

Goal

Previous

Current

The End

What

Why

Automatic Mapping of Array Operations to Specific Architectures
 Simon Andrew Fritzsche Lind
 Math E. B. Kressner
 Brian Vinter
 VLSI & Embedded Systems
 Department of Informatics
 University of Cambridge
 November 18, 2014
 VLSI/EMC 2014 in conjunction with SC14

Tools of the trade

Productivity

Performance

Progressive Language

- Languages used by EDA/IC companies: Verilog, SystemVerilog, "Open Project"
- Most of the performance gain is achieved by the hardware
- EDA/IC companies are not interested in high-level programming

Implementation Scope

Array Operations

- Element-wise aka. map, zip operator over arrays()
- Reduction
- Scan

Array Descriptor

Order, Rank, Strides, Step, Mask

$C = A \otimes B$

$W = \ominus(X \otimes Y \otimes Z)$

HIPERFIT HIL

- Improves mathematical models for Finics
- Express them in vendor-specific Languages (DSLs)
- Execute them efficiently on High-Performance Systems

Backend Design Criteria

Language agnostic

Supports programming model for a specific language

Programming Model

- High-level
- Declarative
- Array-oriented

Language integration via intermediate representation

Efficient

- Target performance: comparable to single-frontend compilers like LLVM for its own applications

Question: Is it possible to construct a language agnostic Backend for high-level language without sacrificing performance?

Bohrium

Language Integration

- Map abstractions
- Vector/Strides - intermediate representation

Internal Representation DSL

- Abstracted vector/Strides

Transformations

- Optimization
- Normalization
- Fusion, preserving bytecode sequence

Backend: control flow, memory access, etc.

Task E. B. Kressner, Simon A. Lind, Math E. B. Kressner, Brian Vinter, VLSI & Embedded Systems, Department of Informatics, University of Cambridge, November 18, 2014

Bohrium

CAPE: C-Targeting Array Processing Engine

- Code generation for array operations with parallelization and vectorization of multiple array operations
- Calling JIT-Compiler and offload storage for array operations locally
- Backend: hardware compilation, buffer management, array operation scheduling and execution

Conclusion

Question: Is it possible to construct a language agnostic Backend for high-level language without sacrificing performance?

User knows the high-level array-oriented programming

Knows the configuration of the computing system

Confirms: the high-level array-oriented programming model and its declarative nature provides implicit data-parallel operations and fusions for the backend to decide how to efficiently compute them.

Future / Ongoing Work

Bohrium

- Collaborative effort
- There is even more to it

Bohrium

Reconfigurable: BI_STACK=[cpu,fuser,peony,gru]

Component	Language	Backend	Target
Bohrium	C	OpenMP	Intel Xeon Phi
Bohrium	C	OpenACC	NVIDIA GPUs
Bohrium	C	LED	Intel Xeon Phi
Bohrium	C	OpenMP	Intel Xeon Phi
Bohrium	C	OpenACC	NVIDIA GPUs
Bohrium	C	LED	Intel Xeon Phi

CAPE: C-Targeting Array Processing Engine

Bohrium

Backend: control flow, memory access, etc.

Task E. B. Kressner, Simon A. Lind, Math E. B. Kressner, Brian Vinter, VLSI & Embedded Systems, Department of Informatics, University of Cambridge, November 18, 2014

CAPE: C-Targeting Array Processing Engine

CSI

- OpenMP
- Experimental
- OpenACC
- LED

Encore

CAPE: C-Targeting Array Processing Engine

Codegen specialization

- Flattening
- Array contraction
- Array operation composition
- Array shape => Loop constructs
- Checks buffer-references for aliasing

CAPE: Xeon PHI

Performance

- best case 2x speedup
- often speedupdown

Data management

- device allocation
- transfer to/from device
- data persistence

Codegeneration

- parallelization
- specialization

CAPE: C-Targeting Array Processing Engine

Memory Management

Buffer Management

SIMD utilization

CAPE codegen takes SIMD into consideration, however, current implementation relies on auto-vectorization by the backend Compiler

Result: marginal gain after 90% where commercial compilers prevail

Strategic approach by using pragmas (e.g. simd) - did not yield expected results

Investigate further and possibly expand codegen with manual or explicit means of emitting the complex data reconstruction and/or access

Thanks

CAPE: C-Targeting Array Processing Engine

Backend: control flow, memory access, etc.

