# Bohrium

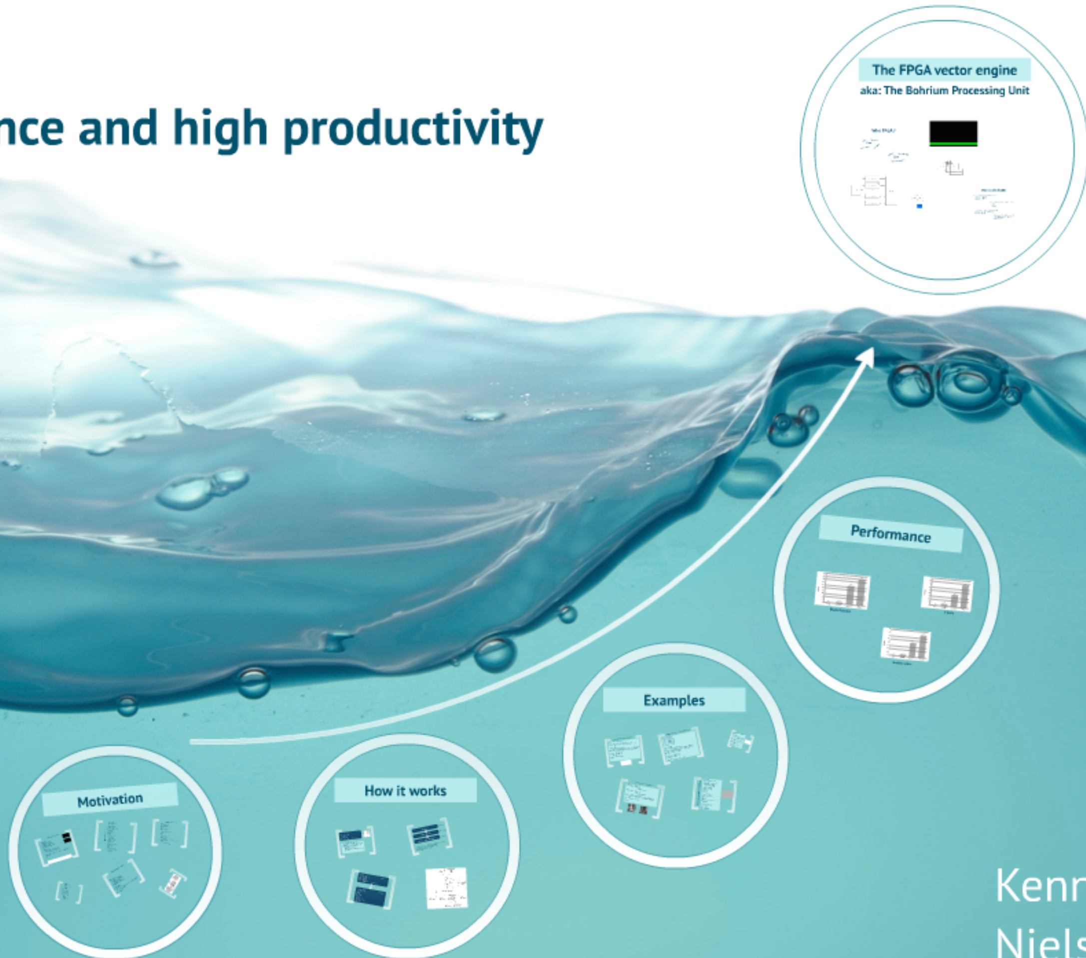
## Bridging high performance and high productivity

P3 Workshop - Bohrium 2015-04-13

Kenneth Skovhede
Niels Bohr Instituttet
Københavns Universitet

# Bohrium

## Bridging high performance and high productivity




P3 Workshop - Bohrium 2015-04-13


Kenneth Skovhede
Niels Bohr Instituttet
Københavns Universitet

# Motivation

Stencil example in Matlab

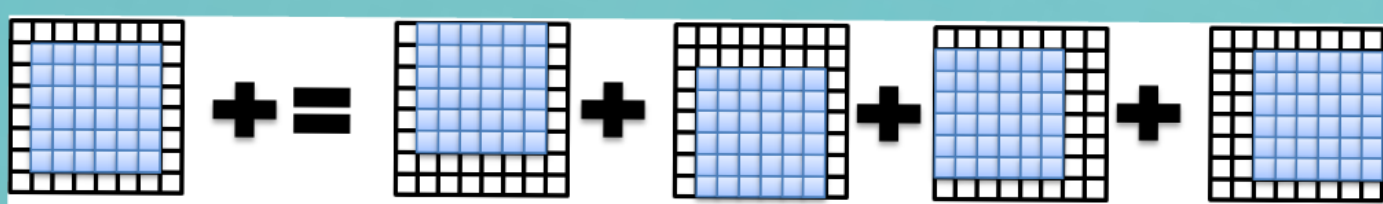
MPI with OpenMP

Stencil example in C

Stencil example in NumPy

# Stencil example in Matlab

```
#Parameters
I %Number of iterations
A %Input & Output Matrix
T %Temporary array
SIZE %Symmetric Matrix Size

#Computation
i = 2:SIZE+1;%Center slice vertical
j = 2:SIZE+1;%Center slice horizontal
for n=1:I,
   T(:) = (A(i,j) + A(i+1,j) + A(i-1,j) + A(i,j+1) + A(i,j-1)) / 5.0;
   A(i,j) = T;
end
```



# Stencil example in C

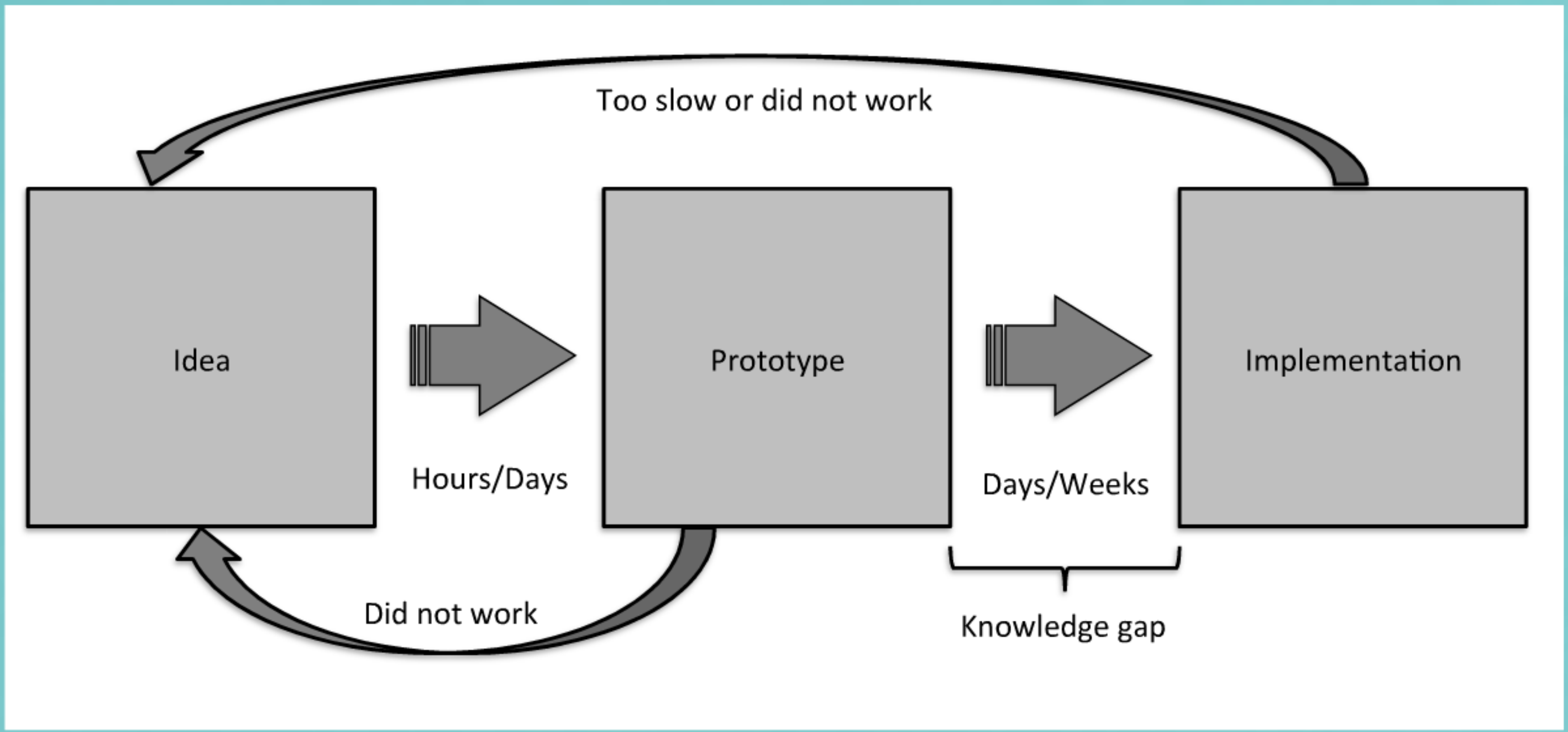
```
//Parameters
int I;     //Number of iterations
double *A; //Input & Output Matrix
double *T; //Temporary array
int SIZE;  //Symmetric Matrix Size

//Computation
int gsize = SIZE+2; //Size + borders.
for(n=0; n<I; n++)
{
  memcpy(T, A, gsize*gsize*sizeof(double));
  double *a = A;
  double *t = T;
  for(i=0; i<SIZE; ++i)
  {
    double *up     = a+1;
    double *left   = a+gsize;
    double *right  = a+gsize+2;
    double *down   = a+1+gsize*2;
    double *center = t+gsize+1;
    for(j=0; j<SIZE; ++j)
      *center++ = (*center + *up++ + *left++ + *right++ + *down++) / 5.0;
    a += gsize;
    t += gsize;
   }
   memcpy(A, T, gsize*gsize*sizeof(double));
}
```

