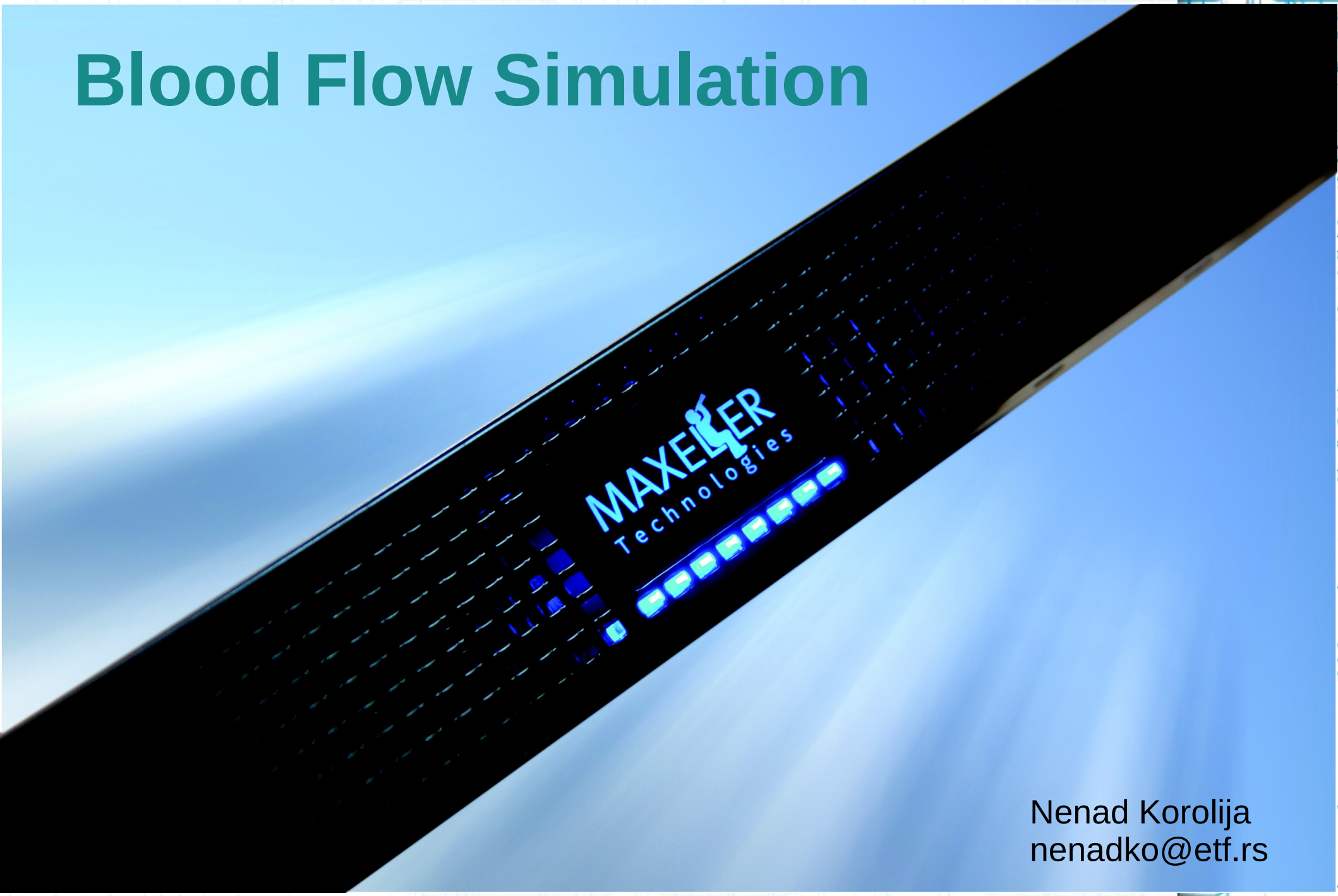
Blood Flow Simulation



Nenad Korolija
nenadko@etf.rs

Problems

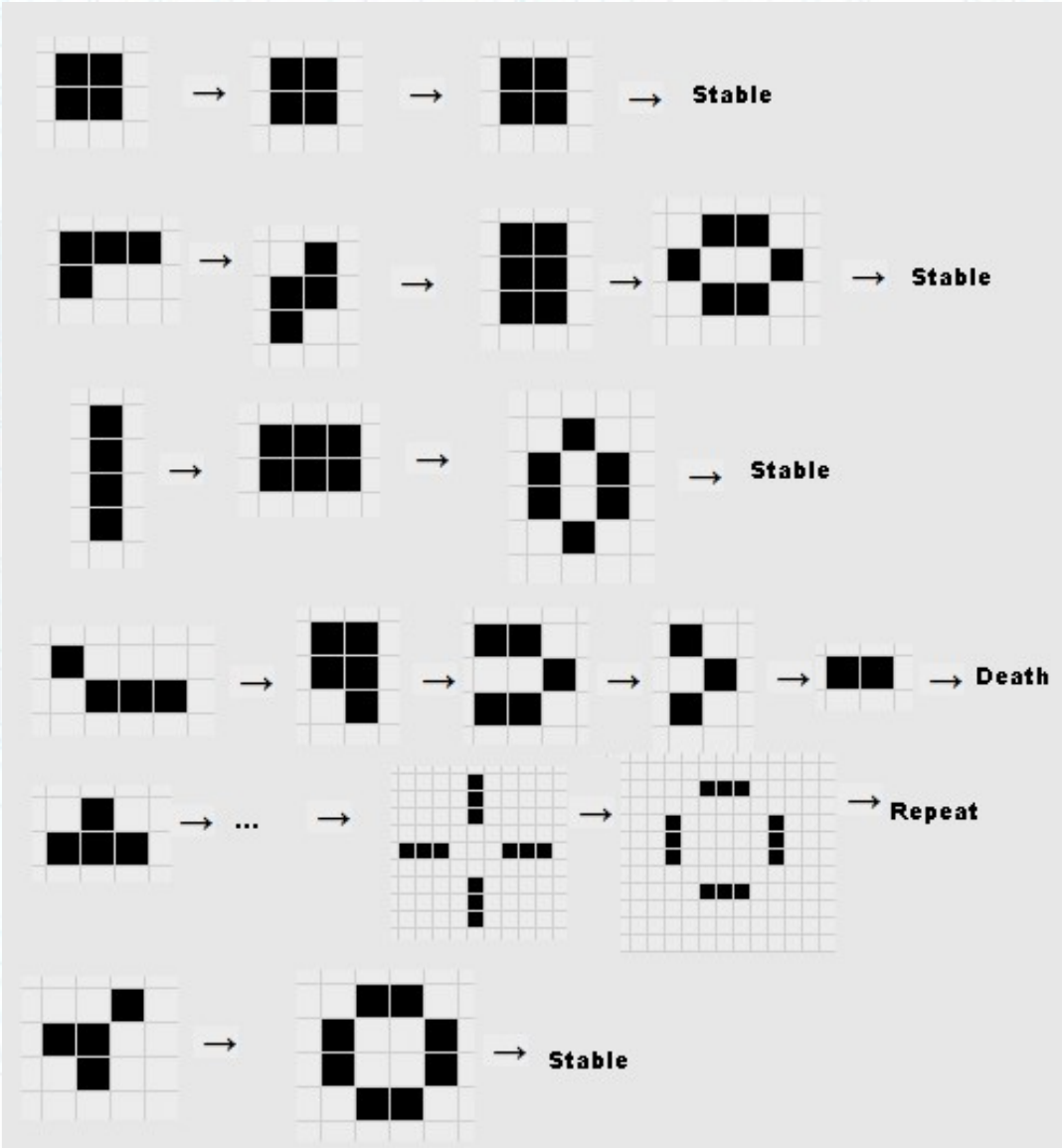
- Bottleneck: 10000 iterations
- In each iteration,
  16 matricies are re-calculated
- Dependencies between
  each two iterations in a row
- Each element of matrix
  depends on surrounding ellements
  belonging to other matricies.

Related to: Conway's Game of Life

Cellular automaton
devised by the British mathematician
John Horton Conway in 1970

- Any live cell with fewer than two live neighbours dies, as if caused by under-population.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overcrowding.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Conway's Game of Life



Matrices Calculation

- Each matrice size: 320 x 112
  Rows: 0<=j<320
  Columns: 0<=i<112
- Matrices stored in arrays

```
for iter=0..maxIter
  stream();
  per_BC();
  in_BC();
  ex_BC_crude();
  collide();
```

