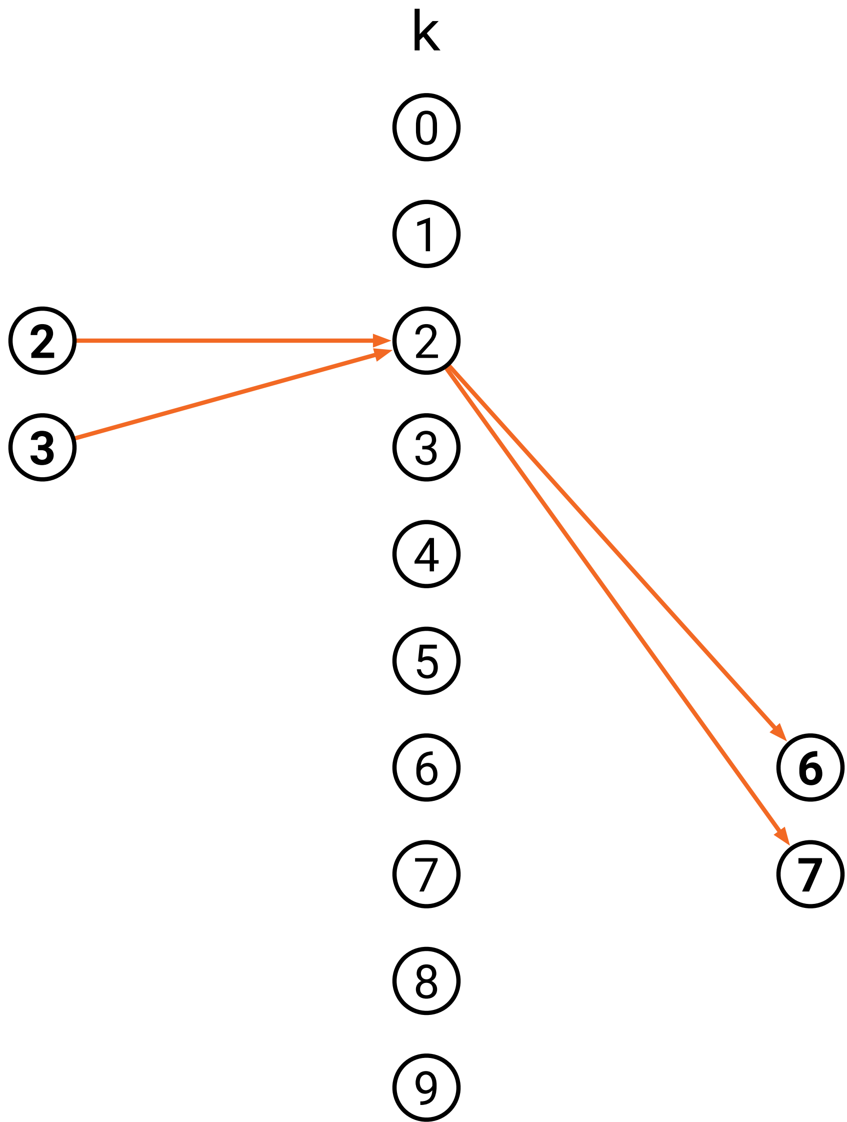
d (input):



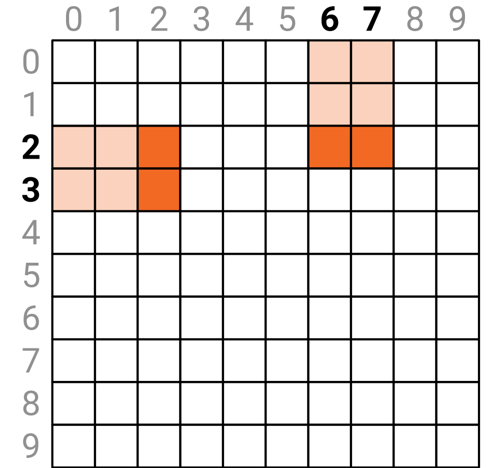


r (output):