# Stencil example with MPI

```
//Parameters
int I;    //Number of iterations
double *A; //Input & Output Matrix (local)
double *T; //Temporary array (local)
int SIZE;  //Symmetric Matrix Size (local)

//Computation
int gsize = SIZE+2; //Size + borders.
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &worldsize);
MPI_Comm comm;
int periods[] = {0};
MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize,
            periods, 1, &comm);
int l_size = SIZE / worldsize;
if(myrank == worldsize-1)
  l_size += SIZE % worldsize;
int l_gsize = l_size + 2;//Size + borders.
for(n=0; n<I; n++)
{
 int p_src, p_dest;
 //Send/receive - neighbor above
 MPI_Cart_shift(comm,0,1,&p_src,&p_dest);
 MPI_Sendrecv(A+gsize,gsize,MPI_DOUBLE,
          p_dest,1,A,gsize, MPI_DOUBLE,
          p_src,1,comm,MPI_STATUS_IGNORE);
 //Send/receive - neighbor below
 MPI_Cart_shift(comm,0,-1,&p_src,&p_dest);
 MPI_Sendrecv(A+(l_gsize-2)*gsize,
          gsize,MPI_DOUBLE,
          p_dest,1,A+(l_gsize-1)*gsize,
          gsize,MPI_DOUBLE,
          p_src,1,comm,MPI_STATUS_IGNORE);
 memcpy(T, A, l_gsize*gsize*sizeof(double));
 double *a = A;
 double *t = T;
 for(i=0; i<SIZE; ++i)
 {
  int a = i * gsize;
  double *up     = &A[a+1];
  double *left   = &A[a+gsize];
  double *right  = &A[a+gsize+2];
  double *down   = &A[a+1+gsize*2];
  double *center = &T[a+gsize+1];
  for(j=0; j<SIZE; ++j)
   *center++ = (*center + *up++ + *left++ + *right++ + *down++) / 5.0;
 }
 MPI_Barrier(MPI_COMM_WORLD);
```

# MPI with OpenMP

```
//Parameters
int I;     //Number of iterations
double *A; //Input & Output Matrix (local)
double *T; //Temporary array (local)
int SIZE;  //Symmetric Matrix Size (local)

//Computation
int gsize = SIZE+2; //Size + borders.
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &worldsize);
MPI_Comm comm;
int periods[] = {0};
MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize,
             periods, 1, &comm);
int l_size = SIZE / worldsize;
if(myrank == worldsize-1)
   l_size += SIZE % worldsize;
int l_gsize = l_size + 2;//Size + borders.
for(n=0; n<I; n++)
{
  int p_src, p_dest;
  MPI_Request reqs[4];

  //Initiate send/receive - neighbor above
  MPI_Cart_shift(comm, 0, 1, &p_src, &p_dest);
  MPI_Isend(A+gsize, gsize, MPI_DOUBLE, p_dest,
          1, comm, &reqs[0]);
  MPI_Irecv(A, gsize, MPI_DOUBLE, p_src,
          1, comm, &reqs[1]);

  //Initiate send/receive - neighbor below
  MPI_Cart_shift(comm, 0, -1, &p_src, &p_dest);
  MPI_Isend(A+(l_gsize-2)*gsize, gsize,
          MPI_DOUBLE,
          p_dest, 1, comm, &reqs[2]);
  MPI_Irecv(A+(l_gsize-1)*gsize, gsize,
          MPI_DOUBLE,
          p_src, 1, comm, &reqs[3]);

  //Handle the non-border elements.
  memcpy(T+gsize, A+gsize, l_size*gsize*sizeof(double));
  #pragma omp parallel for shared(A,T)
  for(i=1; i<l_size-1; ++i)
    compute_row(i,A,T,SIZE,gsize);

  //Handle the upper and lower ghost line
  MPI_Waitall(4, reqs, MPI_STATUSES_IGNORE);
  compute_row(0,A,T,SIZE,gsize);
  compute_row(l_size-1,A,T,SIZE,gsize);

  memcpy(A+gsize, T+gsize, l_size*gsize*sizeof(double));
}
MPI_Barrier(MPI_COMM_WORLD);
```

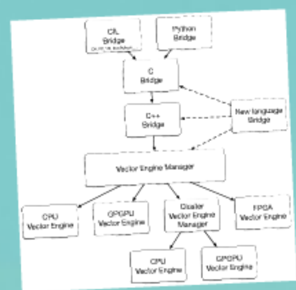

Too slow or did not work
Idea
Prototype
Implementation
Hours/Days
Days/Weeks
Did not work
Knowledge gap

# Stencil example in NumPy

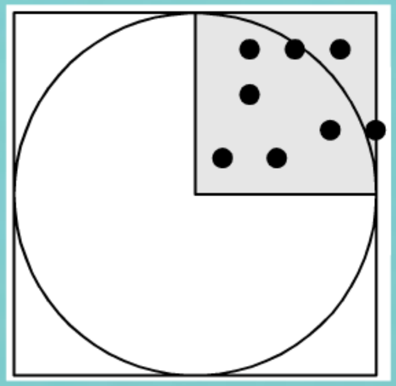
```
#Parameters
I    #Number of iterations
A    #Input & Output Matrix
T    #Temporary array
SIZE #Symmetric Matrix Size

#Computation
for i in xrange(I):
  T[:] = (A[1:-1,1:-1] + A[1:-1,:-2] + A[1:-1,2:] + A[:-2,1:-1] \
        + A[2:,1:-1]) / 5.0
  A[1:-1, 1:-1] = T
```

# How it works


Source code
Any language
Vector byte code
Hardware optimized execution
Any hardware


CIL
Bridge
Python
Bridge
C
Bridge
C++
Bridge
New language
Bridge
Vector Engine Manager
CPU
Vector Engine
GPGPU
Vector Engine
Cluster
Vector Engine
Manager
FPGA
Vector Engine
CPU
Vector Engine
GPGPU
Vector Engine

```
#Monte Carlo PI
size = [100, 2000, 20]
x = random(size)
y = random(size)
sum = add.aggregate((x^2 + y^2) <= 1)*4
```



- Can be written as sequential code
- Can be implemented as library in any language
- Can use special source language constructs
- Bohrium currently has support for:
  - Python/NumPy
  - CIL languages (.Net): C#, F#, VB, IronPython, etc.
  - C++
  - C

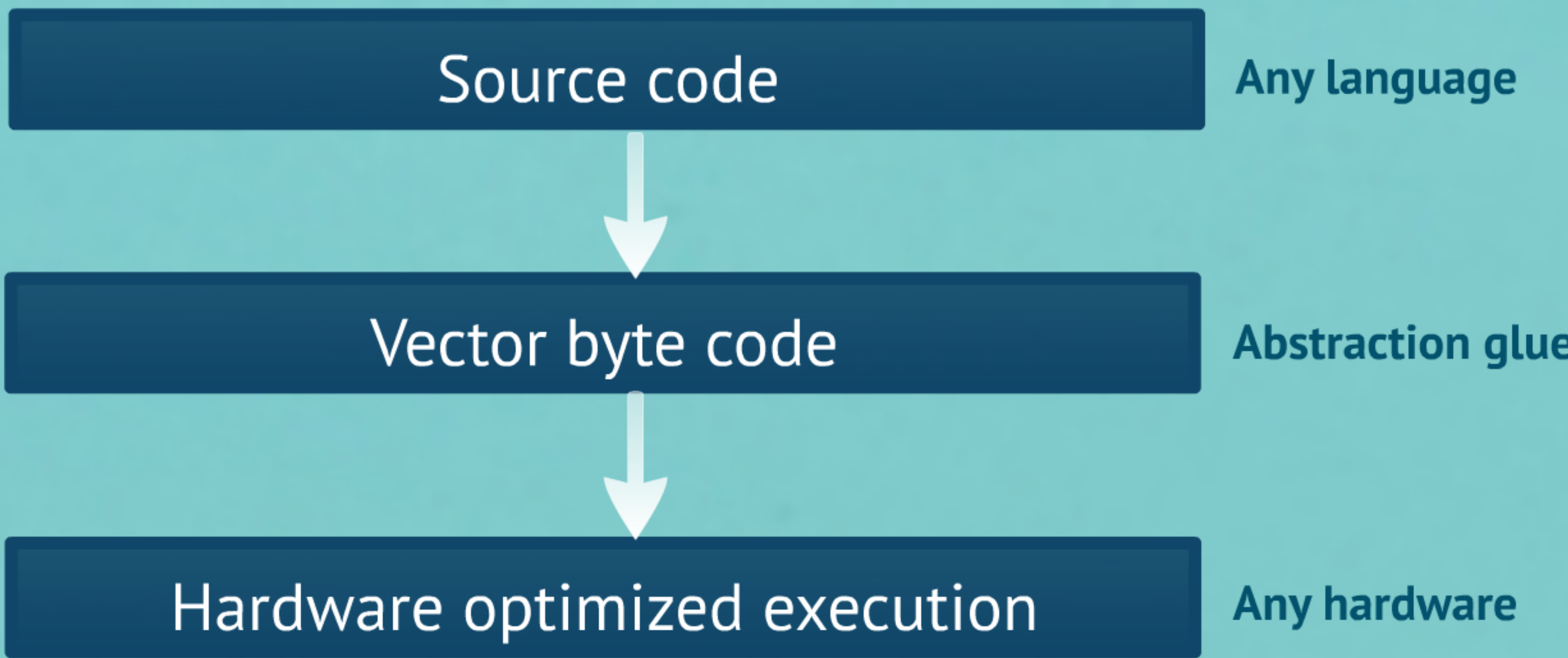
Sequential code

```
#Monte Carlo PI
size = [100, 2000, 20]
x = random(size)
y = random(size)
sum = add.aggregate((x^2 + y^2) <= 1)*4
```

