NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Programming Techniques for Supercomputers Tutorial

Erlangen National High Performance Computing Center

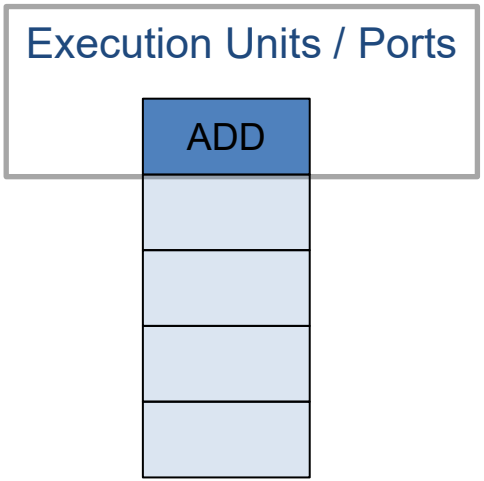
Department of Computer Science

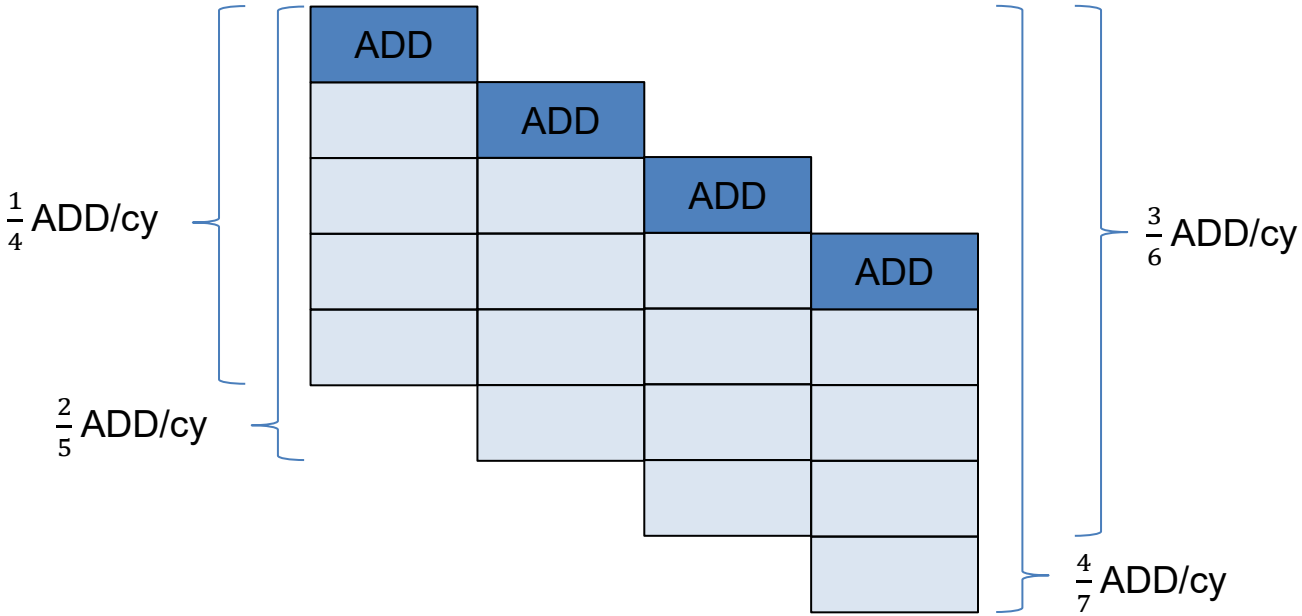
FAU Erlangen-Nürnberg

Sommersemester 2026

Assignment 1 – Task 1

Pipelines





- 4 cy latency
- 1 result per cy

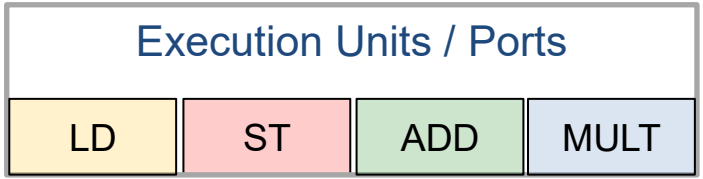
$\boldsymbol{TP_{pipe}} = \frac{\boldsymbol{N}}{\boldsymbol{N+m-1}}$ with $TP_{pipe} = 0.9$ inst/cy and $m = 4$:

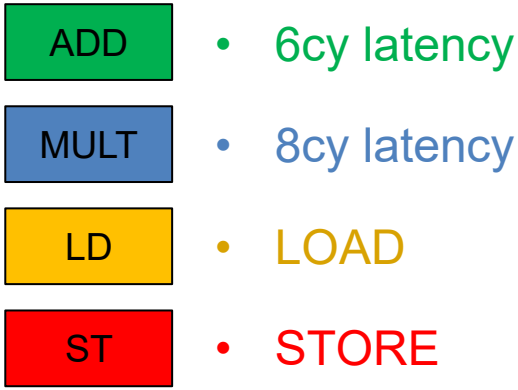
$0.9(\mathrm{N} + 3) = \mathrm{N}$ ➔ $2.7 = 0.1\mathrm{N}$ ➔ $\boldsymbol{N = 27}$ **with two pipelines: N=54**