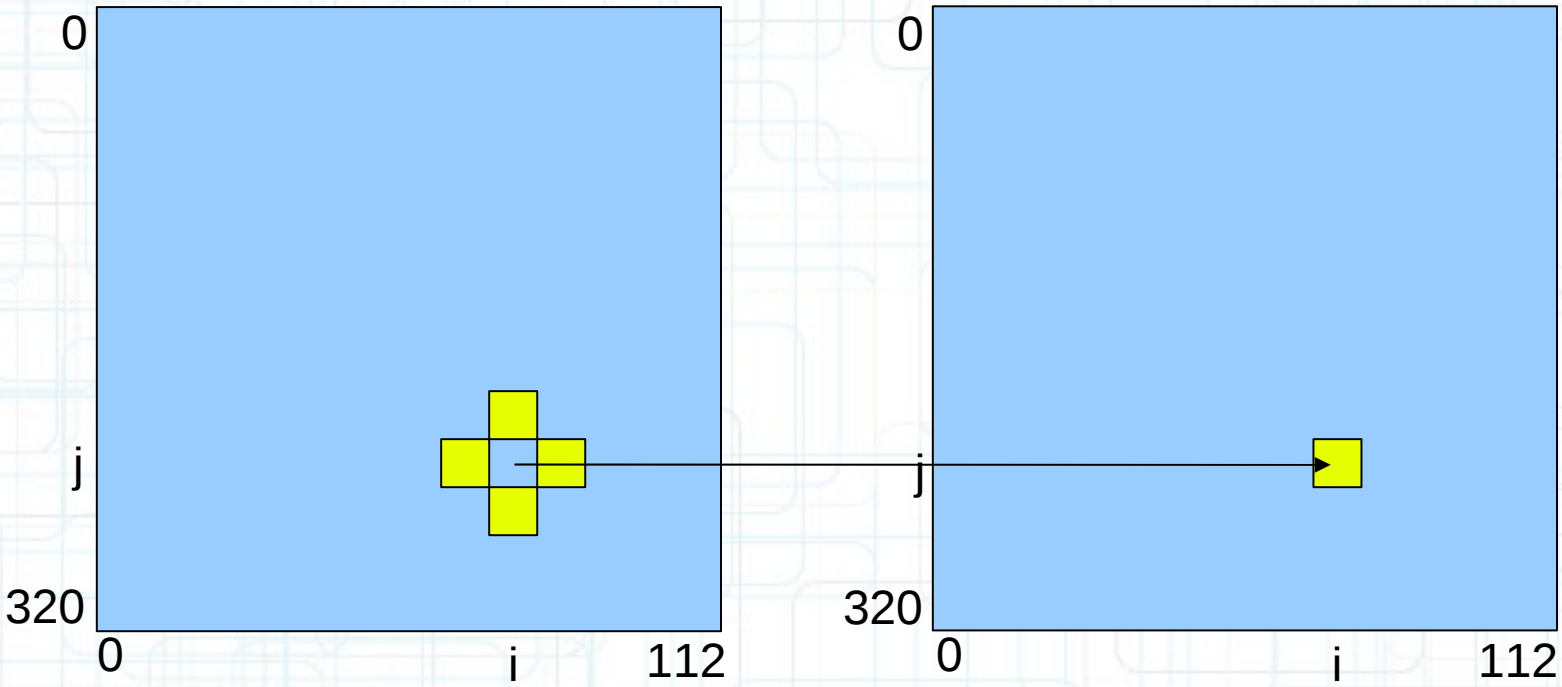
Stream

- Calculating new values for matrices.

```
for j=0...nj
        for i=1...ni
            tmpf1[I2D(ni,i,j)] = f1[I2D(ni,i-1,j)];
```

Stream

1) stream();
Matrix tmpfi[a,b] = f[c,d], 1<=i<=8



2) Copy:
Matrix tmpfi = fi, 1<=i<=8

Non Critical-Code

3) per_BC();

```
for (i=0; i<ni; i++){
        i0 = i;
        i1 = ni*(nj-1)+i;
        f2[i0] = f2[i1];
```

4) in_BC();

```
vx_term = 1.f + 3.f*vxin +3.f*vxin*vxin;
f1new = roout * faceq2 * vx_term;
for (j=0; j<nj; j++){
        i0 = ni*j;
        f1[i0] = f1new;
```

5) ex_BC_crude();

```
for (j=0; j<nj; j++){
        i0 = ni*j+ni-1;
        i1 = i0 - 1;
        f3[i0] = f3[i1];
```