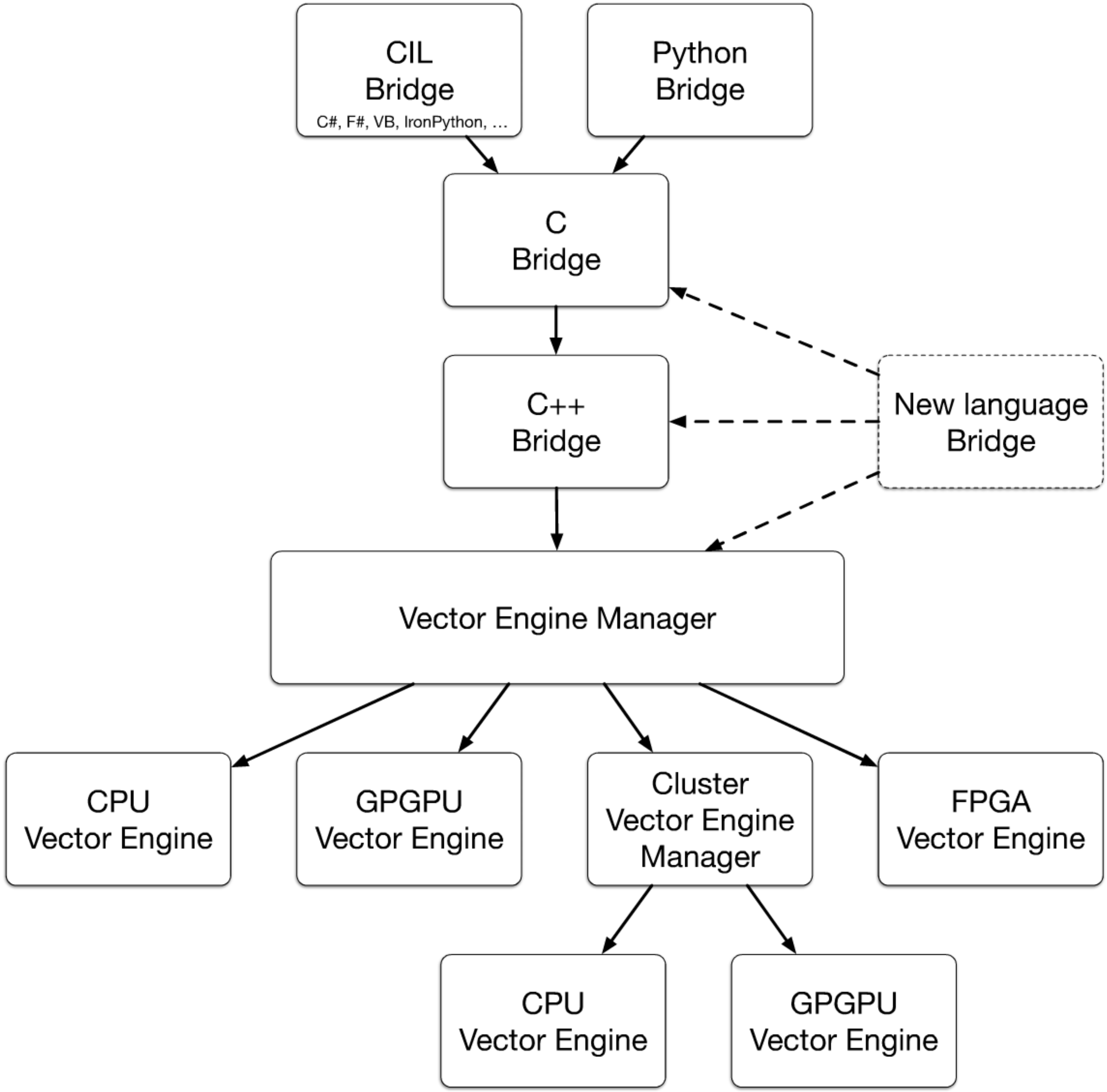
↓

Vector bytecode

```
X = NEW(100,2000,20)
Y = NEW(100,2000,20)
RND(X,X)
RND(Y,Y)
POW(T0,X,2)
POW(T1,Y,2)
ADD(T2,T1,T0)
LTE(T3,T2,1)
AGGREGATE(ADD,T4,T3)
MUL(T5,T4,4)
```




Any existing program can immediately use new hardware because of the abstraction
Every language gets all hardware support without any effort
Same basic idea as compiler intermediate files (aka object files)
but for vector operations and done runtime


CIL
Bridge
C#, F#, VB, IronPython, …
Python
Bridge
C
Bridge
C++
Bridge
New language
Bridge
Vector Engine Manager
CPU
Vector Engine
GPGPU
Vector Engine
Cluster
Vector Engine
Manager
FPGA
Vector Engine
CPU
Vector Engine
GPGPU
Vector Engine

# Examples


F# implementation of the BlackScholes core
Sobel filter for edge detection in Python
n-body simulation in Python

# C# implementation of BlackScholes core

```
private static NdArray CND(NdArray X)
{
   DATA a1 = 0.31938153f, a2 = -0.356563782f,;
   DATA a3 = 1.781477937f, a4 = -1.821255978f, a5 = 1.330274429f;

   var L = X.Abs();
   var K = 1.0f / (1.0f + 0.2316419f * L);
   var w = 1.0f - 1.0f / ((DATA)Math.Sqrt(2 * Math.PI)) * (-L * L / 2.0f).Exp() * (a1 *
K + a2 * (K.Pow(2)) + a3 * (K.Pow(3)) + a4 * (K.Pow(4)) + a5 * (K.Pow(5)));

   var mask1 = (NdArray)(X < 0);
   var mask2 = (NdArray)(X >= 0);

    w = w * mask2 + (1.0f - w) * mask1;
    return w;
}
```

$$C(S,t) = N(d_1)\ S - N(d_2)\ Ke^{-r(T-t)}$$

$$d_1 = \frac{\ln(\frac{S}{K}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = \frac{\ln(\frac{S}{K}) + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}$$

# F# implementation of the BlackScholes core

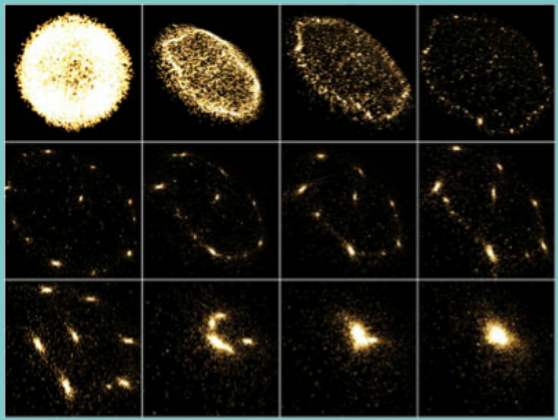
```
let CND(X:NdArray) =
   let a1 = 0.31938153
   let a2 = -0.356563782
   let a3 = 1.781477937
   let a4 = -1.821255978
   let a5 = 1.330274429

   let L = X.Abs()
   let K = 1.0 / (1.0 + 0.2316419 * L)
   let w = 1.0 - 1.0 / ((double(sqrt(2.0 * System.Math.PI)))) * (-L * L / 
2.0).Exp() * (a1 * K + a2 * (K.Pow(2.0)) + a3 * (K.Pow(3.0)) + a4 * 
(K.Pow(4.0)) + a5 * (K.Pow(5.0)));

   let mask1 = double(X < 0.0)
   let mask2 = double(X >= 0.0)

   w * mask2 + (NdArray(1.0) - w) * mask1
```