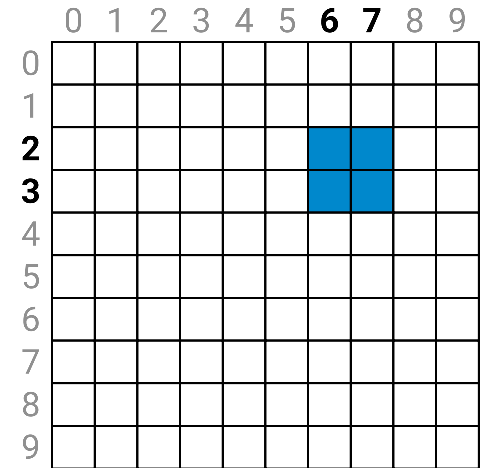


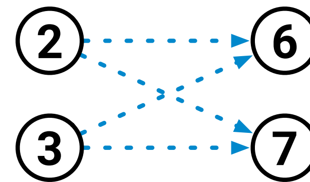


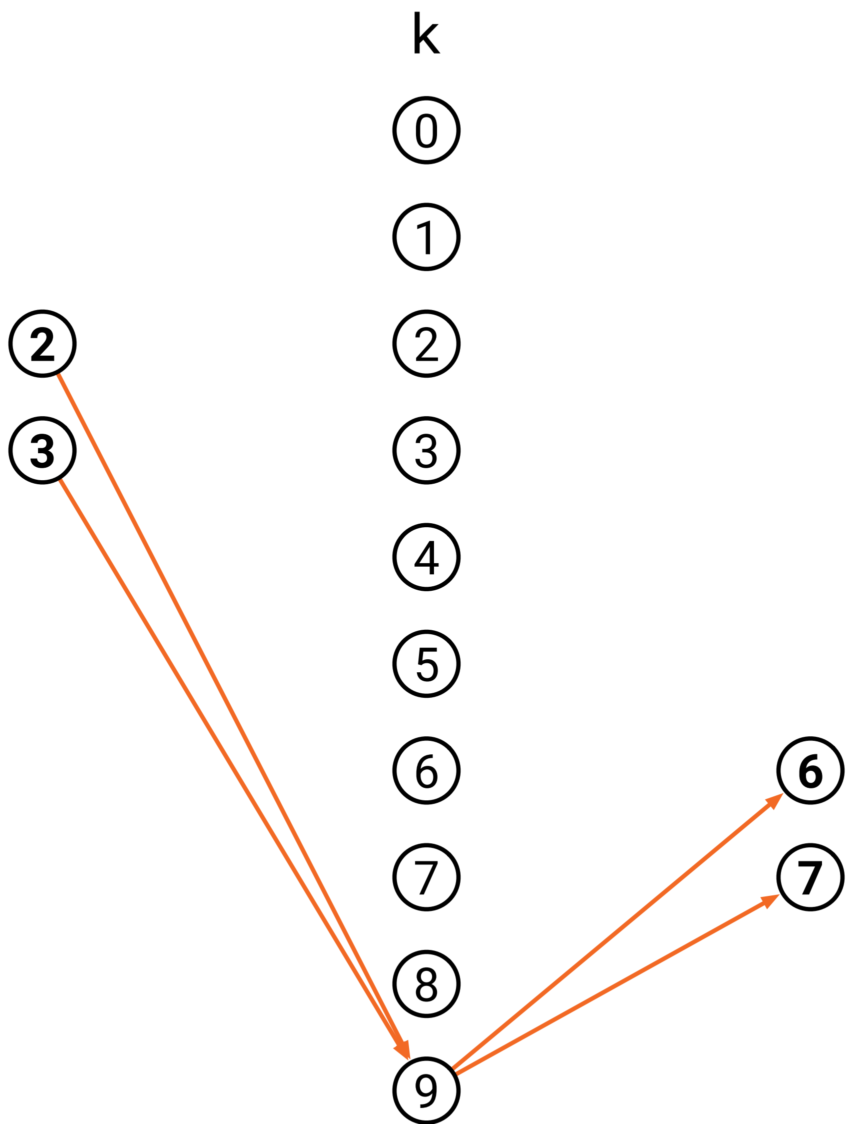
d (input):