Task E. B. Kressner, Simon A. Lind, Math E. B. Kressner, Brian Vinter, VLSI & Embedded Systems, Department of Informatics, University of Cambridge, November 18, 2014

NIELS BOHR INSTITUTE
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN
DENMARK



Automatic Mapping of Array Operations to Specific Architectures

Simon Andreas Frimann Lund
Mads R. B. Kristensen
Brian Vinter

WOLFHPC 2016 in conjunction with SC16

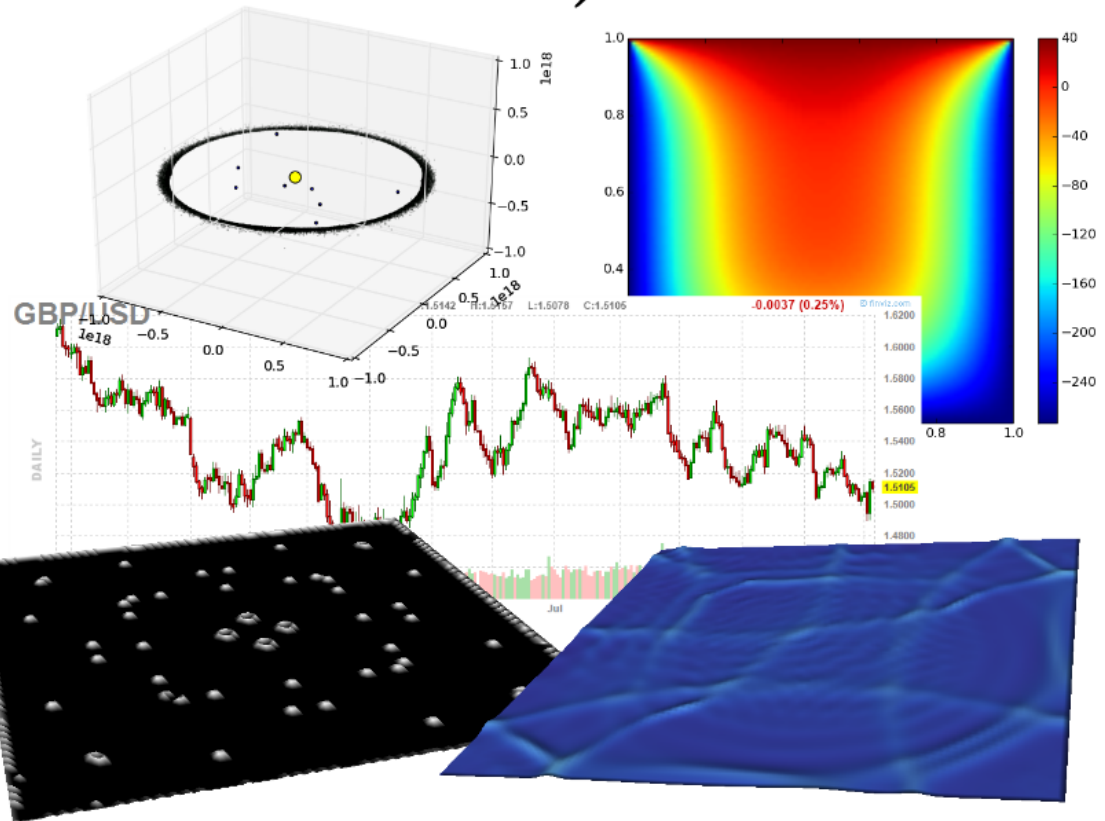
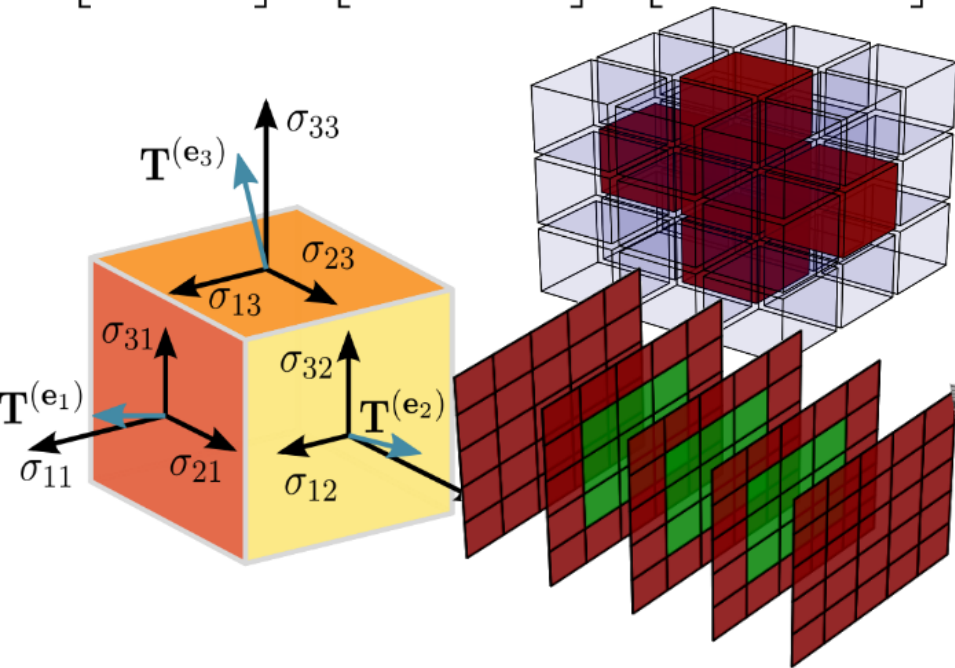
November 13, 2016