## Sobel filter for edge detection in Python

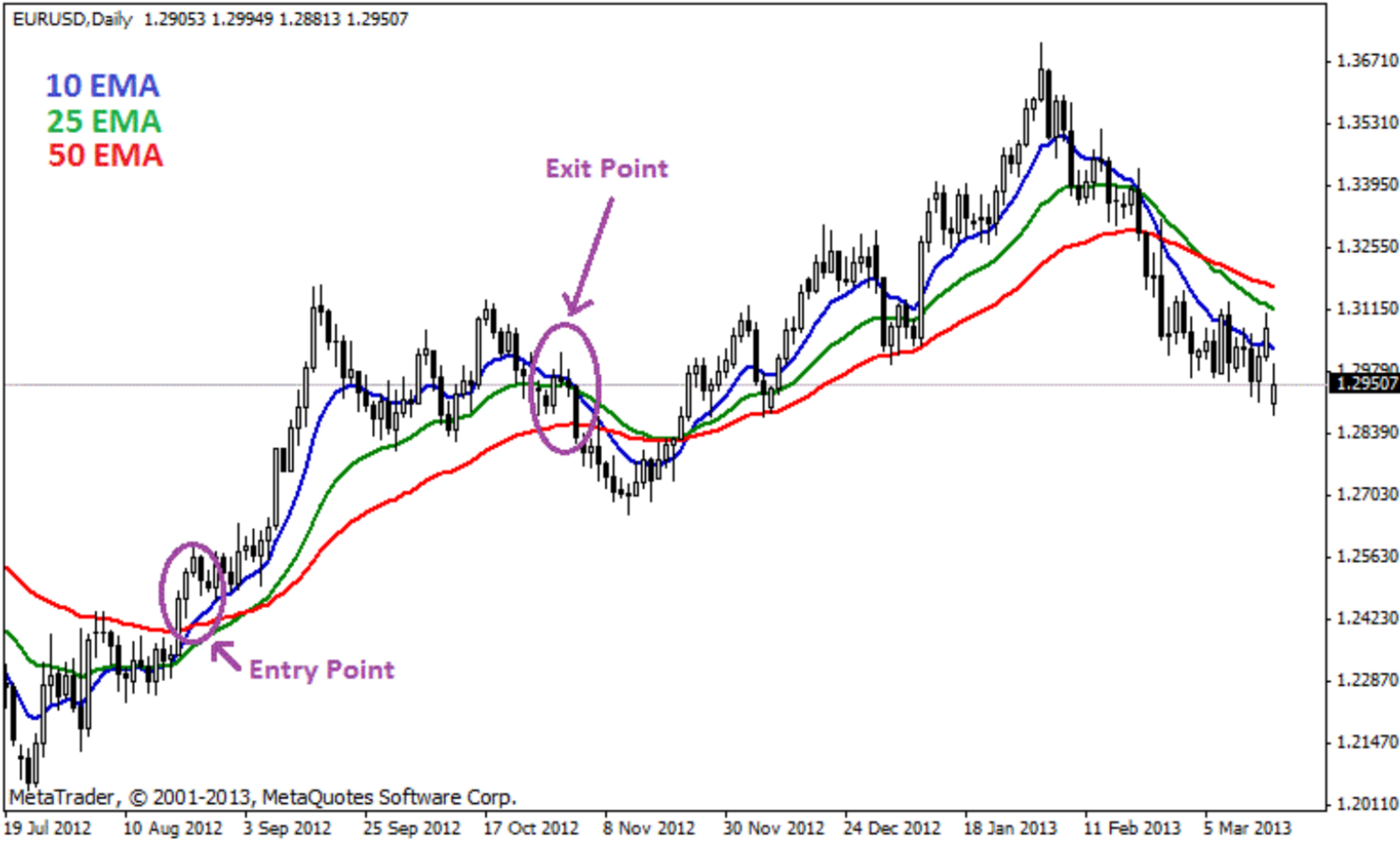
```
def sobel(input, data_type):
    sobel_window_x = array([[-1, 0, 1],
                            [-2, 0, 2],
                            [-1, 0, 1]]).astype(data_type)

    sobel_window_y = array([[-1, -2, -1],
                            [0, 0, 0],
                            [1, 2, 1]]).astype(data_type)

    sobel_x = convolve2d(input, sobel_window_x, out=None, data_type=data_type)
    sobel_y = convolve2d(input, sobel_window_y, out=None, data_type=data_type)

    result = sqrt(sobel_x**2 + sobel_y**2)

    return result
```

# n-body simulation in Python

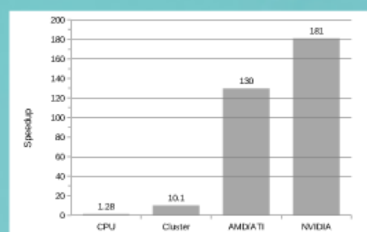
```
def move(galaxy, dt):
    """Move the bodies
    first find forces and change velocity and then move positions
    """
    n  = len(galaxy['x'])
    # Calculate all dictances component wise (with sign)
    dx = galaxy['x'][np.newaxis,:].T - galaxy['x']
    dy = galaxy['y'][np.newaxis,:].T - galaxy['y']
    dz = galaxy['z'][np.newaxis,:].T - galaxy['z']

    # Euclidian distances (all bodys)
    r = np.sqrt(dx**2 + dy**2 + dz**2)
    np.diagonal(r)[:] = 1.0

    # prevent collition
    mask = r < 1.0
    r = r * ~mask + 1.0 * mask

    m = galaxy['m'][np.newaxis,:].T

    # Calculate the acceleration component wise
    Fx = G*m*dx/r**3
    Fy = G*m*dy/r**3
    Fz = G*m*dz/r**3
    # Set the force (acceleration) a body exerts on it self to zero
    np.diagonal(Fx)[:] = 0.0
    np.diagonal(Fy)[:] = 0.0
    np.diagonal(Fz)[:] = 0.0

    galaxy['vx'] += dt*np.sum(Fx, axis=0)
    galaxy['vy'] += dt*np.sum(Fy, axis=0)
    galaxy['vz'] += dt*np.sum(Fz, axis=0)

    galaxy['x'] += dt*galaxy['vx']
    galaxy['y'] += dt*galaxy['vy']
    galaxy['z'] += dt*galaxy['vz']
```



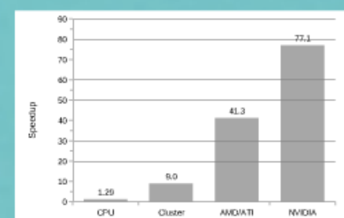
```
from numpy.lib.stride_tricks import as_strided as ast
import numpy as np
import math

alpha = 0.25 #Input value
epsilon = 0.0001 #Cutoff
raw_data = np.random.random_sample((100000,)) #simulated data

# Determine the window size based on epsilon and alpha
window_size = int(math.ceil(math.log(epsilon) / math.log(1-alpha)))

betas = np.empty(window_size) #Precompute contributions
betas.fill(1-alpha)
rates = np.power(betas, len(betas) - 1 - np.arange(0, len(betas)))
rates[1:] *= alpha

padding = np.empty(window_size - 1) #Produce padding
padding.fill(raw_data[0])
data = np.concatenate((padding, raw_data))

# Transform the data into a set of series
series = ast(data, shape=(len(raw_data), window_size), \
 strides=(1*data.itemsize, 1*data.itemsize))

result = np.add.reduce(series * rates, axis=1)
```


EURUSD,Daily 1.29053 1.29949 1.28813 1.29507
10 EMA
25 EMA
50 EMA
Exit Point
Entry Point
MetaTrader, © 2001-2013, MetaQuotes Software Corp.
1.36710
1.35310
1.33950
1.32550
1.31150
1.29790
1.29507
1.28390
1.27030
1.25630
1.24230
1.22870
1.21470
1.20110
19 Jul 2012
10 Aug 2012
3 Sep 2012
25 Sep 2012
17 Oct 2012
8 Nov 2012
30 Nov 2012
24 Dec 2012
18 Jan 2013
11 Feb 2013
5 Mar 2013

