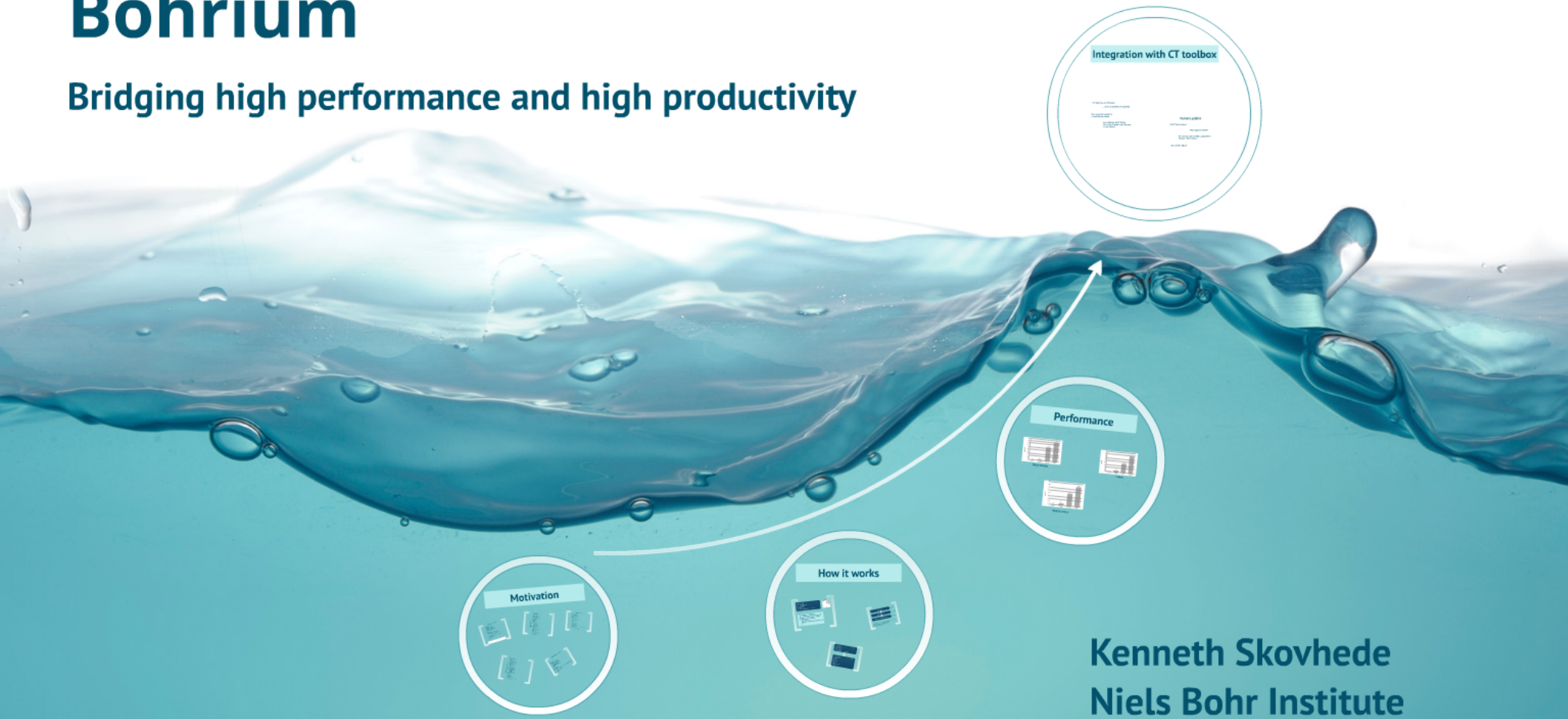
Bohrium

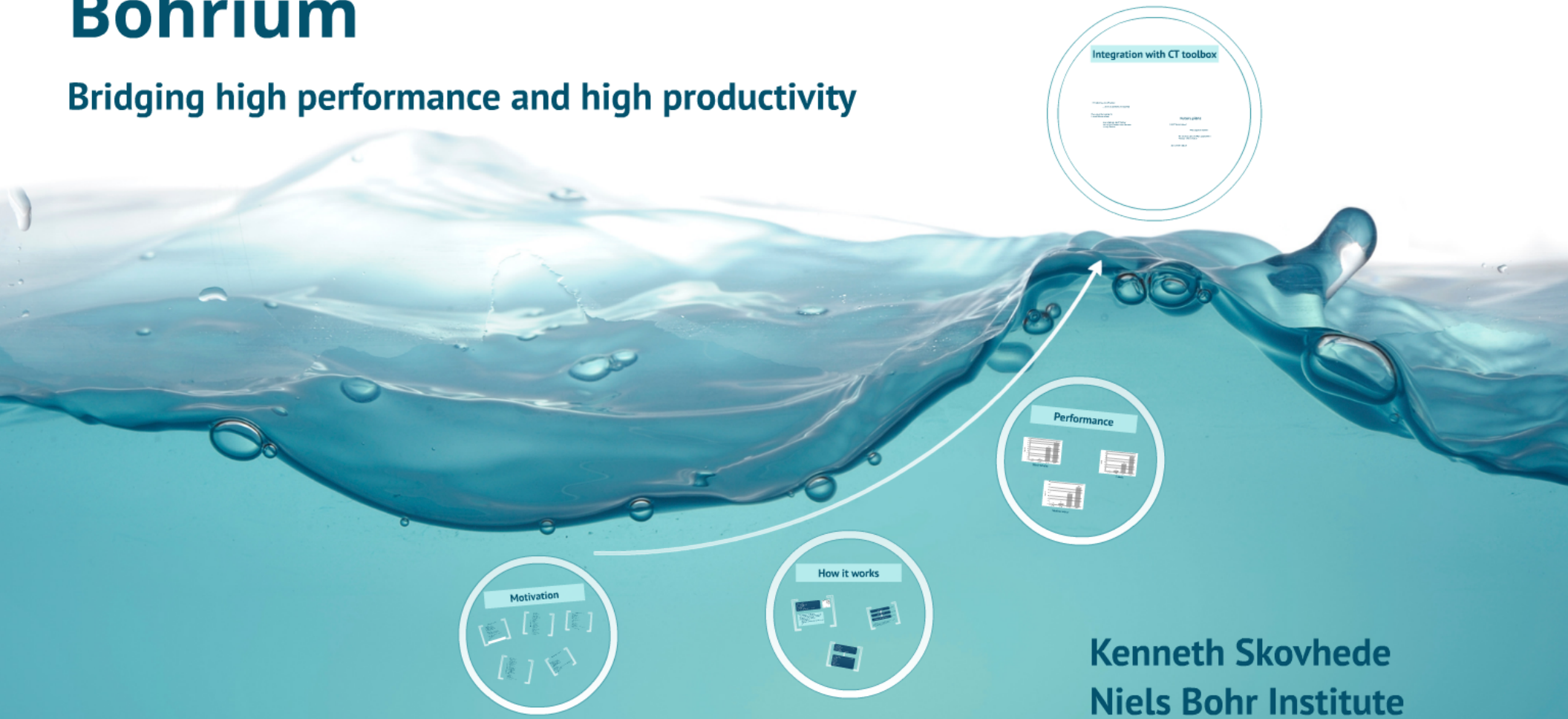
Bridging high performance and high productivity



Kenneth Skovhede
Niels Bohr Institute
Workshop 2014-02-13

Bohrium

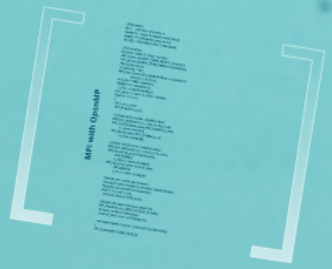
Bridging high performance and high productivity