$$C = A \otimes B$$

$$C = A \otimes B$$

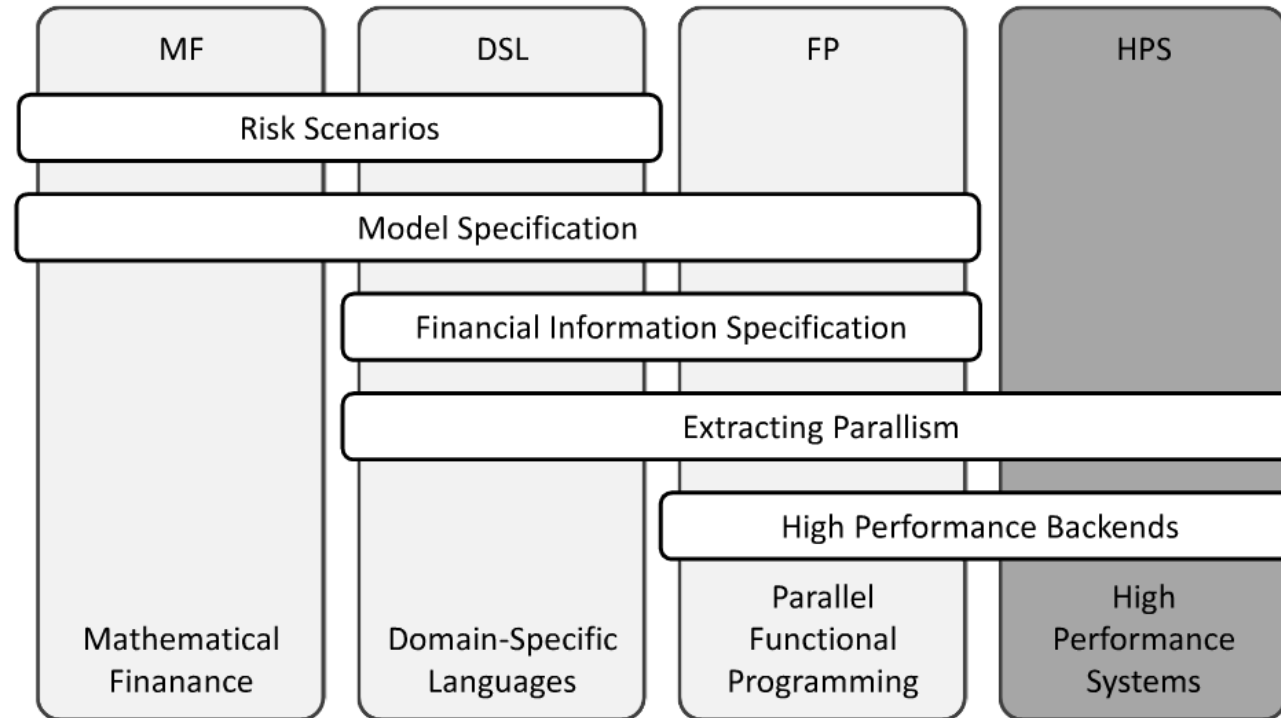
$$W = \ominus(X \otimes Y \oslash Z)$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 1 \\ -2 & 1 & 3 \\ 2 & -5 & 2 \end{bmatrix} + \begin{bmatrix} -2 & -2 & -1 \\ 2 & 0 & -3 \\ -2 & 5 & -1 \end{bmatrix}$$



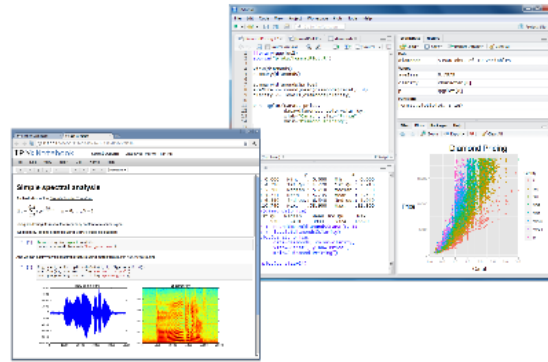
HIPERFIT

Hi.



- Improve mathematical models for Finance
- Express them in verifiable Domain-Specific Languages (DSLs)
- Execute them efficiently on High Performance Systems

Tools of the trade



Productivity

Performance



**A
Programming
Language**



- Languages used by TOP15 Computational Finance / Financial Engineering / "Quant Programs"
<https://www.quantnet.com/mfe-programs-rankings/>
- As well as the University of Copenhagen
- HIPERFIT industry partners with an affinity for APL

#directives

C / C++ / Fortran

OpenMP / pthreads / Qthread

OpenACC / LEO

MPI

OpenCL / CUDA

PGAs



CPUs,

APUs,
Hybrid,
FPGA,



GPUs,
Accelerators,



and clusters of them configured in
shared and distributed memory systems...