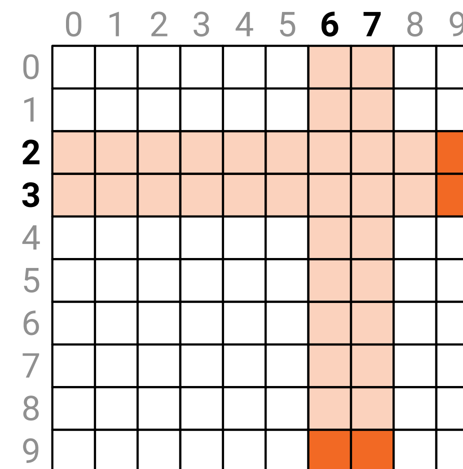
r (output):

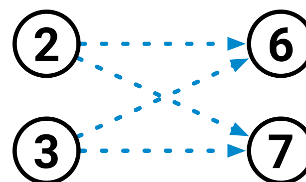




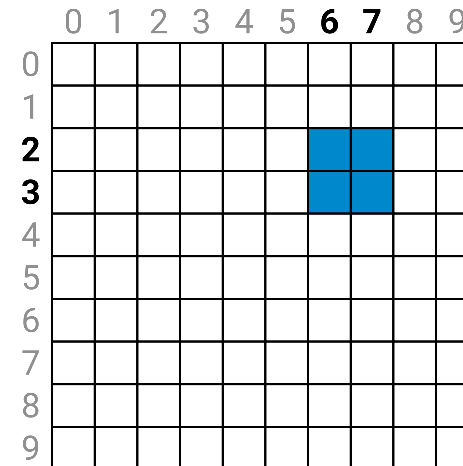


d (input):





r (output):



k

0
1
2
3
4
5
6
7
8
9

2
3

6
7

d (input):

0 1 2 3 4 5 **6** **7** 8 9

0
1
2
3
4
5
6
7
8
9

r (output):

0 1 2 3 4 5 **6** **7** 8 9

0
1
2
3
4
5
6
7
8
9

k

0 1 2 3 4 5 6 7 8 9

0 1

4 5

d (input):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

k

d (input):

r (output):