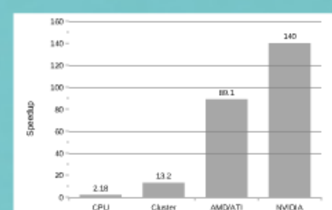
# Performance



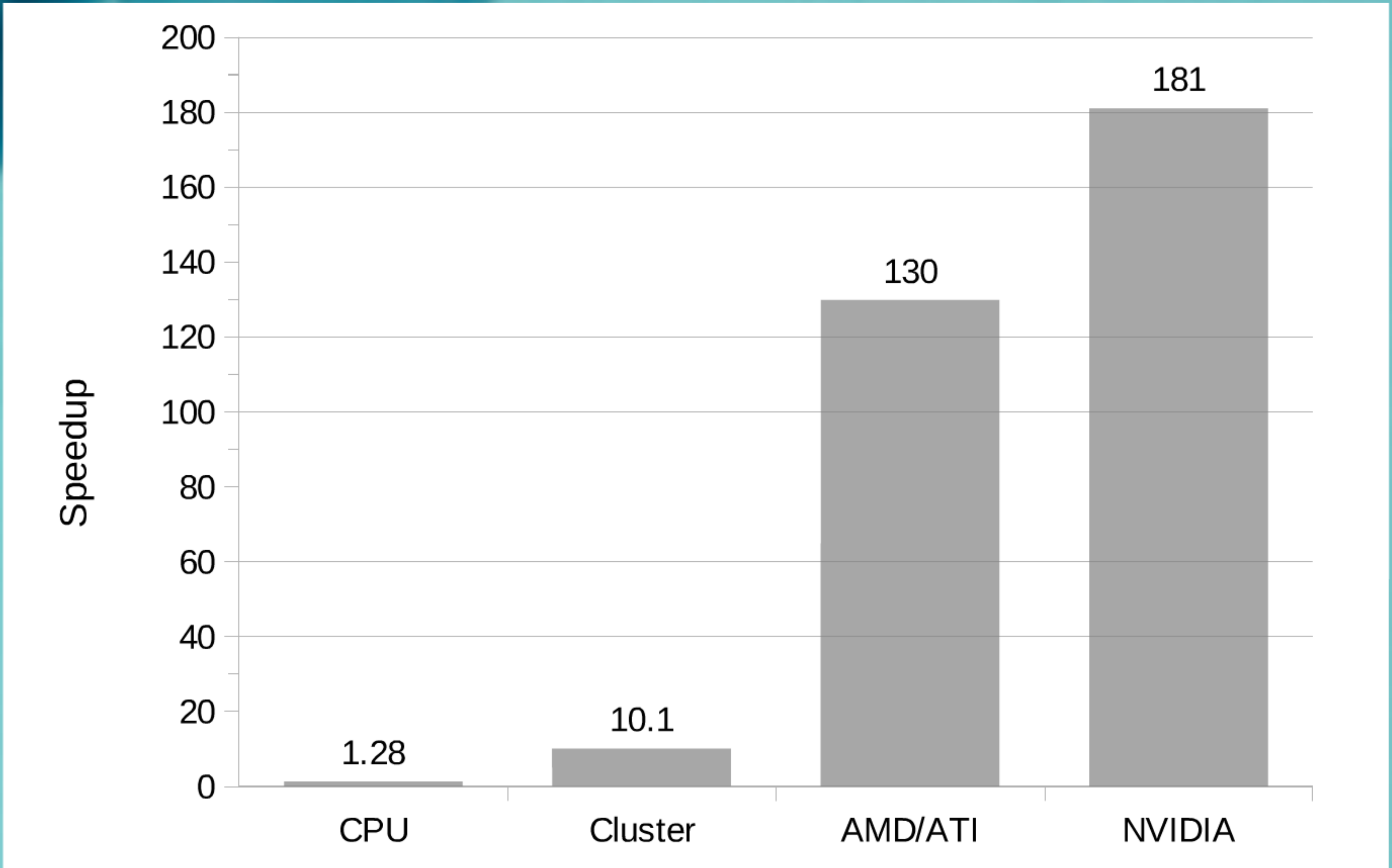

Black-Scholes



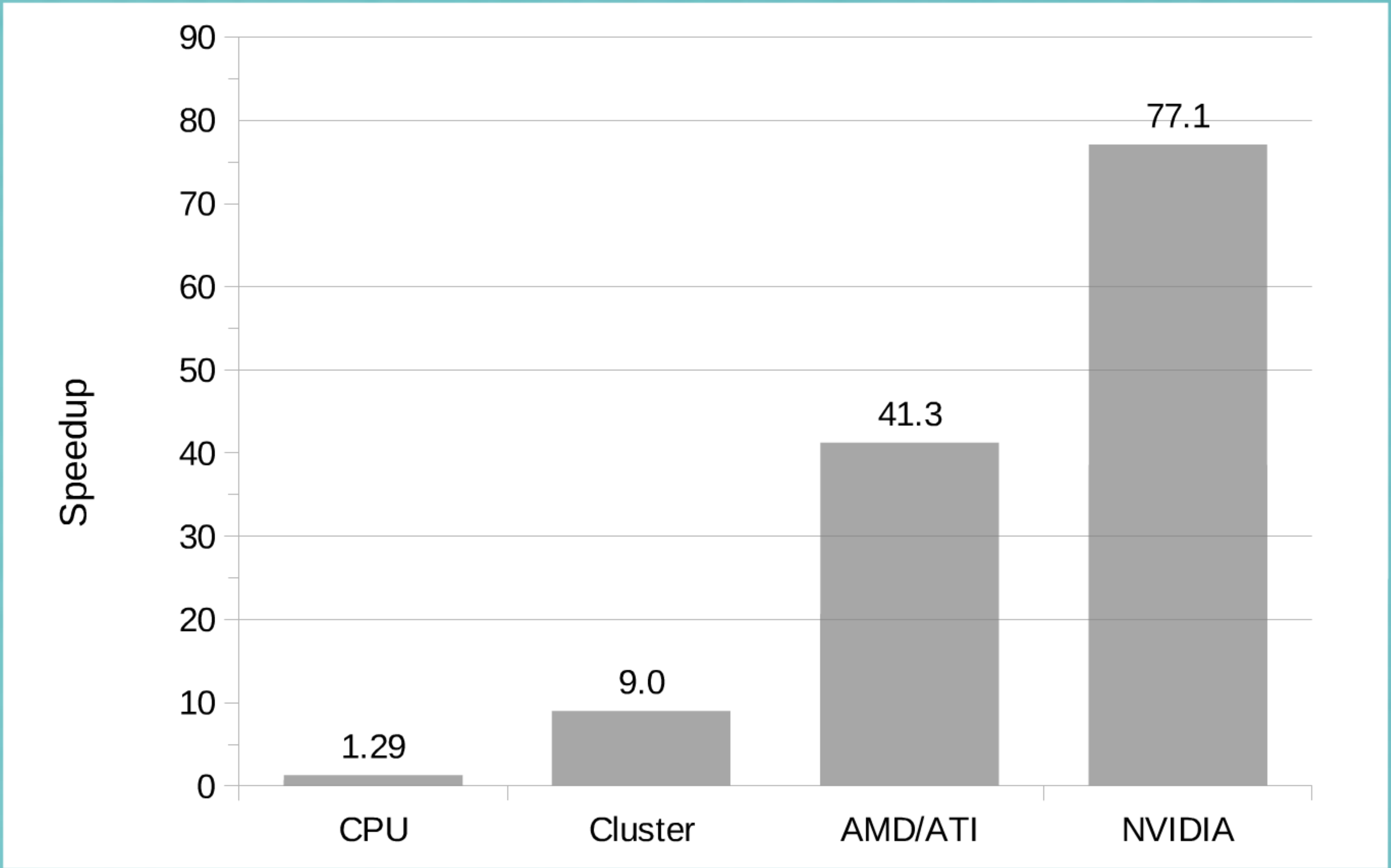

n-body



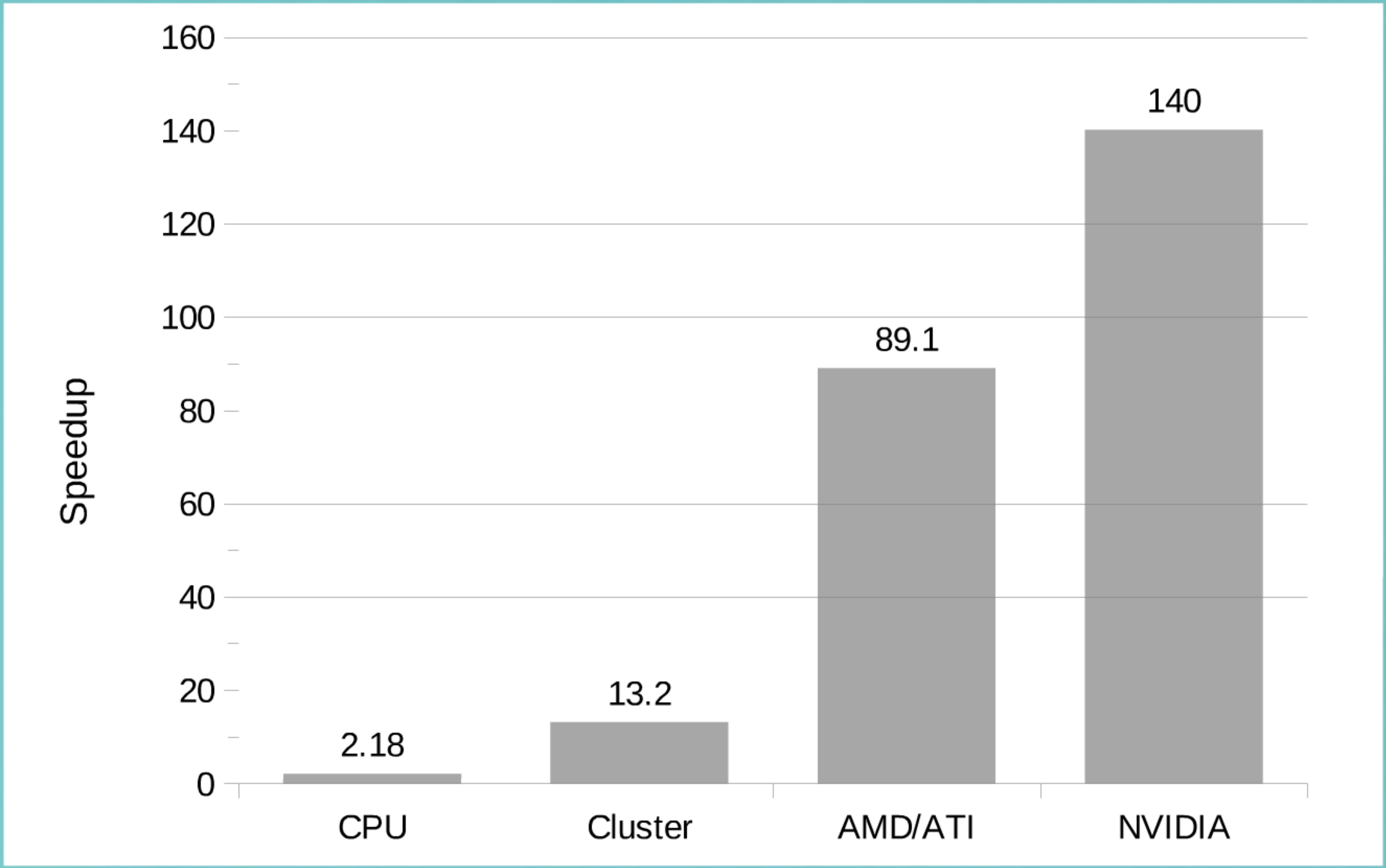

Shallow water




# Black-Scholes


90
80
70
60
50
40
30
20
10
0
Speedup
1.29
9.0
41.3
77.1
CPU
Cluster
AMD/ATI
NVIDIA

# n-body




# Shallow water

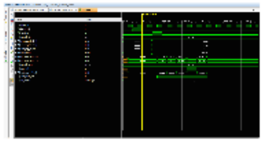
# The FPGA vector engine

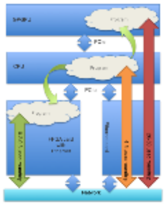
## aka: The Bohrium Processing Unit

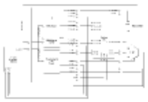
### Why FPGA ?