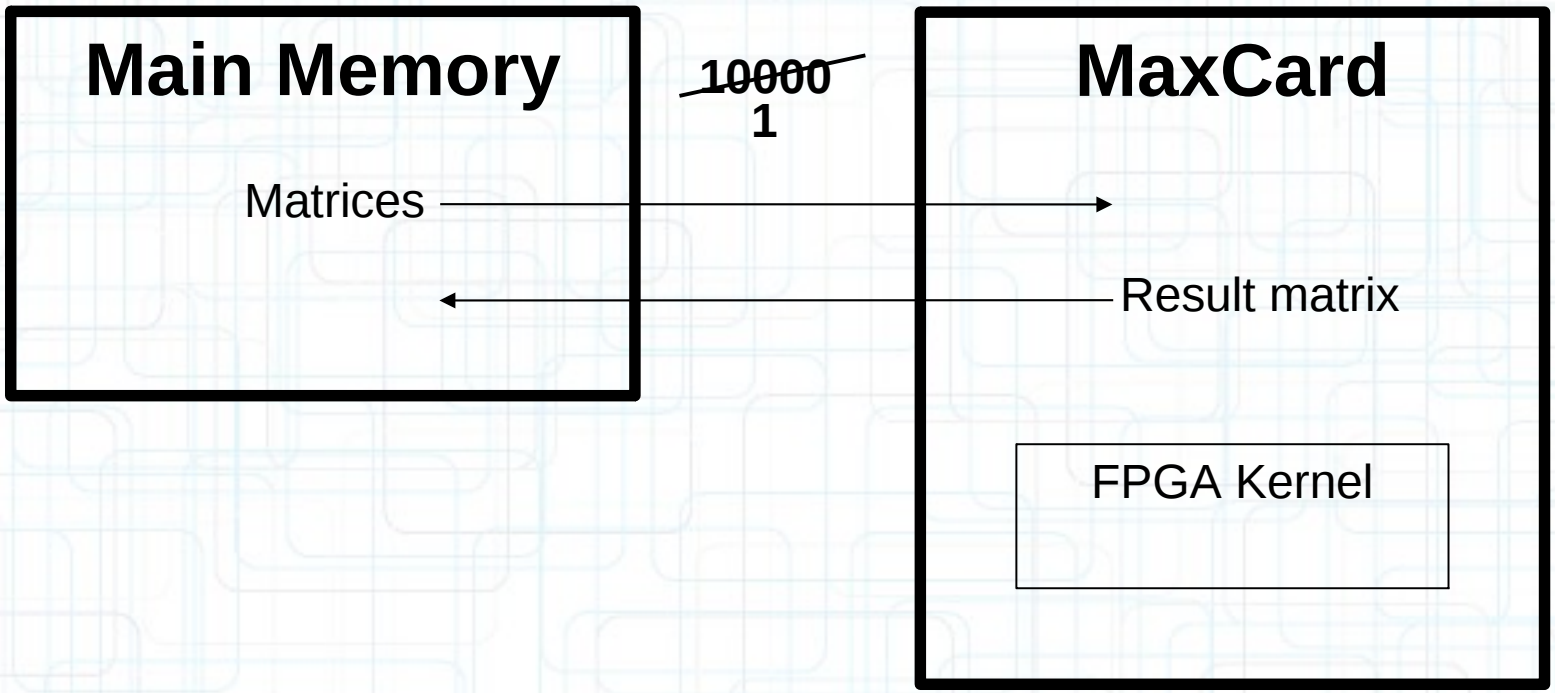
Collide

6) collide();

```
for (j=0; j<nj; j++) {
    for (i=0; i<ni; i++) {
      i0  = 320*j+i;
      // Do the summations needed to evaluate the density and components of velocity
      ro = f0[i0] + f1[i0] + f2[i0] + f3[i0] + f4[i0] + f5[i0] + f6[i0] + f7[i0] + f8[i0];
      rovx = f1[i0] - f3[i0] + f5[i0] - f6[i0] - f7[i0] + f8[i0];
      rovy = f2[i0] - f4[i0] + f5[i0] + f6[i0] - f7[i0] - f8[i0];
      vx = rovx/ro;
      vy = rovy/ro;
      …
      f1eq = ro * faceq2 * (1.f + 3.f*vx + 4.5f*vx*vx - v_sq_term);
      …
      f1[i0] = rtau1 * f1[i0] + rtau * f1eq; // f1eq depends on f0..f8
      ...
```

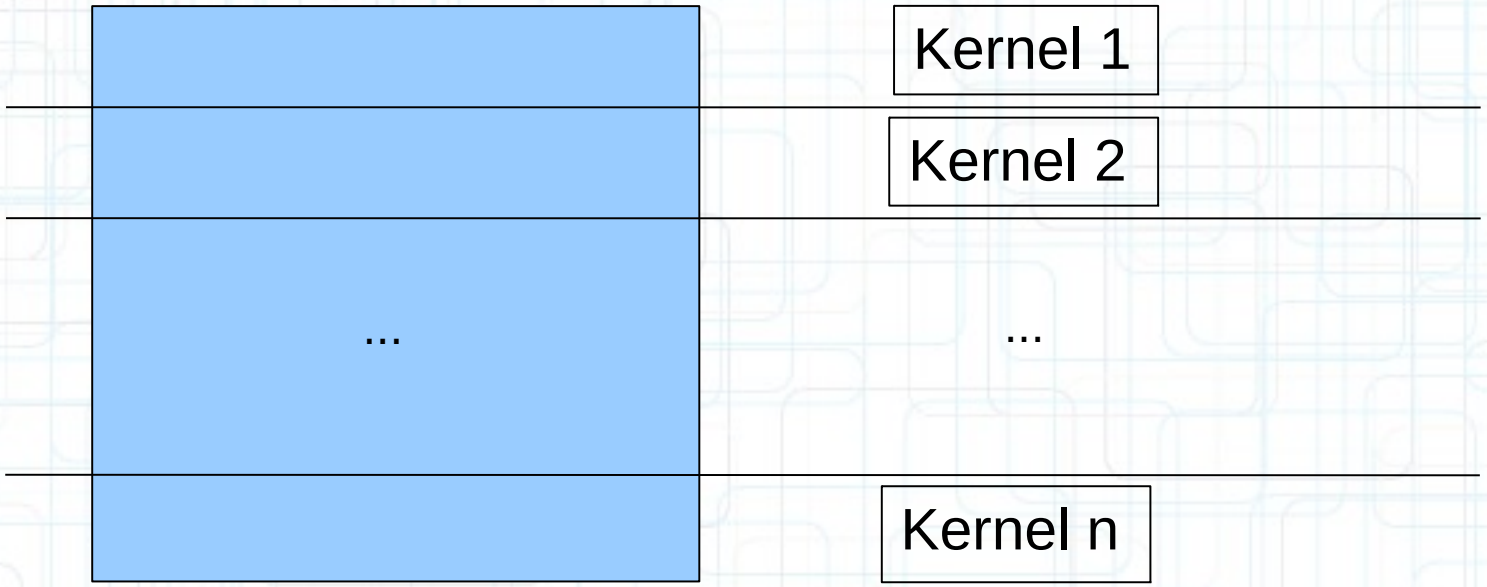
Memory Transfer

10000 iterations



Try (1)