k

0 1 2 3 4 5 6 7 8 9

0 1 2

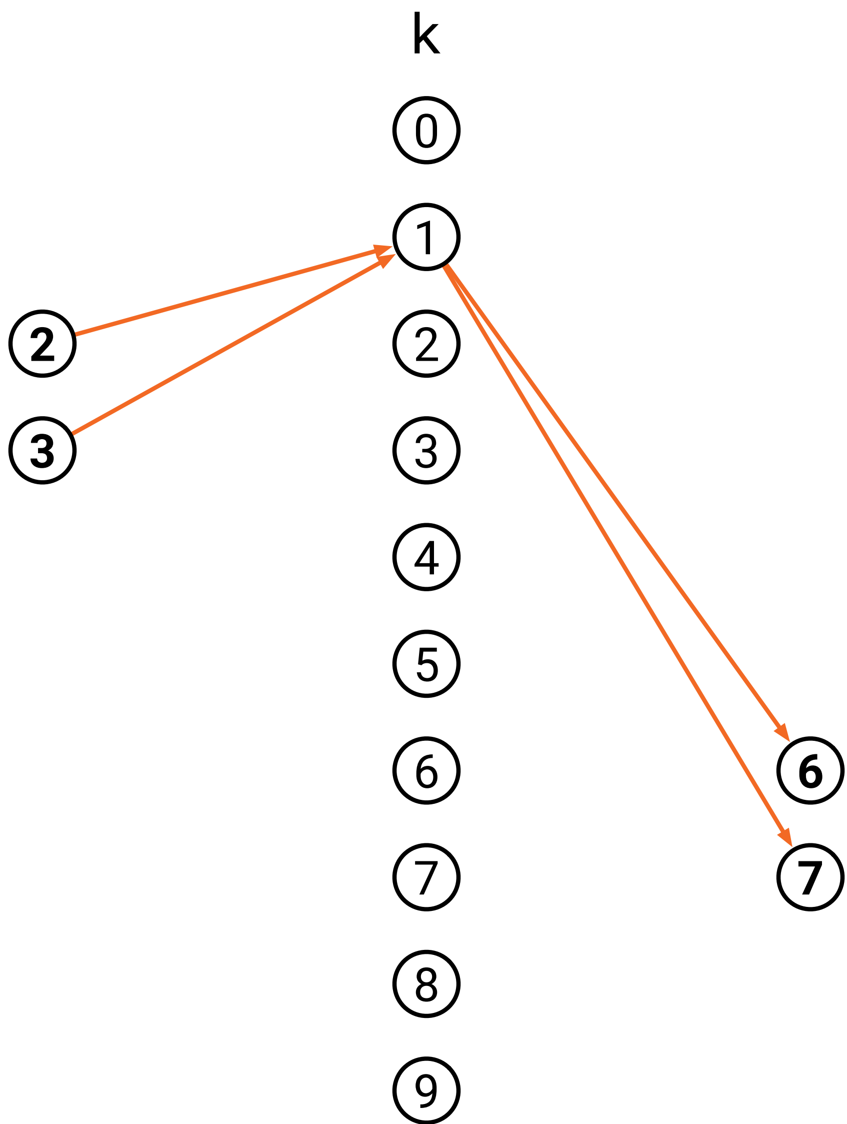
3 4 5

d (input):

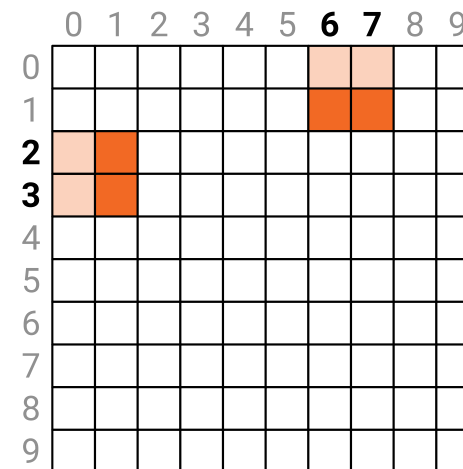
	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

r (output):

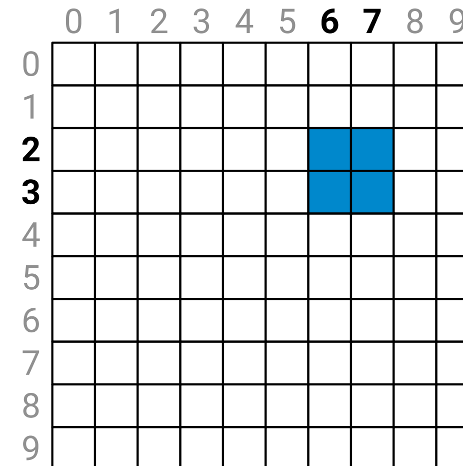
	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

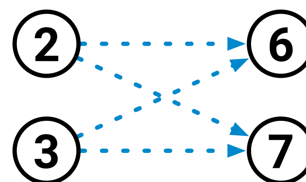


d (input):



r (output):