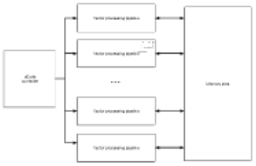
**Pro**
- Stable
- Stand-alone
- Power-efficient
- Low latency

**Con**
- Expensive
- Low availability
- Difficult to program









### Micro-code design

# Why FPGA ?

**Pro**

Stable

Stand-alone

Power-efficient

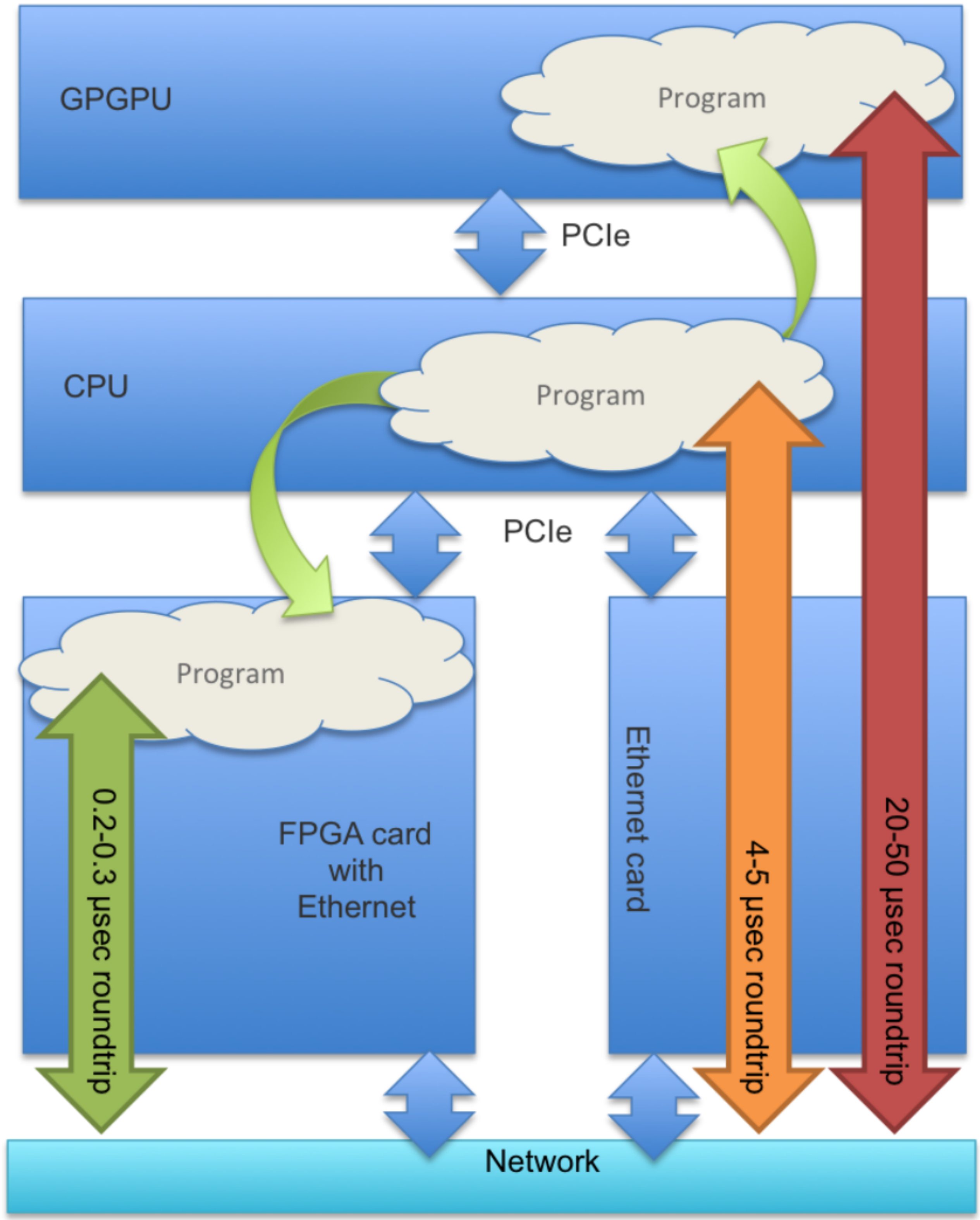
Low latency

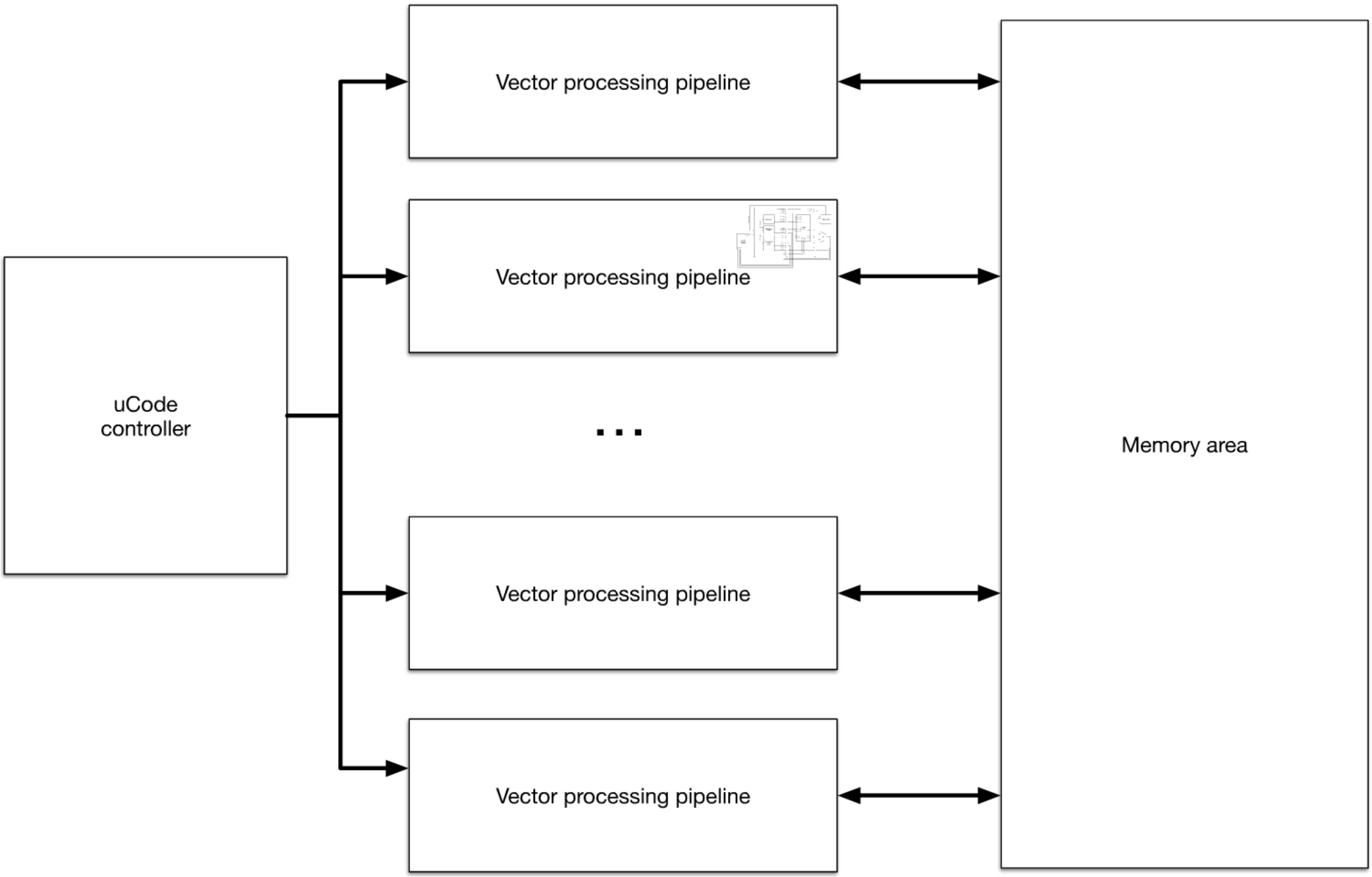
**Con**

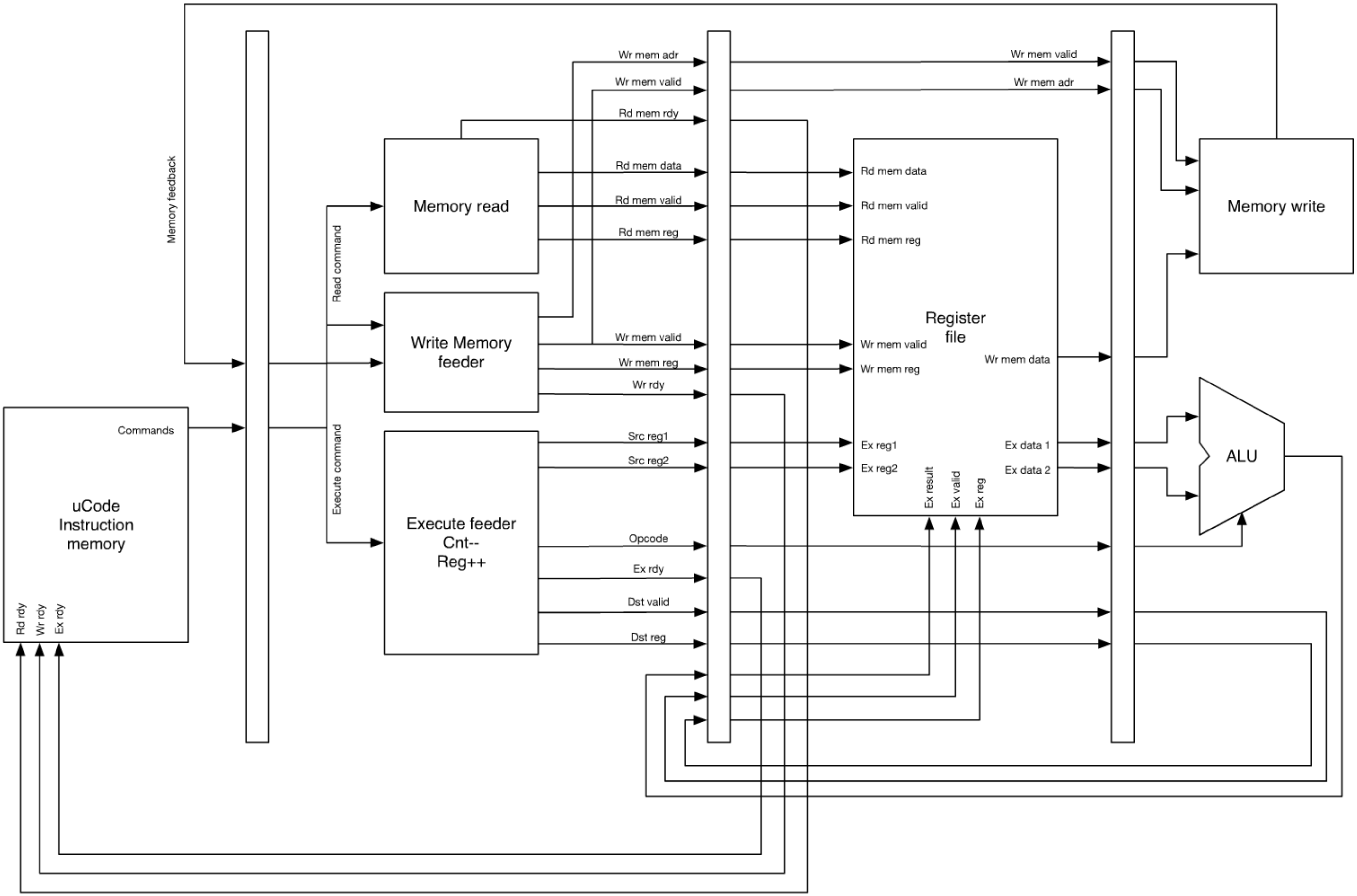
Expensive

Low availability

Difficult to program


GPGPU
Program
PCIe
CPU
Program
PCIe
Program
FPGA card
with
Ethernet
Ethernet card
0.2-0.3 μsec roundtrip
4-5 μsec roundtrip
20-50 μsec roundtrip
Network

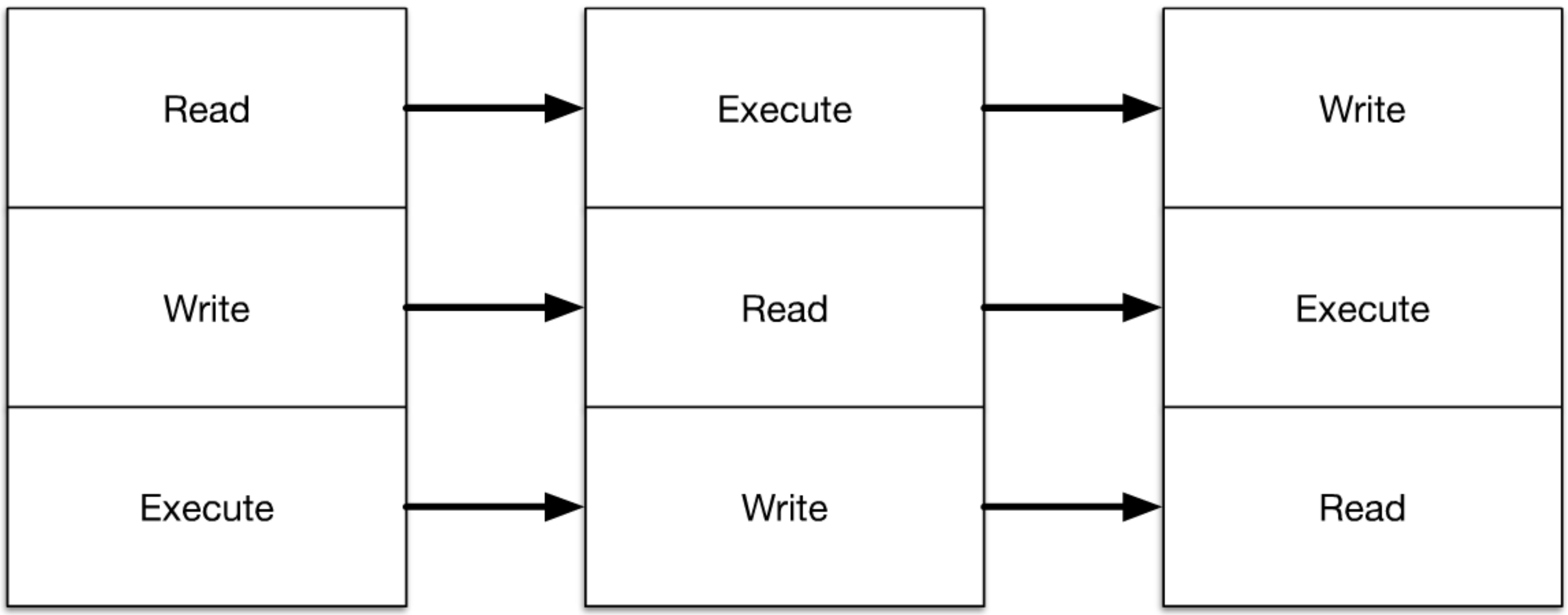

Vector processing pipeline
Vector processing pipeline
uCode
controller
...
Memory area
Vector processing pipeline
Vector processing pipeline