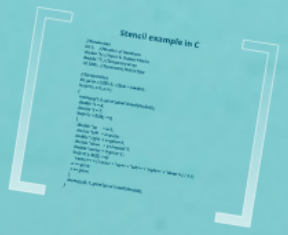


Kenneth Skovhede
Niels Bohr Institute
Workshop 2014-02-13

Motivation


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(I,J = (A(I,J) + A(I+1,J) + A(I-1,J) + A(I,J+1)...
+ A(I,J-1)) / 5.0;
A(:,J) = T;
end


MPI with Openmp

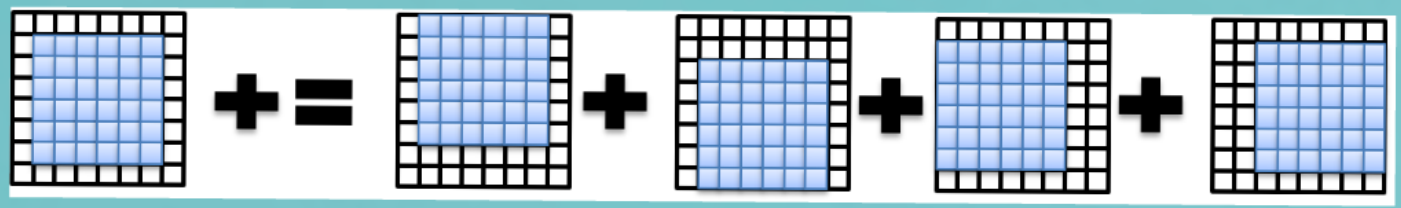

Stencil example in C


Stencil example with MPI


Stencil example in NumPy
#Parameters
I # Number of Iterations
A #Input & Output Matrix
T #Temporary array
SIZE #Symmetric Matrix Size
#Computation
for i in range(0):
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

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 l;     //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<l; 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 l;     //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<l; 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);
```

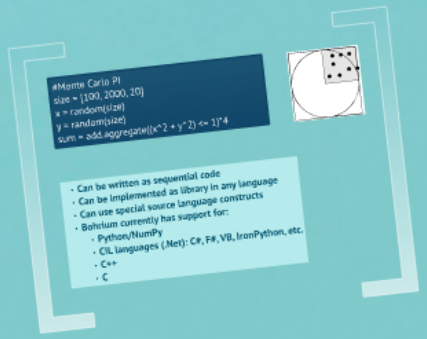
MPI with OpenMP

```
//Parameters
int l;    //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<l; 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);
```

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
```

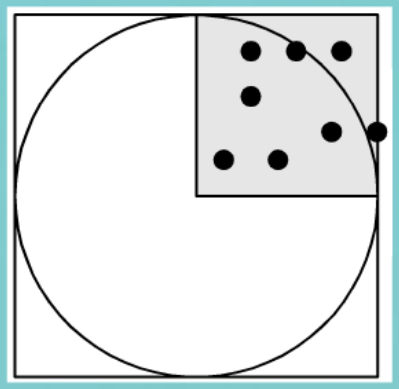
How it works


#Monte Carlo PI
size = 1000, 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