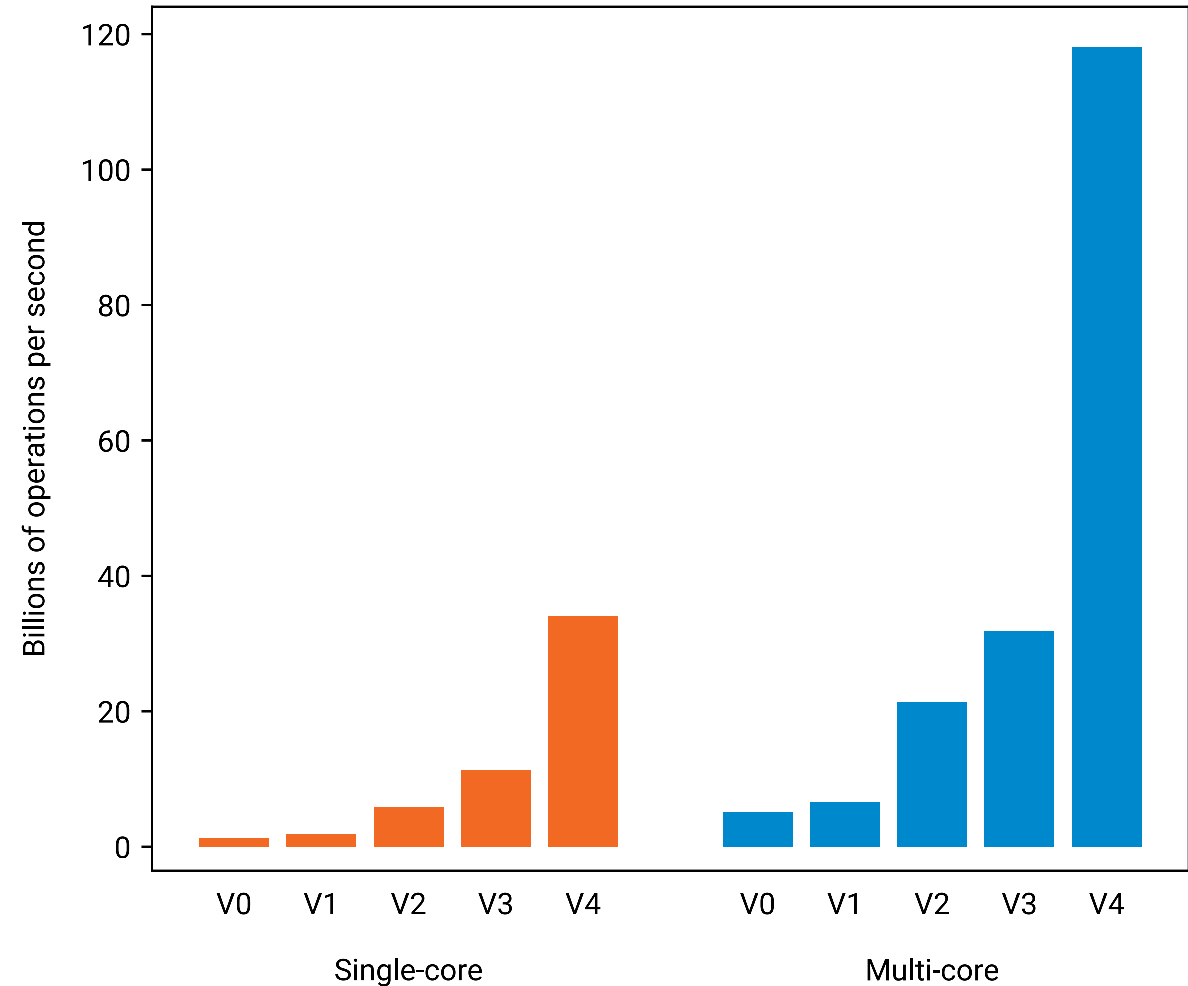
Reuse data in registers

V2: instruction-level parallelism

V3: vectorization

V4: reuse data

Running time improved from **99 s** to **1 s**



Reuse data in registers

V2: instruction-level parallelism

V3: vectorization

V4: reuse data

V5: more data reuse…