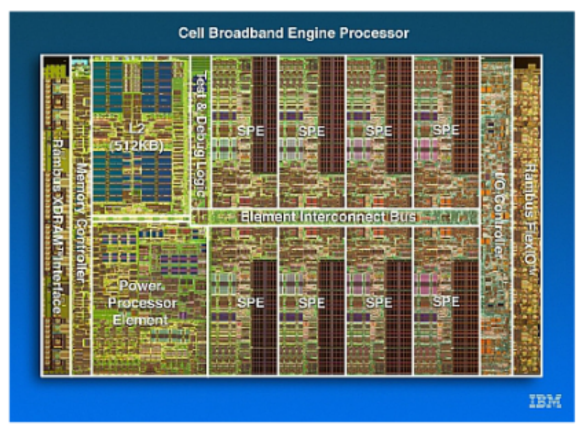


Memory feedback
Wr mem adr
Wr mem valid
Rd mem rdy
Rd mem data
Rd mem valid
Rd mem reg
Memory read
Read command
Write Memory feeder
Wr mem valid
Wr mem reg
Wr rdy
Commands
Execute command
Execute feeder Cnt-- Reg++
Src reg1
Src reg2
Opcode
Ex rdy
Dst valid
Dst reg
uCode Instruction memory
Rd rdy
Wr rdy
Ex rdy
Wr mem valid
Wr mem adr
Register file
Rd mem data
Rd mem valid
Rd mem reg
Wr mem valid
Wr mem reg
Wr mem data
Ex reg1
Ex reg2
Ex data 1
Ex data 2
Ex result
Ex valid
Ex reg
Memory write
ALU

# Register file shuffle - 3-state execution

## Triple buffering



# Micro-code design

Each microcode can express 3 distinct operations:

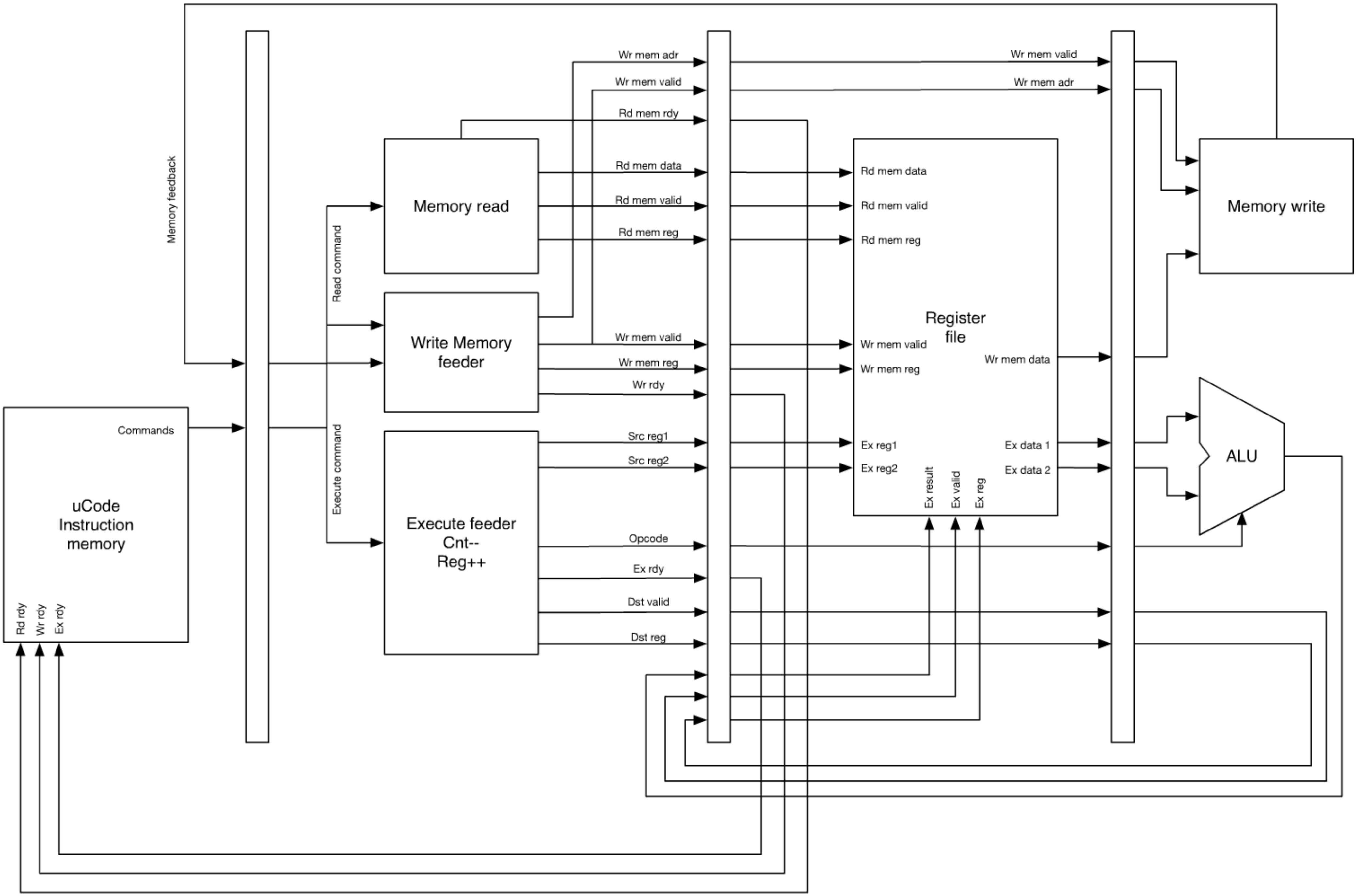
- Vectorized memory read
- Vectorized memory write
- Vectorized execute