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


#Monte Carlo PI
Sequential code
Vector bytecode

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



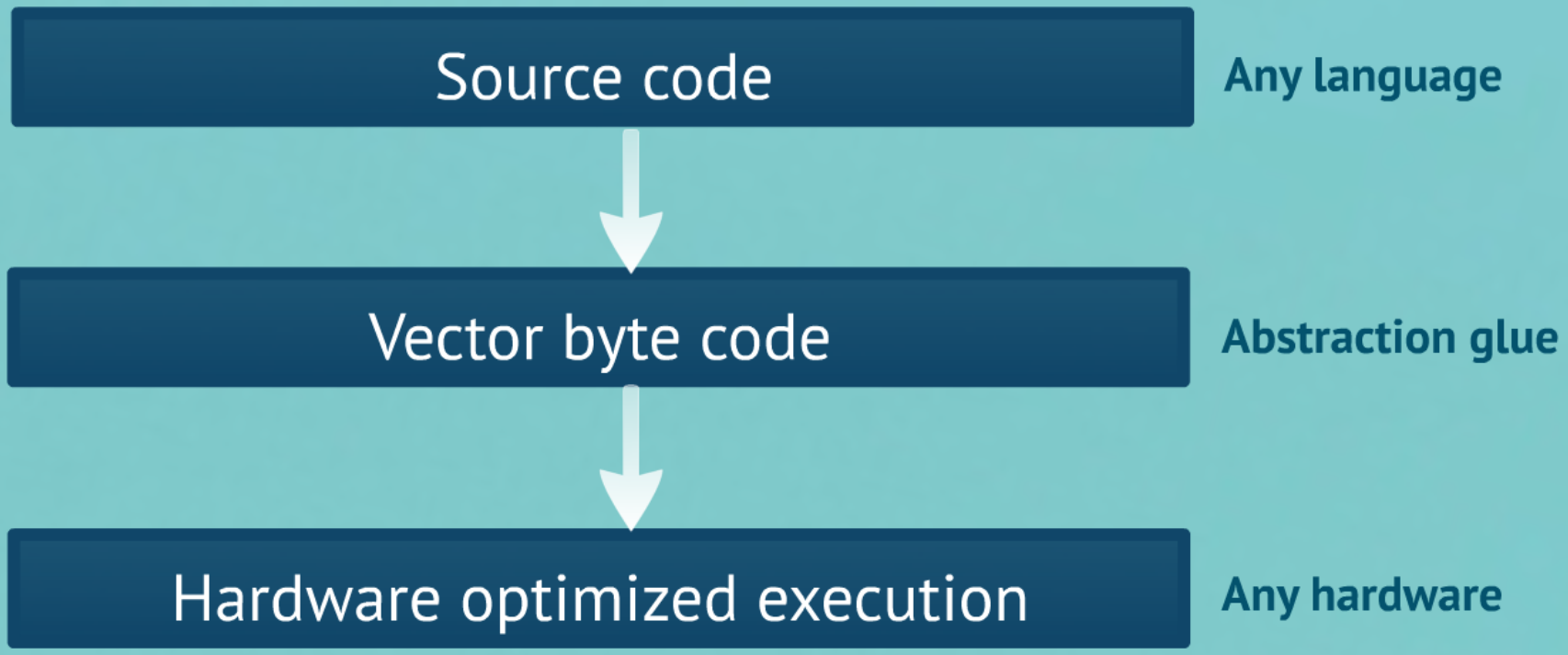
- Can be written as sequential code
- Can be implemented as library in any language
- Can use special source language constructs
- Bohrium currently has support for:
  - Python/NumPy
  - CIL languages (.Net): C#, F#, VB, IronPython, etc.
  - C++
  - C

Sequential code

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

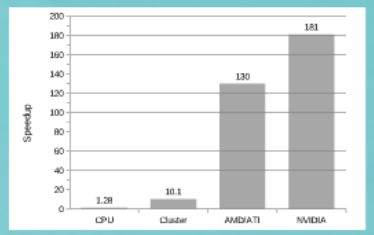
Vector bytecode

```
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)
```

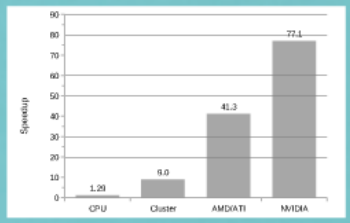


Any existing program can immediately use new hardware because of the abstraction
Every language gets all hardware support without any effort
Same basic idea as compiler intermediate files (aka object files)
but for vector operations and done runtime