- Kernel calculating critical code

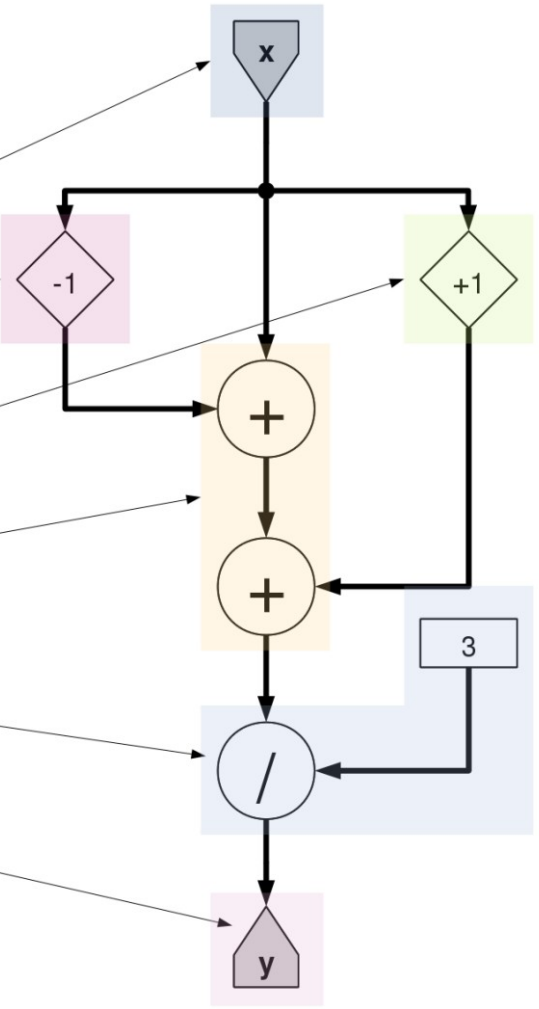


Problems

- Limited number of DSPs in MAX 2
- Stream offset
- Too much other work at ETF :)

What is Stream Offset: Moving Average in MaxCompiler?

```
 7  public class MovingAverageKernel extends Kernel {
 8
 9      public MovingAverageKernel(KernelParameters parameters, int N) {
10          super(parameters);
11
12          // Input
13          HWVar x = io.input("x", hwFloat(8, 24));
14
15          // Data
16          HWVar prev = stream.offset(x, -1);
17
18          HWVar next = stream.offset(x, 1);
19
20          HWVar sum = prev+x+next;
21
22          HWVar result = sum/3;
23
24          // Output
25          io.output("y", result, hwFloat(8, 24));
26      }
27  }
```



Compile Time too Long

SimpleKernel.java

```
tmpf1[i0] = stream.offset(f1, j320+im1);
tmpf2[i0] = stream.offset(f2, j320+jm1+i);
tmpf3[i0] = stream.offset(f3, j320+ip1);
tmpf4[i0] = stream.offset(f4, j320+jp1+i);
tmpf5[i0] = stream.offset(f5, j320+jm1+im1);
tmpf6[i0] = stream.offset(f6, j320+jm1+ip1);
tmpf7[i0] = stream.offset(f7, j320+jp1+ip1);
tmpf8[i0] = stream.offset(f8, j320+jp1+im1);
```

- Ako je odkomentarisana **tmpf2[i0]** linija, kompajliranje traje 1.05 min.

Tue 03:44: (1/3) – GenerateMaxFileDataFile
Tue 03:44: (2/3) - SimCompilePass
Tue 03:45: (3/3) - AddSimObjectToMaxFilePass
Tue 03:45: MAX file: /home/demo/Desktop/MaxCompiler-Builds/SimpleHostSim/results/SimpleHostSim.max (MD5Sum: a5510a4702048b0920ea95be30d39c6f)
Tue 03:45: Build completed: Tue Oct 23 03:45:10 PDT 2012 **(took 1 min, 5 secs)**

- Ako je odkomentarisana **tmpf3[i0]** linija, kompajliranje kernela (build-sim) traje preko 10 min.

Tue 03:50: (1/3) - GenerateMaxFileDataFile
Tue 03:50: (2/3) - SimCompilePass
Tue 03:55: (3/3) - AddSimObjectToMaxFilePass
Tue 03:56: MAX file: /home/demo/Desktop/MaxCompiler-Builds/SimpleHostSim/results/SimpleHostSim.max (MD5Sum: ce99699e7f44fcc3fba40aea9f2fa971)
Tue 03:56: Build completed: Tue Oct 23 03:56:08 PDT 2012 **(took 10 mins, 31 secs)**

- Sa odkomentarisanom linijom: **tmpf4[i0]** linija, kompajliranje traje 2 sata, 29min.