Heat Equation in Python / NumPy

```
1 def solve(grid, epsilon):
2     center = grid[1:-1, 1:-1]
3     north = grid[0:-2, 1:-1]
4     south = grid[2: , 1:-1]
5     east = grid[1:-1, 2: ]
6     west = grid[1:-1, 0:-2]
7
8     delta = epsilon + 1
9     while delta > epsilon:
10        work = (center+north+east+west+south)*0.2
11        delta = np.sum(np.absolute(work-center))
12        center[:] = work
```


Heat Equation in C

```
1#include <math.h>
2solve(int size, double *grid, double epsilon)
3{
4    int gsize = size+2; //Size + borders.
5    double *T = malloc(gsize*gsize*sizeof(double));
6    double delta = epsilon+1;
7    while(delta > epsilon)
8    {
9        double *a = grid;
10       double *t = T;
11       delta = 0;
12       for(i=0; i<size; ++i)
13       {
14           double *up      = a+1;
15           double *left    = a+gsize;
16           double *right   = a+gsize+2;
17           double *down    = a+1+gsize*2;
18           double *center  = a+gsize+1;
19           double *t_center = t+gsize+1;
20           for(j=0; j<size; ++j)
21           {
22               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
23               delta += fabs(t_center-center);
24           }
25           a += gsize;
26           t += gsize;
27       }
28       memcpy(A, T, gsize*gsize*sizeof(double));
29     }
30 }
```

Heat Equation in C and OpenMP

```
1 #include <math.h>
2 solve(int size, double *grid, double epsilon)
3 {
4     int gsize = size+2; //Size + borders.
5     double *T = malloc(gsize*gsize*sizeof(double));
6     double delta = epsilon+1;
7     while(delta > epsilon)
8     {
9         delta = 0;
10        #pragma omp parallel for shared(grid,T) reduction(+:delta)
11        for(i=0; i<size; ++i)
12        {
13            int a = i * gsize;
14            double *up      = &grid[a+1];
15            double *left    = &grid[a+gsize];
16            double *right   = &grid[a+gsize+2];
17            double *down    = &grid[a+1+gsize*2];
18            double *center  = &grid[a+gsize+1];
19            double *t_center = &T[a+gsize+1];
20            for(j=0; j<size; ++j)
21            {
22                *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
23                delta += fabs(t_center-center);
24            }
25        }
26        memcpy(grid, T, gsize*gsize*sizeof(double));
27    }
28 }
```

Heat Equation in C and OpenMP

```

1#include <math.h>
2solve(int size, double *grid, double epsilon)
3{
4    int gsize = size+2; //Size + borders.
5    double *T = malloc(gsize*gsize*sizeof(double));
6    double delta = epsilon+1;
7    while(delta > epsilon)
8    {
9        delta = 0;
10       #pragma omp parallel for shared(grid,T) reduction(+:delta)
11       for(i=0; i<size; ++i)
12       {
13           int a = i * gsize;
14           double *up = &grid[a+1];
15           double *left = &grid[a+gsize];
16           double *right = &grid[a+gsize+2];
17           double *down = &grid[a+1+gsize*2];
18           double *center = &grid[a+gsize+1];
19           double *t_center = &T[a+gsize+1];
20           for(j=0; j<size; ++j)
21           {
22               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
23               delta += fabs(t_center+center);
24           }
25       }
26       memcpy(grid, T, gsize*gsize*sizeof(double));
27     }
28 }

```

Heat Equation in C and OpenMP and MPI

```

1#include <math.h>
2#include <mpi.h>
3solve(int size, double *grid, double epsilon)
4{
5    int gsize = SIZE+2; //Size + borders.
6    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
7    MPI_Comm_size(MPI_COMM_WORLD, &worldsize);
8    MPI_Comm comm;
9    int periods[] = {0};
10   MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize,
11                 periods, 1, &comm);
12   int l_size = SIZE / worldsize;
13   if(myrank == worldsize-1)
14       l_size += SIZE % worldsize;
15   int l_gsize = l_size + 2; //Size + borders.
16
17   double delta = epsilon+1;
18   while(delta > epsilon)
19   {
20
21       int p_src, p_dest;
22       //Send/receive - neighbor above
23       MPI_Cart_shift(comm, 0, 1, &p_src, &p_dest);
24       MPI_Sendrecv(grid+gsize, gsize, MPI_DOUBLE,
25                  p_dest, 1, grid, gsize, MPI_DOUBLE,
26                  p_src, 1, comm, MPI_STATUS_IGNORE);
27       //Send/receive - neighbor below
28       MPI_Cart_shift(comm, 0, -1, &p_src, &p_dest);
29       MPI_Sendrecv(grid+(l_gsize-2)*gsize,
30                  gsize, MPI_DOUBLE,
31                  p_dest, 1, grid+(l_gsize-1)*gsize,
32                  gsize, MPI_DOUBLE,
33                  p_src, 1, comm, MPI_STATUS_IGNORE);
34
35       delta = 0;
36       #pragma omp parallel for shared(grid,T) reduction(+:delta)
37       for(i=0; i<size; ++i)
38       {
39           int a = i * gsize;
40           double *up = &grid[a+1];
41           double *left = &grid[a+gsize];
42           double *right = &grid[a+gsize+2];
43           double *down = &grid[a+1+gsize*2];
44           double *center = &grid[a+gsize+1];
45           double *t_center = &T[a+gsize+1];
46           for(j=0; j<size; ++j)
47           {
48               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
49               delta += fabs(t_center+center);
50           }
51       }
52       memcpy(grid, T, gsize*gsize*sizeof(double));
53       MPI_Allreduce(&delta, &delta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
54   }
55 }