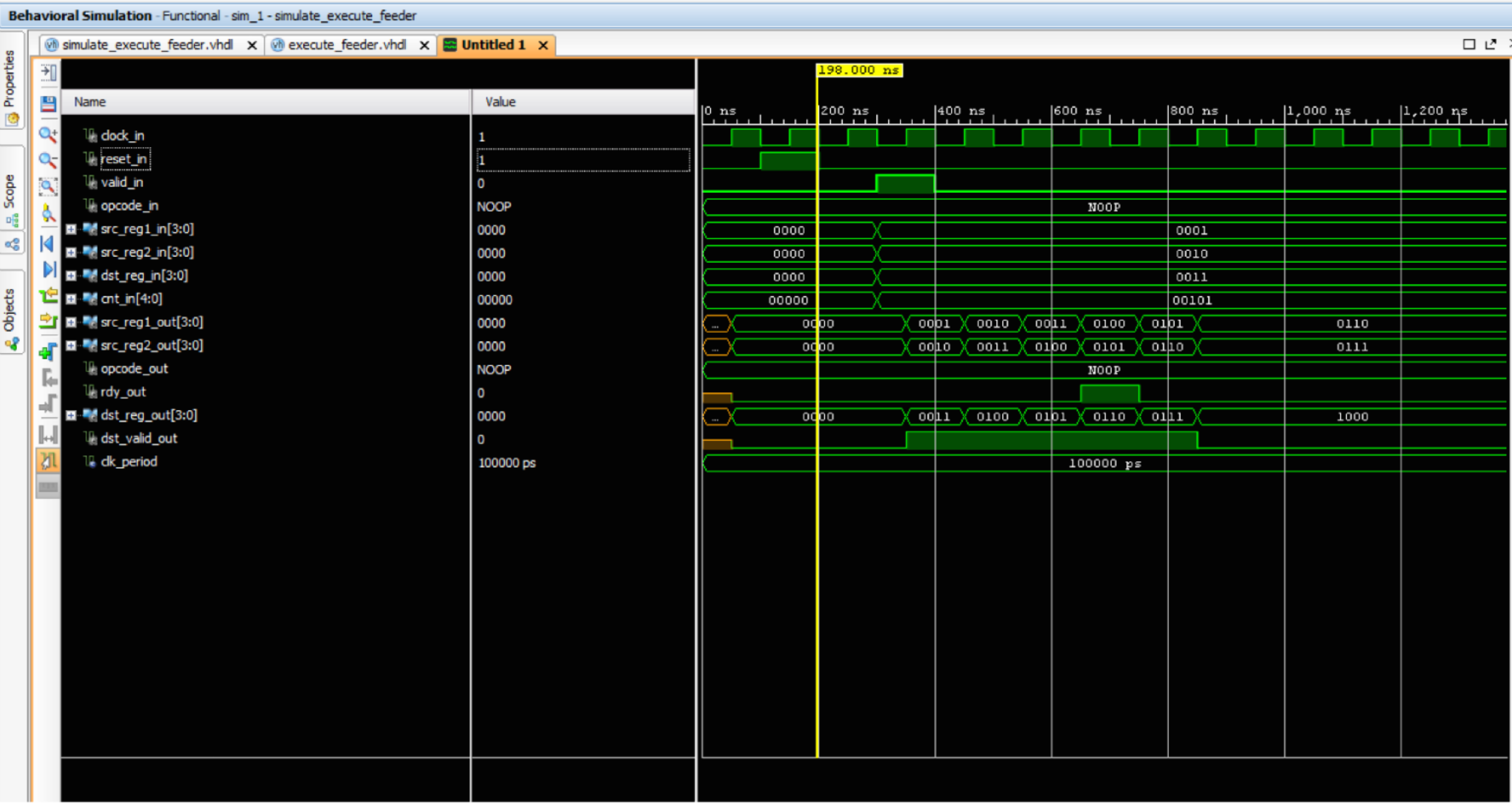
Each microcode can depend on previous instructions

- For read
- For write
- For execute

The micro-code design places a large burden on the programmer and is not suited for writing conventional programs

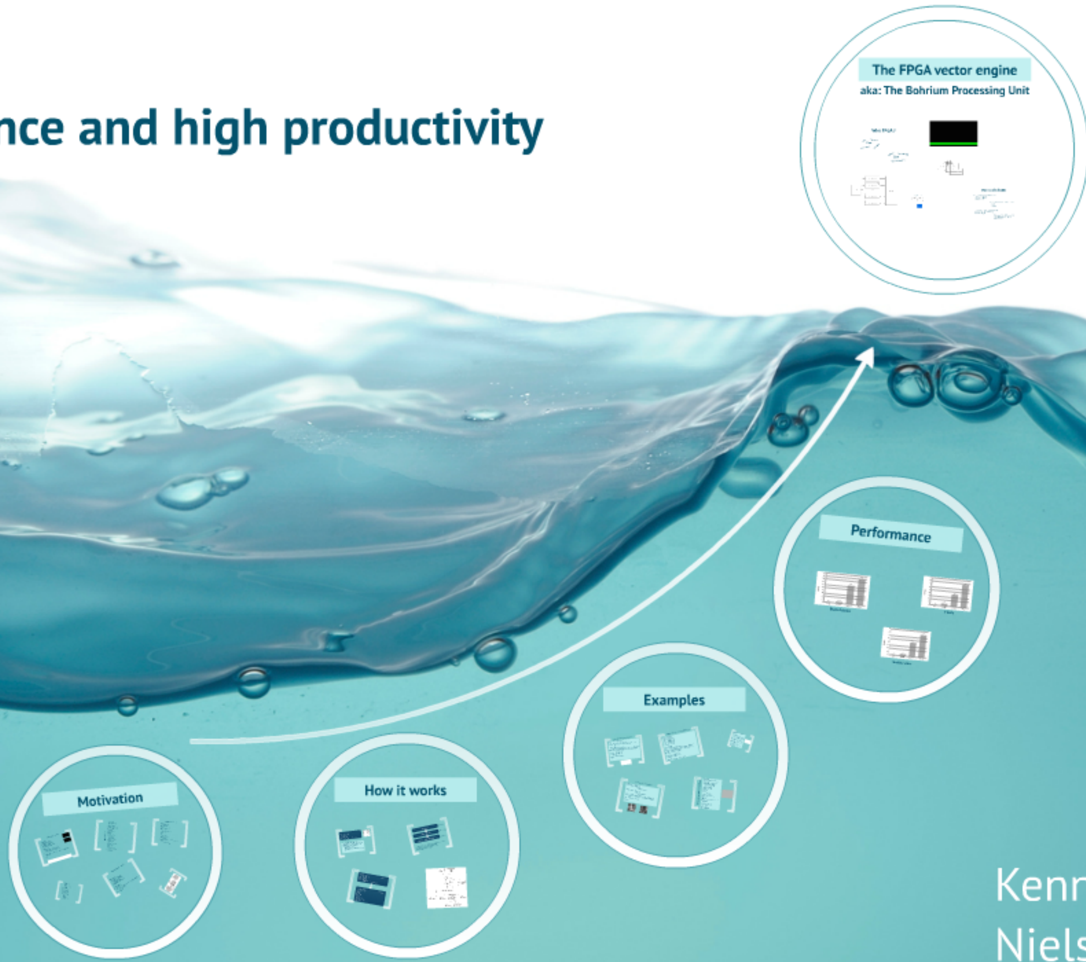
But it is a perfect fit for the Bohrium bytecode, and essentially a processor designed for the programming model


Memory feedback
Wr mem adr
Wr mem valid
Rd mem rdy
Rd mem data
Rd mem valid
Rd mem reg
Memory read
Read command
Write Memory feeder
Wr mem valid
Wr mem reg
Wr rdy
Commands
uCode Instruction memory
Rd rdy
Wr rdy
Ex rdy
Execute command
Execute feeder Cnt-- Reg++
Src reg1
Src reg2
Opcode
Ex rdy
Dst valid
Dst reg
Register file
Rd mem data
Rd mem valid
Rd mem reg
Wr mem valid
Wr mem reg
Wr mem data
Ex reg1
Ex reg2
Ex data 1
Ex data 2
Ex result
Ex valid
Ex reg
Wr mem valid
Wr mem adr
Memory write
ALU


Behavioral Simulation - Functional - sim_1 - simulate_execute_feeder
simulate_execute_feeder.vhdl
execute_feeder.vhdl
Untitled 1
Name
Value
clock_in
1
reset_in
1
valid_in
0
opcode_in
NOOP
src_reg1_in[3:0]
0000
src_reg2_in[3:0]
0000
dst_reg_in[3:0]
0000
cnt_in[4:0]
00000
src_reg1_out[3:0]
0000
src_reg2_out[3:0]
0000
opcode_out
NOOP
rdy_out
0
dst_reg_out[3:0]
0000
dst_valid_out
0
clk_period
100000 ps
198.000 ns
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
NOOP
0001
0010
0011
00101
0110
0111
1000
100000 ps
Properties
Scope
Objects

# Bohrium

## Bridging high performance and high productivity




P3 Workshop - Bohrium 2015-04-13

Kenneth Skovhede
Niels Bohr Instituttet
Københavns Universitet