Tue 06:51: (1/3) - GenerateMaxFileDataFile
Tue 06:51: (2/3) - SimCompilePass
Tue 06:55: (3/3) - AddSimObjectToMaxFilePass
Tue 06:58: MAX file: /home/demo/Desktop/MaxCompiler-Builds/SimpleHostSim/results/SimpleHostSim.max (MD5Sum: 83207165e467e8c2374762d34596e1b6)
Tue 06:58: Build completed: Tue Oct 23 06:58:29 PDT 2012 **(took 2 hours, 59 mins, 4 secs)**

Try (2)

Loop unrolling

Problem:

- Kernel size
- Too many float multiplications demands too many DSPs.

Dataflow Graph Generation: Java Loops

What dataflow graph is generated?

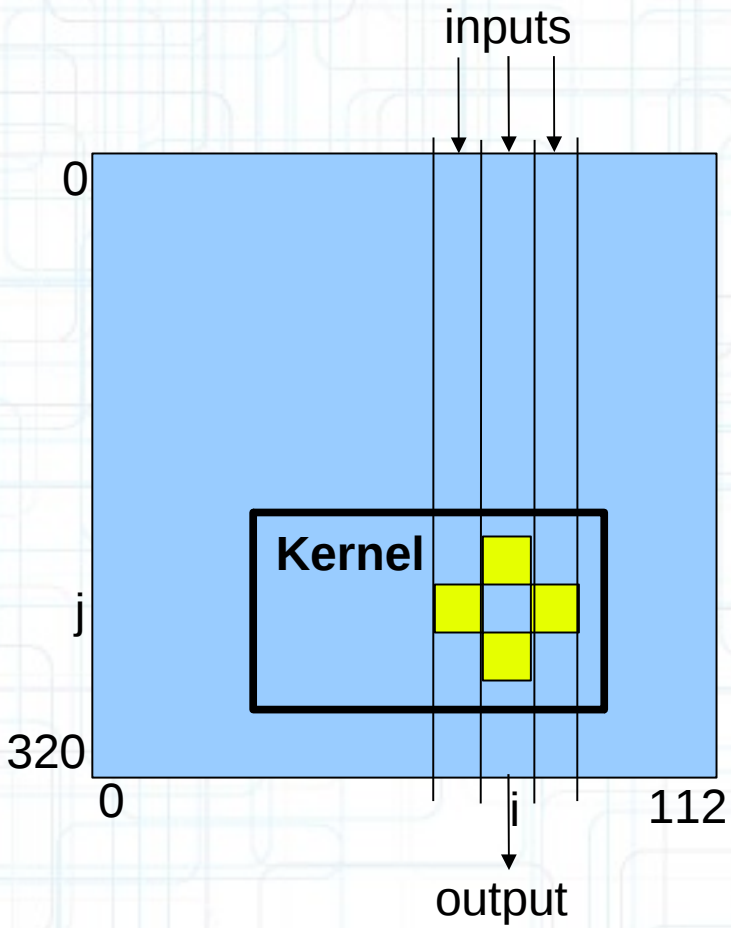
```
HWVar x = io.input(“x”, type);
HWVar y = x;
for (int i = 1; i <= 3; i++) {
    y = y + i;
}
io.output(“y”, y, type);
```

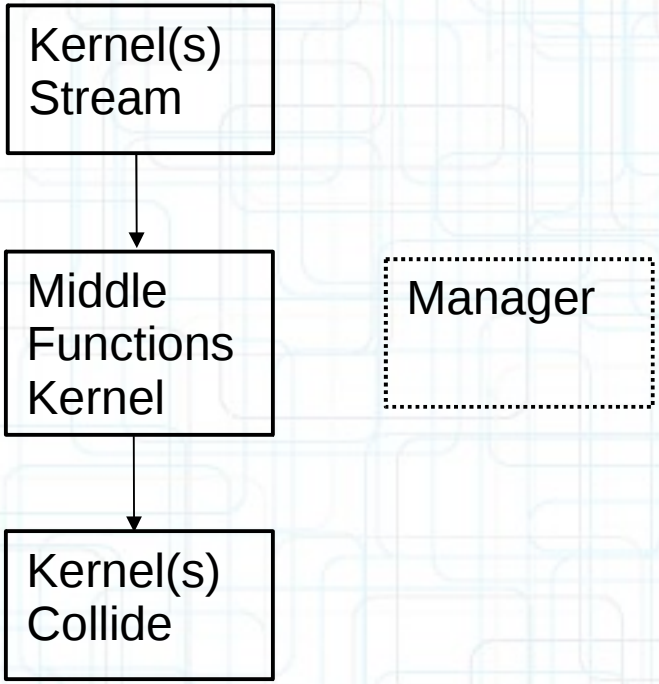
Can make the loop any size until you run out of space on the chip.

Larger loops can be partially unrolled in space and used multiple times in time.