```

Heat Equation in C and OpenMP and MPI with Latency Hiding

Heat Equation in C and OpenMP and MPI

```
1#include <math.h>
2#include <mpi.h>
3solve(int size, double *grid, double epsilon)
4{
5    int gsize = SIZE+2; //Size + borders.
6    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
7    MPI_Comm_size(MPI_COMM_WORLD, &worldsize);
8    MPI_Comm comm;
9    int periods[] = {0};
10   MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize,
11                  periods, 1, &comm);
12   int l_size = SIZE / worldsize;
13   if(myrank == worldsize-1)
14       l_size += SIZE % worldsize;
15   int l_gsize = l_size + 2;//Size + borders.
16
17   double delta = epsilon+1;
18   while(delta > epsilon)
19   {
20
21       int p_src, p_dest;
22       //Send/receive - neighbor above
23       MPI_Cart_shift(comm,0,1,&p_src,&p_dest);
24       MPI_Sendrecv(grid+gsize,gsize,MPI_DOUBLE,
25                   p_dest,1,grid,gsize,MPI_DOUBLE,
26                   p_src,1,comm,MPI_STATUS_IGNORE);
27       //Send/receive - neighbor below
28       MPI_Cart_shift(comm,0,-1,&p_src,&p_dest);
29       MPI_Sendrecv(grid+(l_gsize-2)*gsize,
30                   gsize,MPI_DOUBLE,
31                   p_dest,1,grid+(l_gsize-1)*gsize,
32                   gsize,MPI_DOUBLE,
33                   p_src,1,comm,MPI_STATUS_IGNORE);
34
35       delta = 0;
36       #pragma omp parallel for shared(grid,T) reduction(+:delta)
37       for(i=0; i<size; ++i)
38       {
39           int a = i * gsize;
40           double *up = &grid[a+1];
41           double *left = &grid[a+gsize];
42           double *right = &grid[a+gsize+2];
43           double *down = &grid[a+1+gsize*2];
44           double *center = &grid[a+gsize+1];
45           double *t_center = &T[a+gsize+1];
46           for(j=0; j<size; ++j)
47           {
48               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
49               delta += fabs(t_center+center);
50           }
51       }
52       memcpy(grid, T, gsize*gsize*sizeof(double));
53       MPI_Allreduce(&delta, &delta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
54   }
```

```
1#include <math.h>
2#include <mpi.h>
3solve(int size, double *grid, double epsilon)
4{
5    int gsize = SIZE+2; //Size + borders.
6    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
7    MPI_Comm_size(MPI_COMM_WORLD, &worldsize);
8    MPI_Comm comm;
9    int periods[] = {0};
10   MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize,
11                  periods, 1, &comm);
12   int l_size = SIZE / worldsize;
13   if(myrank == worldsize-1)
14       l_size += SIZE % worldsize;
15   int l_gsize = l_size + 2;//Size + borders.
16
17   double delta = epsilon+1;
18   while(delta > epsilon)
19   {
20
21       int p_src, p_dest;
22       MPI_Request reqs[4];
23       //Initiate send/receive - neighbor above
24       MPI_Cart_shift(comm, 0, 1, &p_src, &p_dest);
25       MPI_Isend(grid+gsize, gsize, MPI_DOUBLE, p_dest,
26                1, comm, &reqs[0]);
27       MPI_Irecv(grid, gsize, MPI_DOUBLE, p_src,
28                1, comm, &reqs[1]);
29       //Initiate send/receive - neighbor below
30       MPI_Cart_shift(comm, 0, -1, &p_src, &p_dest);
31       MPI_Isend(grid+(l_gsize-2)*gsize, gsize,
32                MPI_DOUBLE,
33                p_dest, 1, comm, &reqs[2]);
34       MPI_Irecv(grid+(l_gsize-1)*gsize, gsize,
35                MPI_DOUBLE,
36                p_src, 1, comm, &reqs[3]);
37       //Handle the non-border elements.
38       delta = 0;
39       #pragma omp parallel for shared(grid,T) reduction(+:delta)
40       for(i=0; i<size-1; ++i)
41       {
42           int a = i * gsize;
43           double *up = &grid[a+1];
44           double *left = &grid[a+gsize];
45           double *right = &grid[a+gsize+2];
46           double *down = &grid[a+1+gsize*2];
47           double *center = &grid[a+gsize+1];
48           double *t_center = &T[a+gsize+1];
49           for(j=0; j<size-1; ++j)
50           {
51               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
52               delta += fabs(t_center+center);
53           }
54       }
55       //Handle the upper ghost line
56       MPI_Waitall(2, reqs, MPI_STATUSES_IGNORE);
57       {
58           int a = 0 * gsize;
59           double *up = &grid[a+1];
60           double *left = &grid[a+gsize];
61           double *right = &grid[a+gsize+2];
62           double *down = &grid[a+1+gsize*2];
63           double *center = &grid[a+gsize+1];
64           double *t_center = &T[a+gsize+1];
65           for(j=0; j<size-1; ++j)
66           {
67               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
68               delta += fabs(t_center+center);
69           }
70       }
71       //Handle the lower ghost line
72       MPI_Waitall(2, reqs+2, MPI_STATUSES_IGNORE);
73       {
74           int a = (size-1) * gsize;
75           double *up = &grid[a+1];
76           double *left = &grid[a+gsize];
77           double *right = &grid[a+gsize+2];
78           double *down = &grid[a+1+gsize*2];
79           double *center = &grid[a+gsize+1];
80           double *t_center = &T[a+gsize+1];
81           for(j=0; j<size-1; ++j)
82           {
83               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
84               delta += fabs(t_center+center);
85           }
86       }
87       memcpy(grid, T, gsize*gsize*sizeof(double));
88       MPI_Allreduce(&delta, &delta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
89   }
```