Assignment 1 – Task 2

More pipelines a)

```
double a[...];
double s=0.1;
// a[], b[], c[] contain sensible data
for(int i=2; i<N; ++i) {
    a[i] += s * a[i-2];
}
for(int i=2; i<N; ++i) {
    b[i] += s * b[i-2];
}
for(int i=2; i<N; ++i) {
    c[i] += s * c[i-2];
}
```

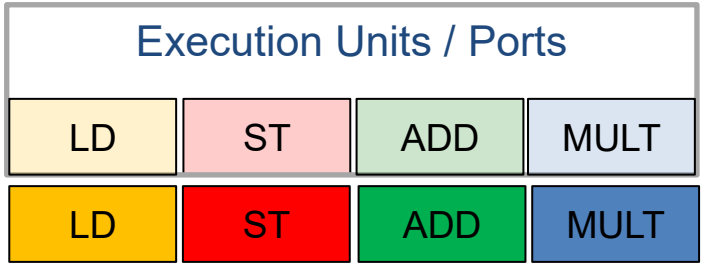




Assignment 1 – Task 2

More pipelines a)

```
double a[...];
double s=0.1;
// a[], b[], c[] contain sensible data
for(int i=2; i<N; ++i) {
  a[i] += s * a[i-2];
}
//...
```



- 1 ADD, 6cy latency
- 1 MULT, 8cy latency
- 1 LOAD
- 1 STORE

Does this mean we can do 1 iter. per cycle (i.e., 2 flops in 1 cy)?

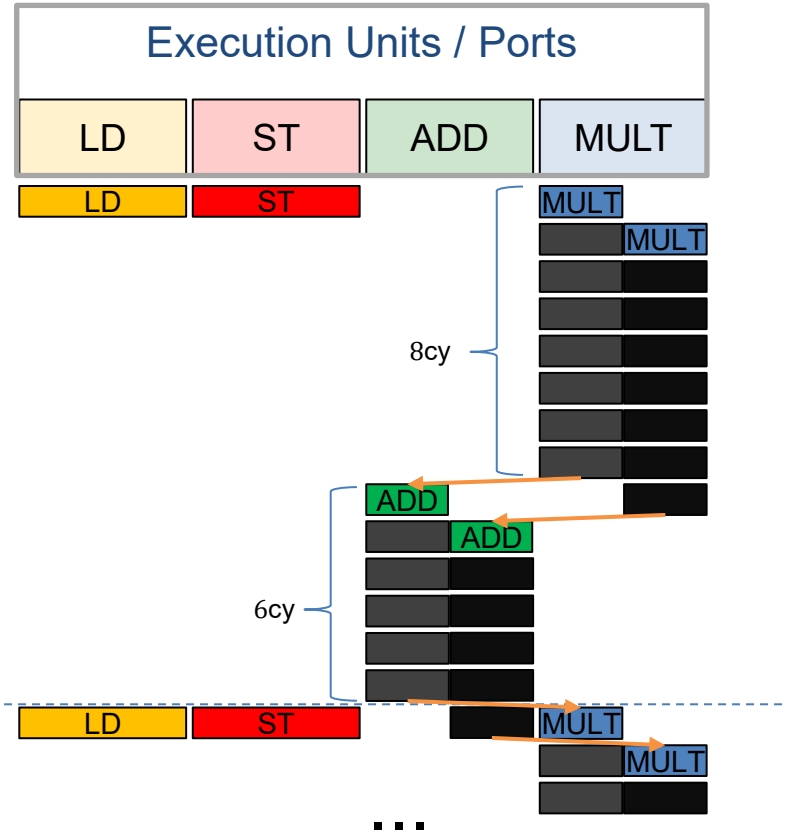
NO

Remember: Pipelining and dependencies from lecture (slide set 3a)

Assignment 1 – Task 2

More pipelines a)

```
double a[...];
double s=0.1;
// a[], b[], c[] contain sensible data
for(int i=2; i<N; ++i) {
   a[i] += s * a[i-2];
}
//...
```