Try (3)

- Pipelining inner loops



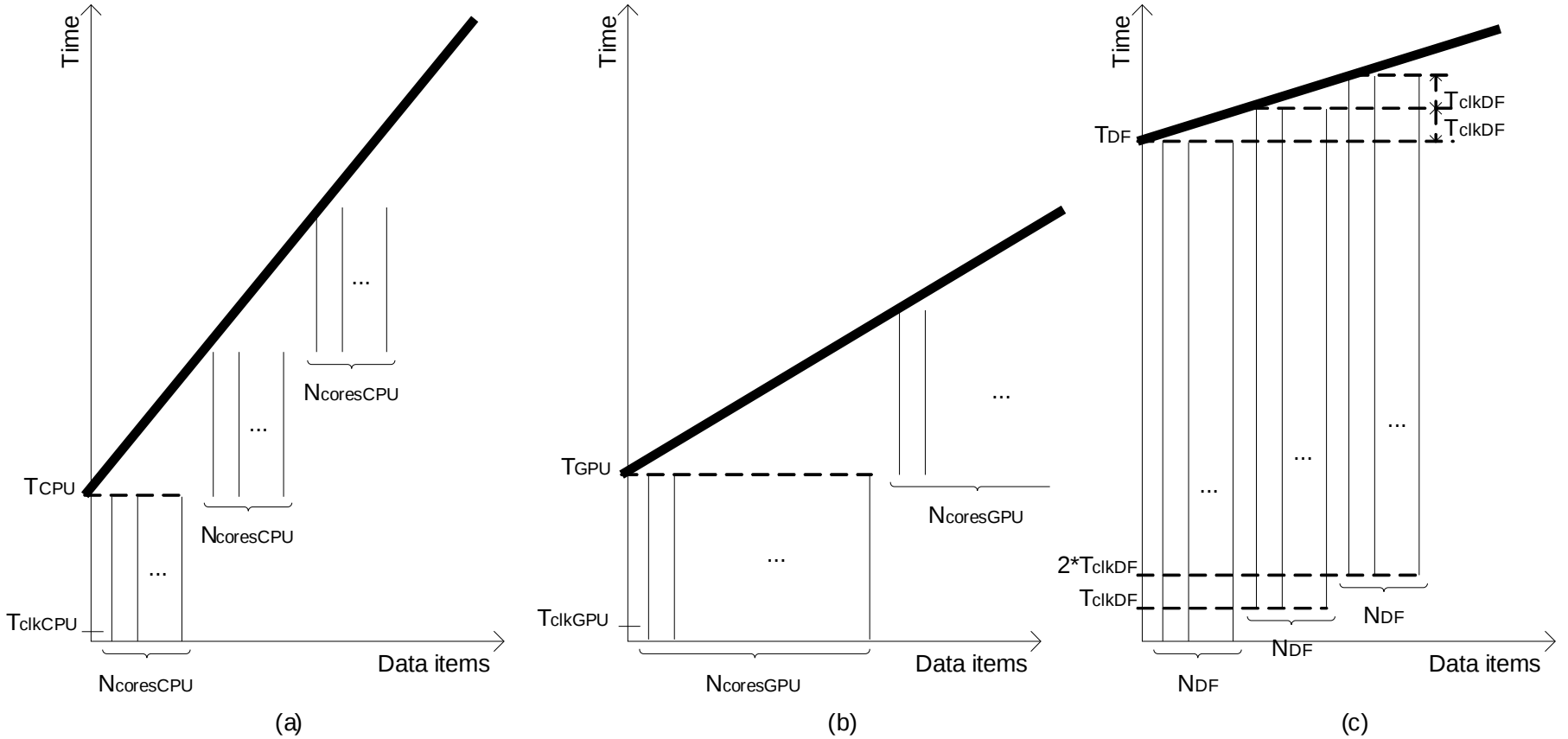


My View at the Essential Figure

$t_{CPU} = N * N_{OPS} * C_{CPU} * T_{clkCPU} / N_{coresCPU}$

$t_{GPU} = N * N_{OPS} * C_{GPU} * T_{clkGPU} / N_{coresGPU}$

$t_{DF} = N_{OPS} * C_{DF} * T_{clkDF} + (N - N_{DF}) * T_{clkDF} / N_{DF}$

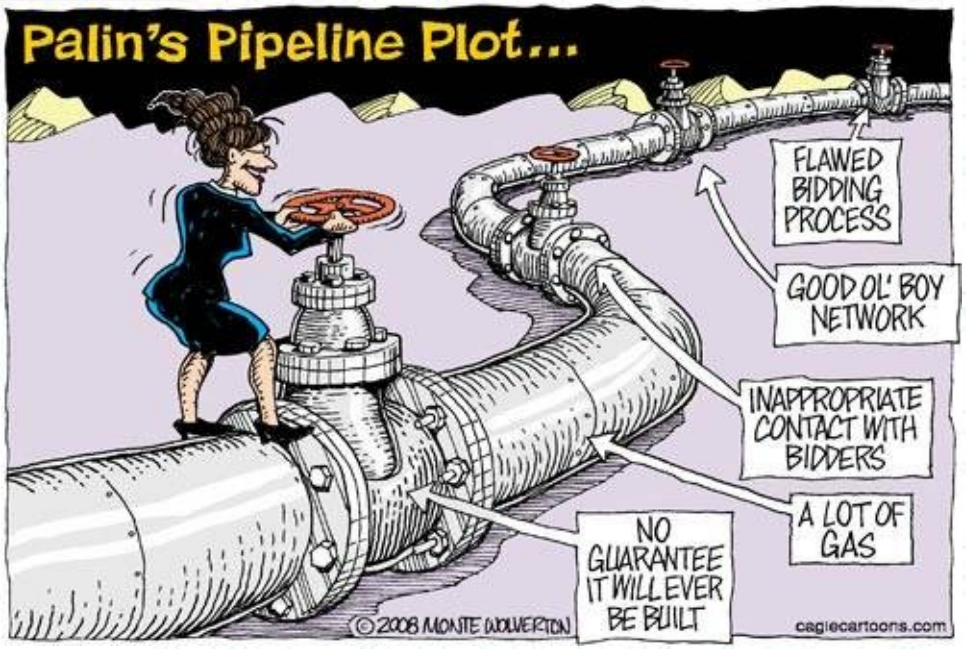


Assumptions:

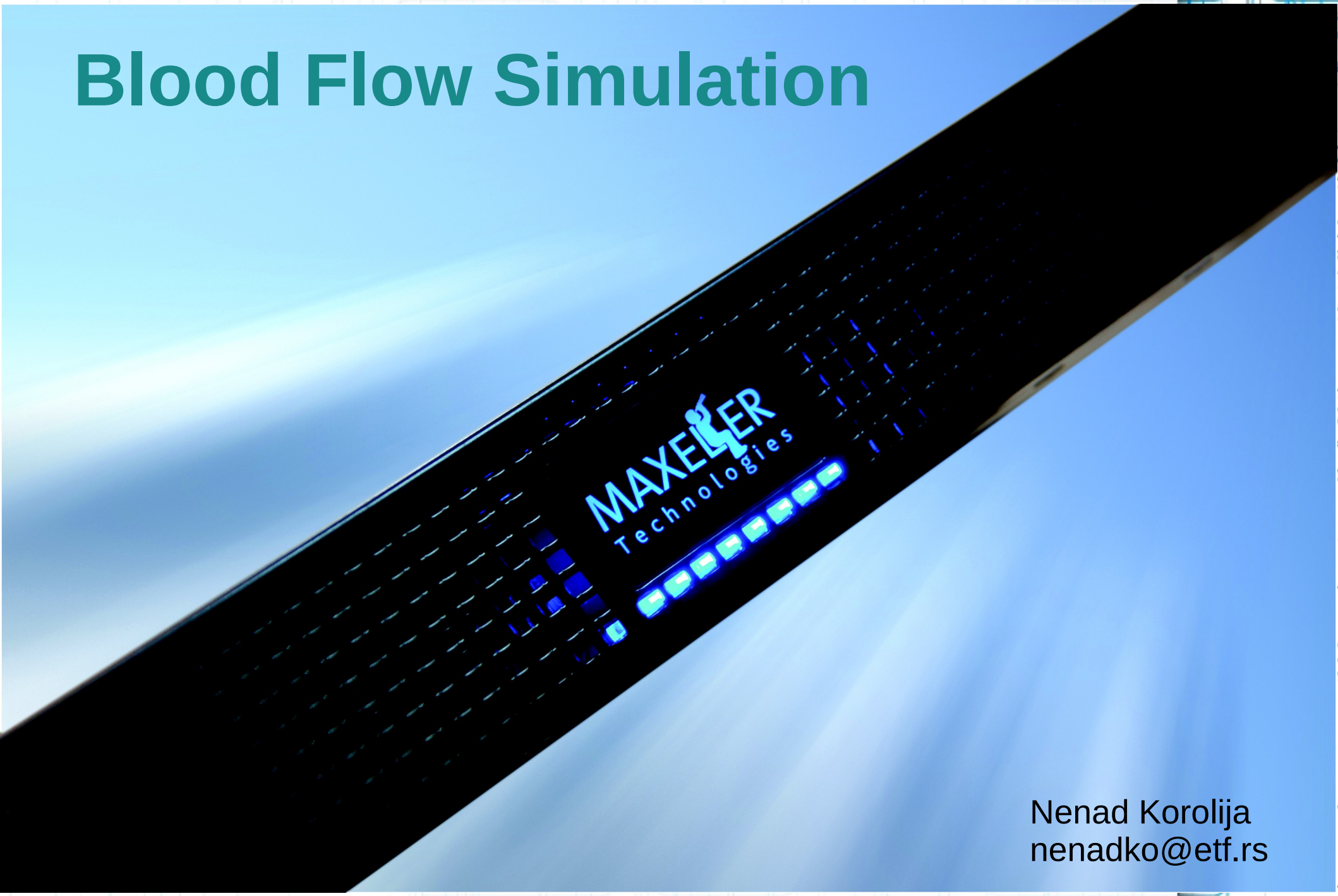
1. Software includes enough parallelism to keep all cores busy
2. The only limiting factor is the number of cores.

Expected Acceleration

- Same conditions as for Saša Stojanović
- Expected to run about three times faster
  - for some parts of code,
    calculations can run in parallel for about 9 instructions
  - for some parts of code,
    memory bandwidth is the bottleneck



Blood Flow Simulation



Nenad Korolija
nenadko@etf.rs