Heat Equation in C and OpenMP and MPI with Latency Hiding

```
1#include <math.h>
2#include <mpi.h>
3solve(int size, double *grid, double epsilon)
4{
5    int gsize = SIZE+2; //Size + borders.
6    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
7    MPI_Comm_size(MPI_COMM_WORLD, &worldsize);
8    MPI_Comm comm;
9    int periods[] = {0};
10   MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize,
11                  periods, 1, &comm);
12   int l_size = SIZE / worldsize;
13   if(myrank == worldsize-1)
14       l_size += SIZE % worldsize;
15   int l_gsize = l_size + 2; //Size + borders.
16
17   double delta = epsilon+1;
18   while(delta > epsilon)
19   {
20
21       int p_src, p_dest;
22       MPI_Request reqs[4];
23       //Initiate send/recv - neighbor above
24       MPI_Cart_shift(comm, 0, 1, &p_src, &p_dest);
25       MPI_Isend(grid+gsize, gsize, MPI_DOUBLE, p_dest,
26                1, comm, &reqs[0]);
27       MPI_Irecv(grid, gsize, MPI_DOUBLE, p_src,
28                1, comm, &reqs[1]);
29       //Initiate send/recv - neighbor below
30       MPI_Cart_shift(comm, 0, -1, &p_src, &p_dest);
31       MPI_Isend(grid+(l_gsize-2)*gsize, gsize,
32                 MPI_DOUBLE,
33                 p_dest, 1, comm, &reqs[2]);
34       MPI_Irecv(grid+(l_gsize-1)*gsize, gsize,
35                 MPI_DOUBLE,
36                 p_src, 1, comm, &reqs[3]);
37       //Handle the non-border elements.
38       delta = 0;
39       #pragma omp parallel for shared(grid,T) reduction(+:delta)
40       for(i=0; i<size-1; ++i)
41       {
42           int a = i * gsize;
43           double *up = &grid[a+1];
44           double *left = &grid[a+gsize];
45           double *right = &grid[a+gsize+2];
46           double *down = &grid[a+1+gsize*2];
47           double *center = &grid[a+gsize+1];
48           double *t_center = &T[a+gsize+1];
49           for(j=0; j<size-1; ++j)
50           {
51               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
52               delta += fabs(t_center+center);
53           }
54       }
55       //Handle the upper ghost line
56       MPI_Waitall(2, reqs, MPI_STATUSES_IGNORE);
57       {
58           int a = 0 * gsize;
59           double *up = &grid[a+1];
60           double *left = &grid[a+gsize];
61           double *right = &grid[a+gsize+2];
62           double *down = &grid[a+1+gsize*2];
63           double *center = &grid[a+gsize+1];
64           double *t_center = &T[a+gsize+1];
65           for(j=0; j<size-1; ++j)
66           {
67               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
68               delta += fabs(t_center+center);
69           }
70       }
71       //Handle the lower ghost line
72       MPI_Waitall(2, reqs+2, MPI_STATUSES_IGNORE);
73       {
74           int a = (size-1) * gsize;
75           double *up = &grid[a+1];
76           double *left = &grid[a+gsize];
77           double *right = &grid[a+gsize+2];
78           double *down = &grid[a+1+gsize*2];
79           double *center = &grid[a+gsize+1];
80           double *t_center = &T[a+gsize+1];
81           for(j=0; j<size-1; ++j)
82           {
83               *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
84               delta += fabs(t_center+center);
85           }
86       }
87       memcpy(grid, T, gsize*gsize*sizeof(double));
88       MPI_Allreduce(&delta, &delta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
89   }
```