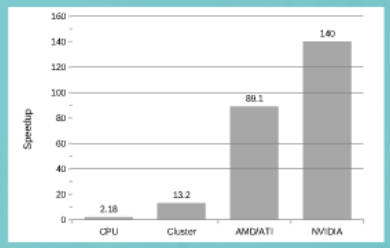
Performance



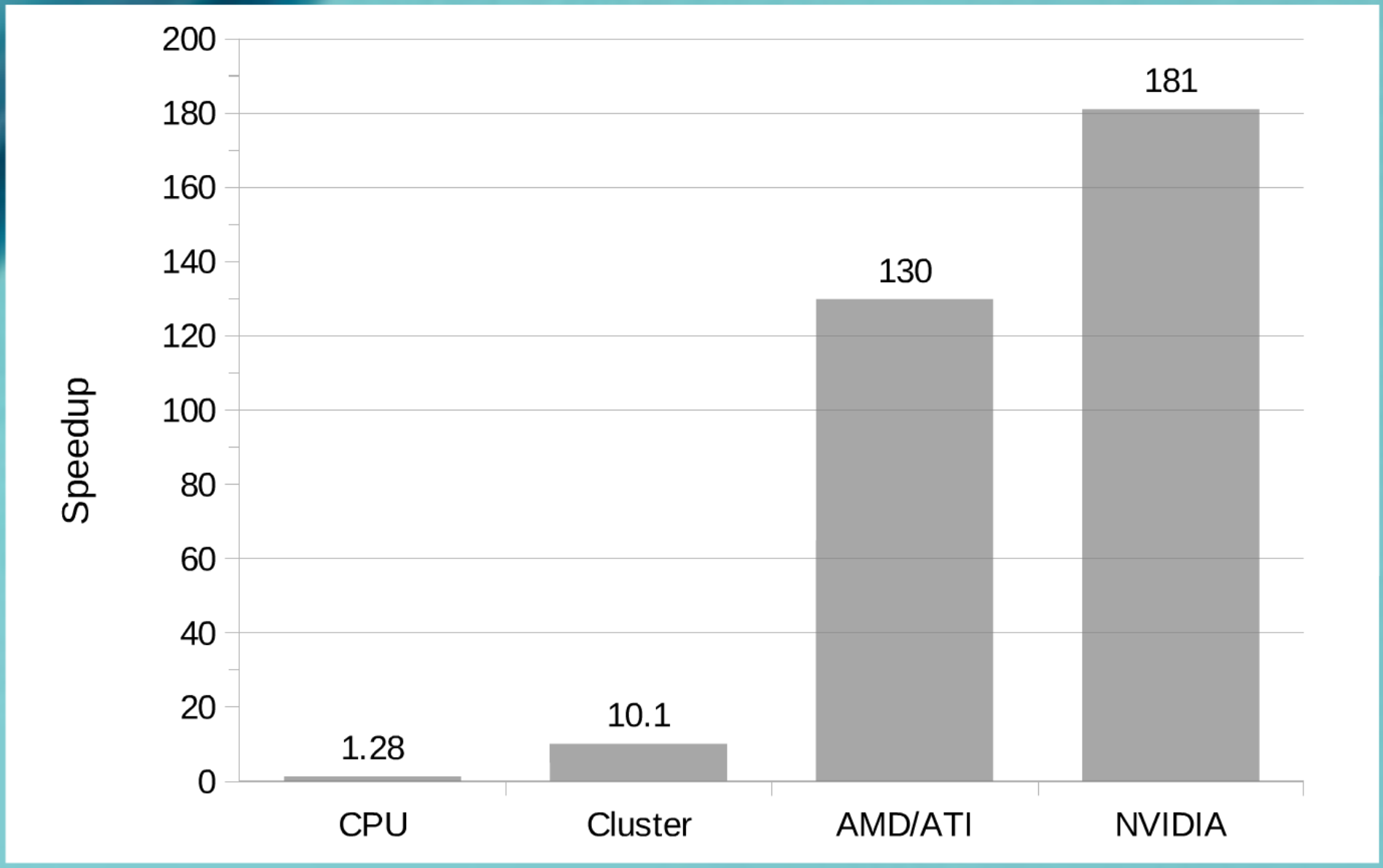
Black-Scholes



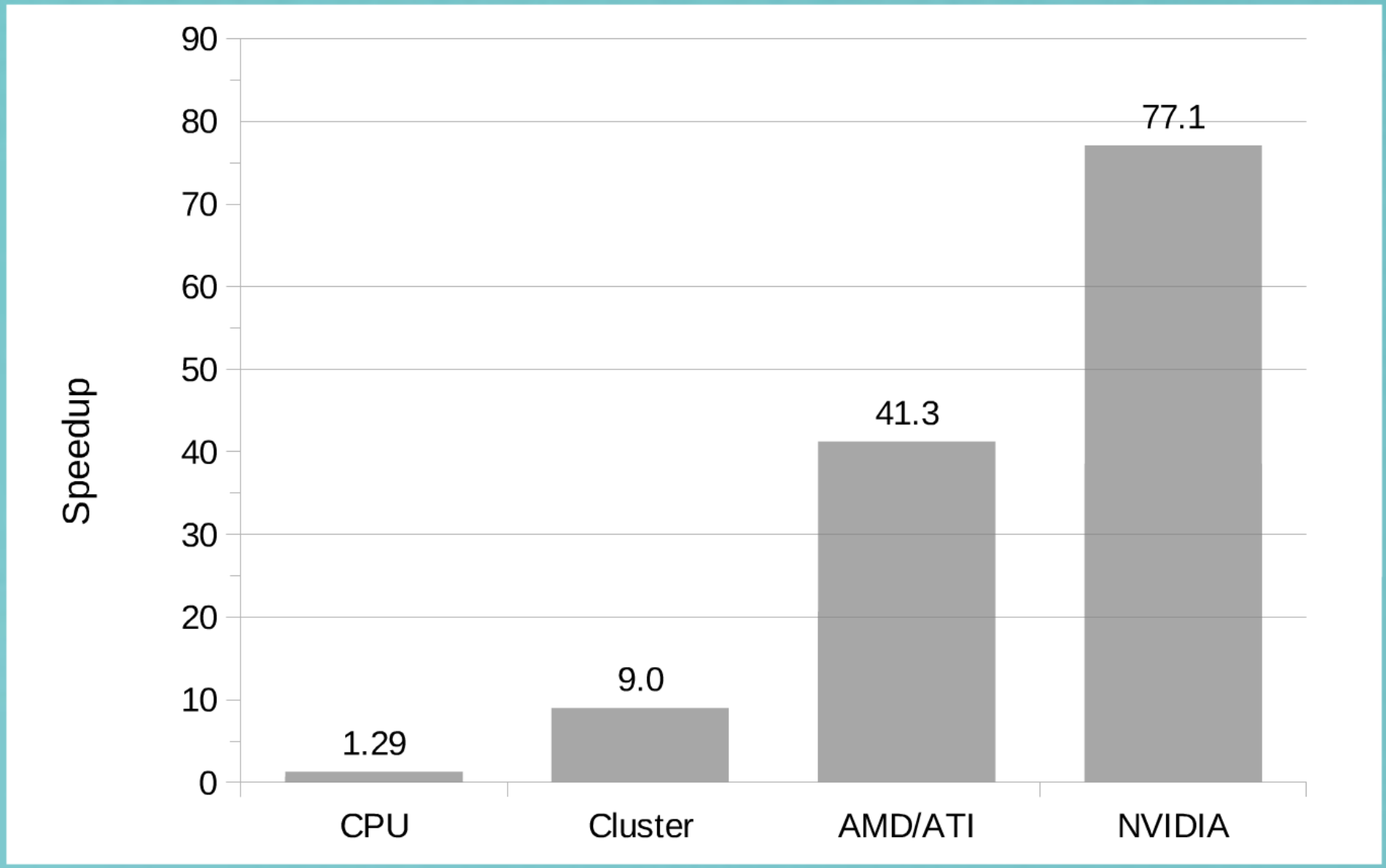
n-body



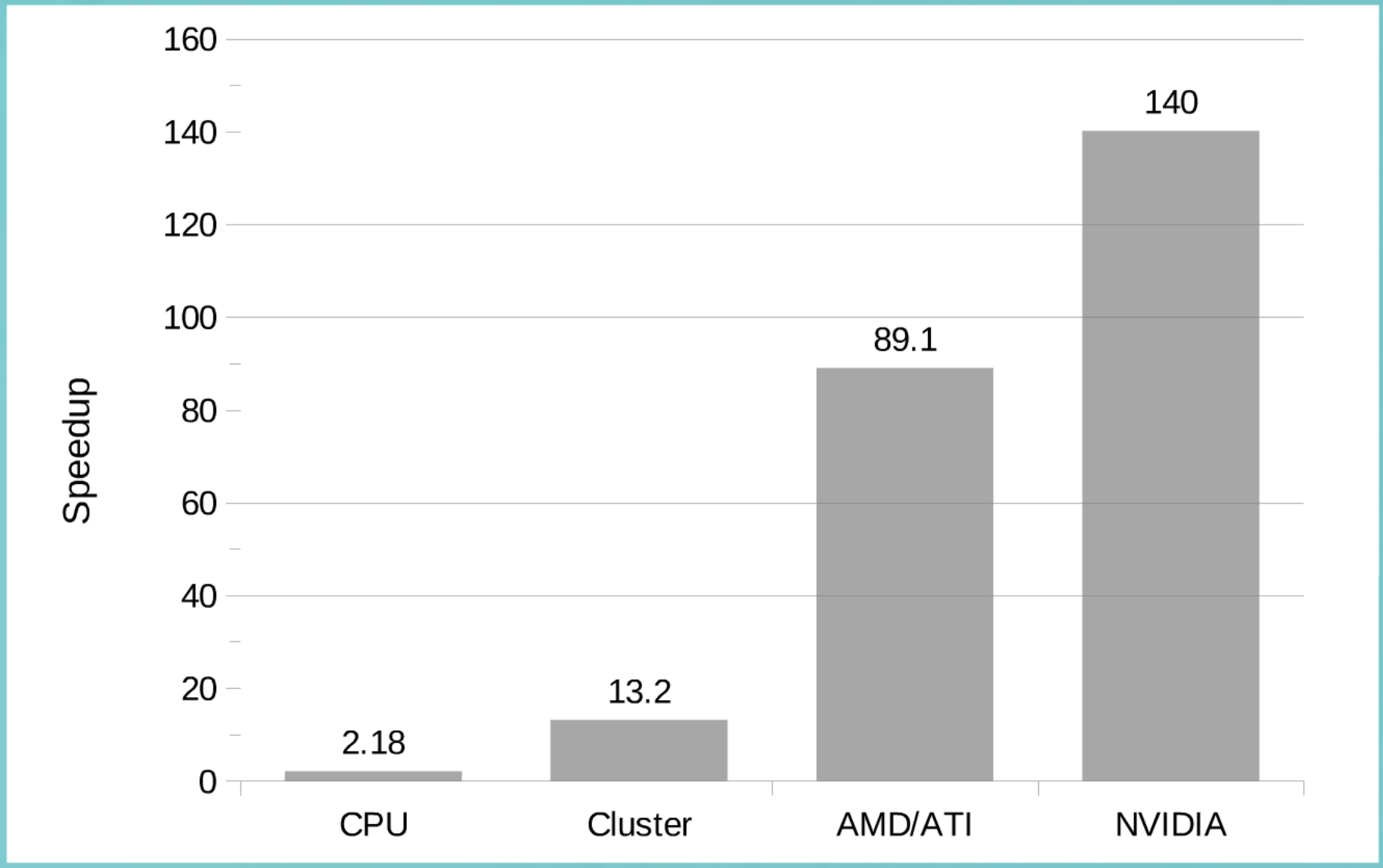
Shallow water



Black-Scholes


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

n-body



Shallow water

Integration with CT toolbox

CT Toolbox has a NumPy version

... but not all operations are supported

These are on the roadmap for
the next Bohrium release

Once completed, any CT Toolbox
application will support more hardware
through Bohrium

Future plans

Full CT Toolbox support

FPGA support in Bohrium

We have most parts of a FPGA based Bohrium
Processor with a simulator

Sparse matrix support

CT Toolbox has a NumPy version

... but not all operations are supported

These are on the roadmap for
the next Bohrium release

Once completed, any CT Toolbox
application will support more hardware
through Bohrium