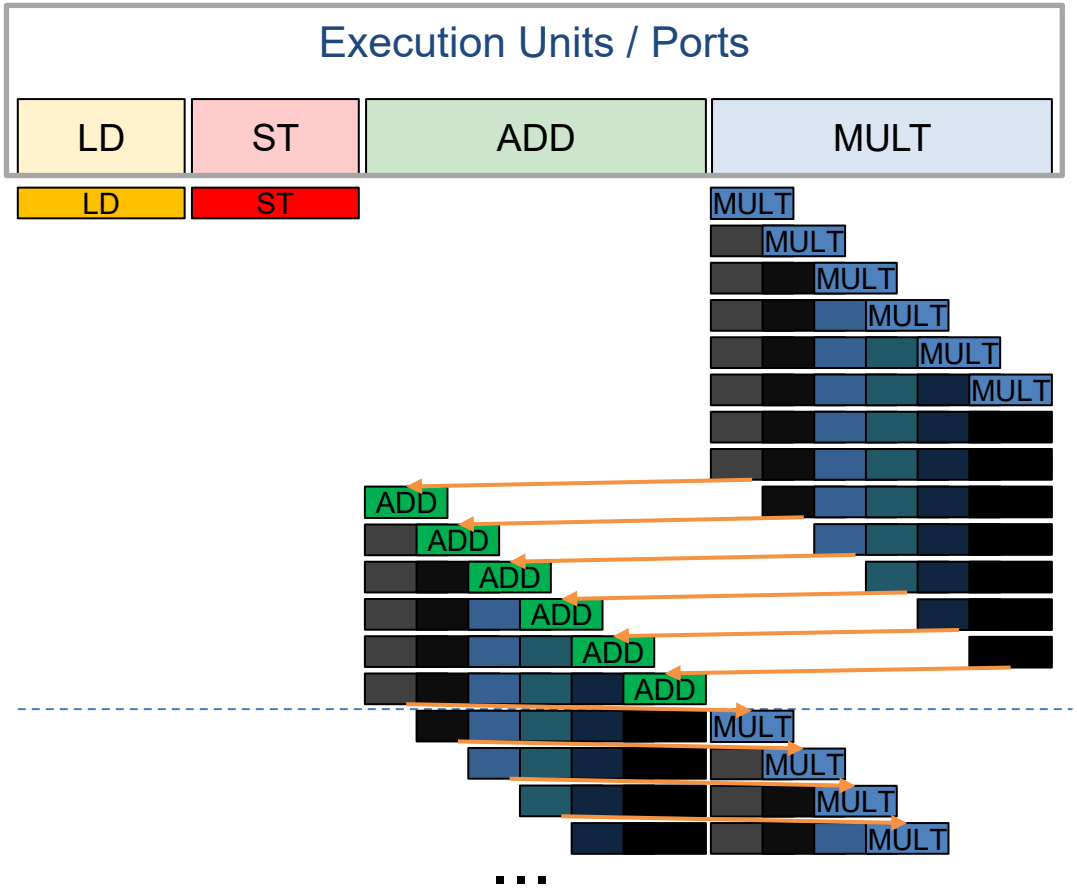
Bottleneck: ADD/MUL pipeline stall (latency)

➔ P_{max} = 2 iter/14 cy = 4 flops/ 14 cy
= 0.286 flops/cy

Assignment 1 – Task 2

More pipelines b)

```
double a[...];
double s=0.1;
// a[], b[], c[] contain sensible data
for(int i=2; i<N; ++i) {
    a[i] += s * a[i-2];
    b[i] += s * b[i-2];
    c[i] += s * c[i-2];
}
```



6 instead of 2 independent loop-carried dependencies!

➔ P_{max} = 6 iter/14 cy = 12 flops/ 14 cy
= 0.857 flops/cy

Assignment 1 – Task 3

Square root in depth a)

```
sum = 0.;
for(int i=0; i<N; i++) {
  x = (i + 0.5) * delta_x;
  sum += 4.0 * sqrt(1.0 - x*x);
}
```

```
sum = 0.; sum1 = 0.; sum2 = 0.;
for(int i=0; i<N; i+=2) {
  x1 = (i + 0.5) * delta_x;
  x2 = (i+1+0.5) * delta_x;
  sum1 += 4.0 * sqrt(1.0 - x1*x1);
  sum2 += 4.0 * sqrt(1.0 - x2*x2);
}
sum = sum1 + sum2;
```

- Bottleneck: SQRT with 22cy latency and 3cy reciprocal throughput

- BUT: Modulo Variable Expansion (MVE) can be beneficial
  → **different order of accumulation** → SQRTs can overlap → hide latency
- **partial sums must be reduced to one after the loop**

Assignment 1 – Task 3

Square root in depth b)

Measurement (N=10^9): 6.02cy

```
sum = 0.;
for(int i=0; i<N; i++) {
  x = (i + 0.5) * delta_x;
  sum += 4.0 * sqrt(1.0 - x*x);
}
```

```
.LBB1_7:
  vaddsd      %xmm5, %xmm2, %xmm3
  vmulsd      %xmm0, %xmm3, %xmm3
  vfnmadd213sd %xmm4, %xmm3, %xmm3
  vsqrtsd     %xmm3, %xmm3, %xmm3
  vfmadd231sd %xmm6, %xmm3, %xmm1
  vaddsd      %xmm4, %xmm2, %xmm2
  decl        %eax
  jne         .LBB1_7
```

- int2float
- 3 ADDs
- 3 MULTs
- 1 SQRT
- 3 instr. loop mechanics

IPC = 11 instr/6 cy
= 1.83 instr/cy

IPC = 8 instr/6 cy
= 1.33 instr/cy