Heat Equation in Python / NumPy

```
1 def solve(grid, epsilon):
2     center = grid[1, 1, 1]
3     north = grid[0, 2, 1]
4     south = grid[2, 1, 1]
5     east = grid[1, 2, 1]
6     west = grid[1, 0, 2]
7
8     delta = epsilon + 1
9     while delta > epsilon:
10        work = (center+north+east+west+south)*0.2
11        delta = np.sum(np.absolute(work-center))
12        center[:] = work
```

Heat Equation in Python / NumPy

```
1 def solve(grid, epsilon):
2     center = grid[1:-1, 1:-1]
3     north = grid[0:-2, 1:-1]
4     south = grid[2: , 1:-1]
5     east = grid[1:-1, 2: ]
6     west = grid[1:-1, 0:-2]
7
8     delta = epsilon + 1
9     while delta > epsilon:
10        work = (center+north+east+west+south)*0.2
11        delta = np.sum(np.absolute(work-center))
12        center[:] = work
```

Programming Pitfalls

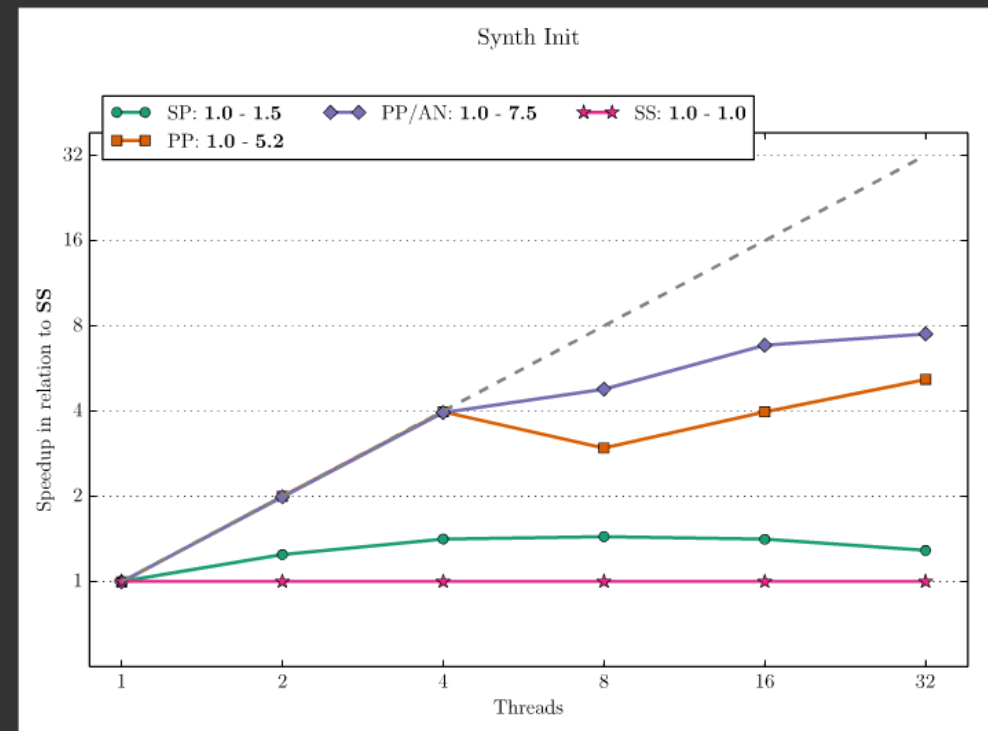


Performance

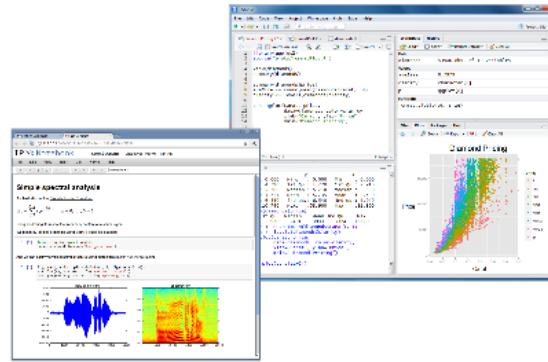


Correctness

- Deadlocks
- Race-conditions



Tools of the trade



Productivity

Performance



**A
Programming
Language**



- Languages used by TOP15 Computational Finance / Financial Engineering / "Quant Programs"
<https://www.quantnet.com/mfe-programs-rankings/>
- As well as the University of Copenhagen
- HIPERFIT industry partners with an affinity for APL