Future plans

Full CT Toolbox support

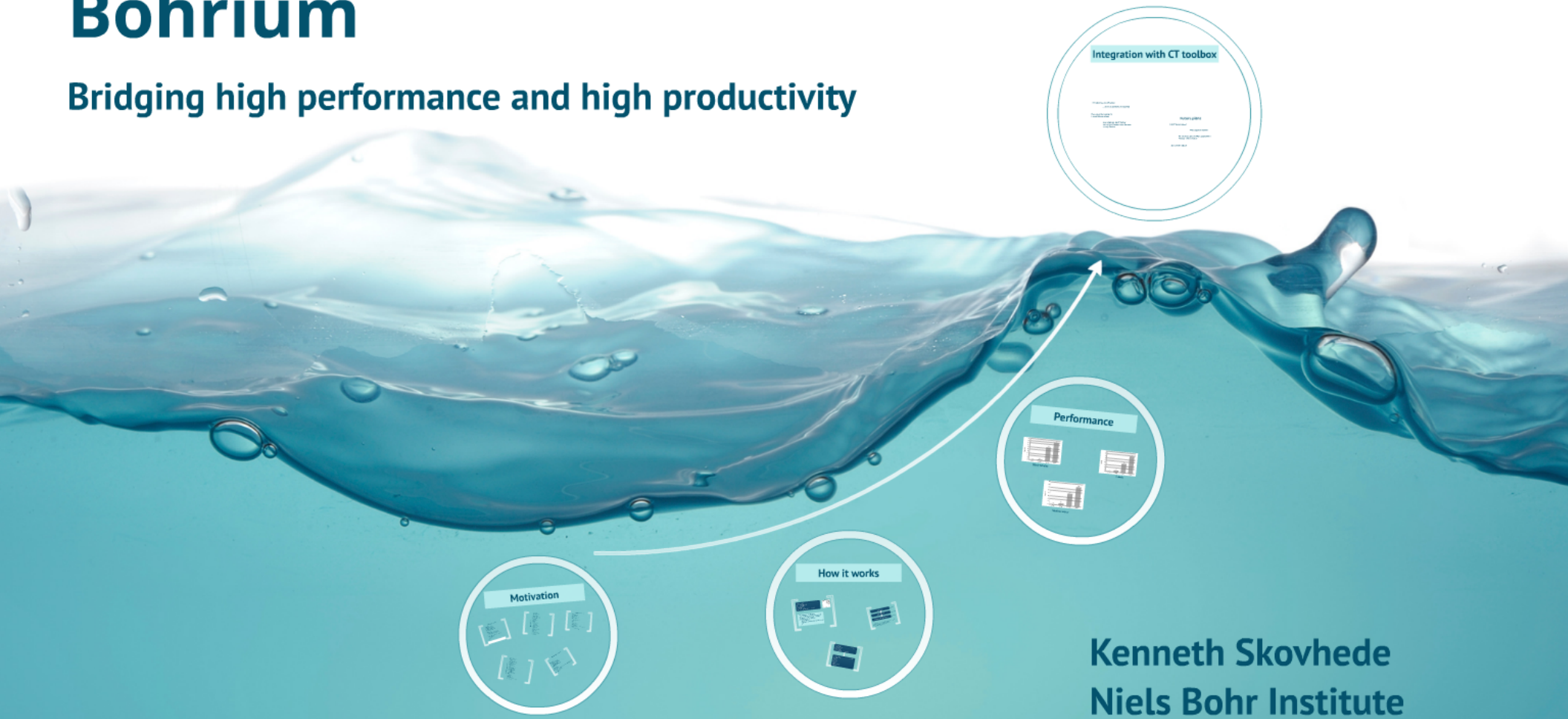
FPGA support in Bohrium

We have most parts of a FPGA based Bohrium Processor with a simulator

Sparse matrix support

Bohrium

Bridging high performance and high productivity



Kenneth Skovhede
Niels Bohr Institute
Workshop 2014-02-13