Backend Design Criteria

Language agnostic

Support a programming model not a specific language

Programming Model

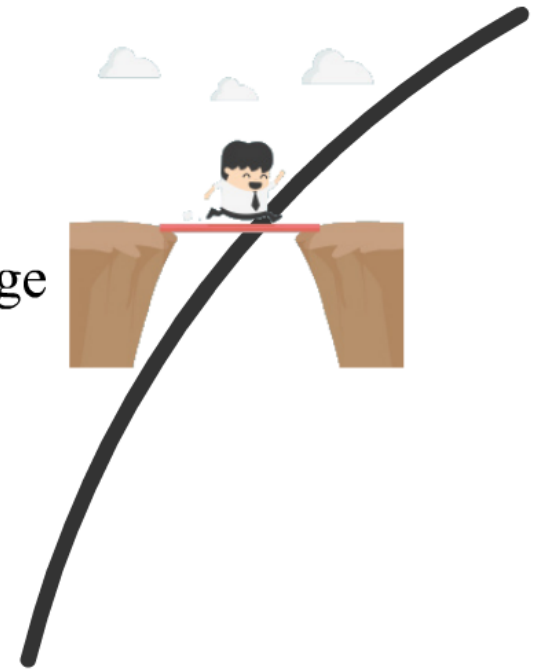
- High-level
- Declarative
- **Array-oriented**

Language integration via intermediate representation

Efficient

Target a performance comparable to straight forward hand-coded C/C++ for the same application

Question: *Is it possible to construct a language agnostic back for high-level languages without sacrificing performance?*

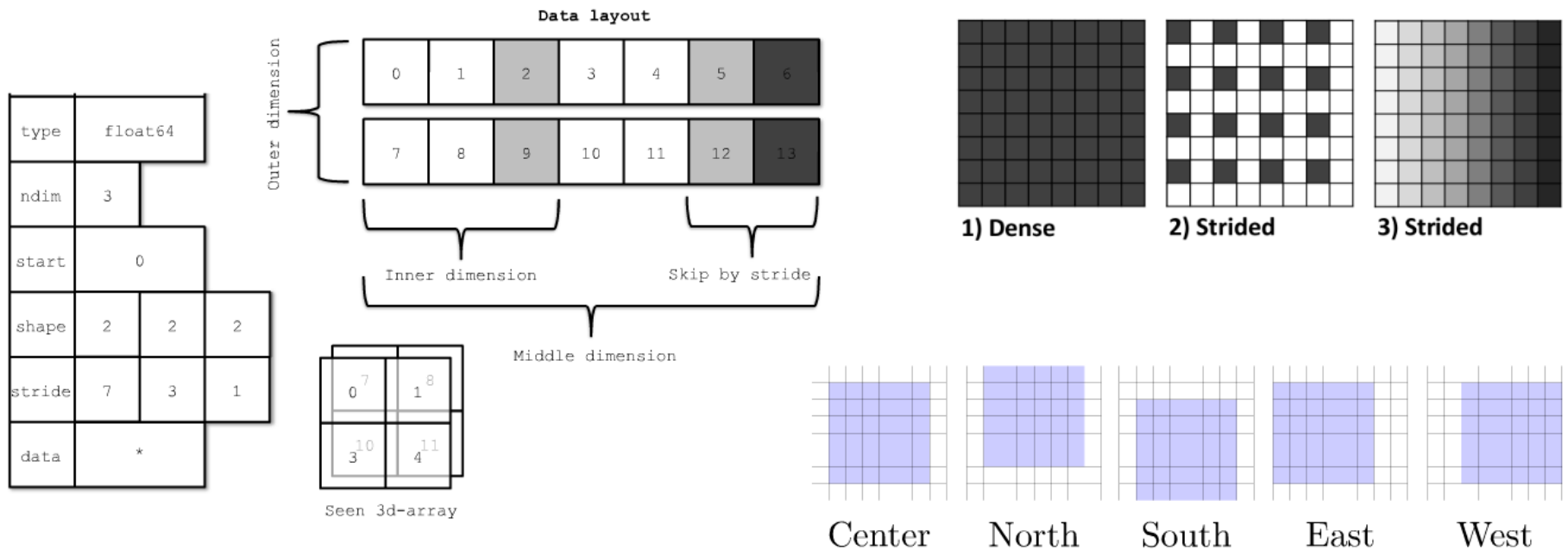


Implementation Scope

Array Operations

- Element-wise aka *map*, *zip* operator over array(s)
- Reduction
- Scan

Array Descriptor





Language Integration

- Map abstractions
- *vector bytecode* - intermediate representation

Internal Representation BhIr

- Annotated *vector bytecode*

Transformations

- Optimization
- Normalization
- Fusion, grouping bytecode sequences

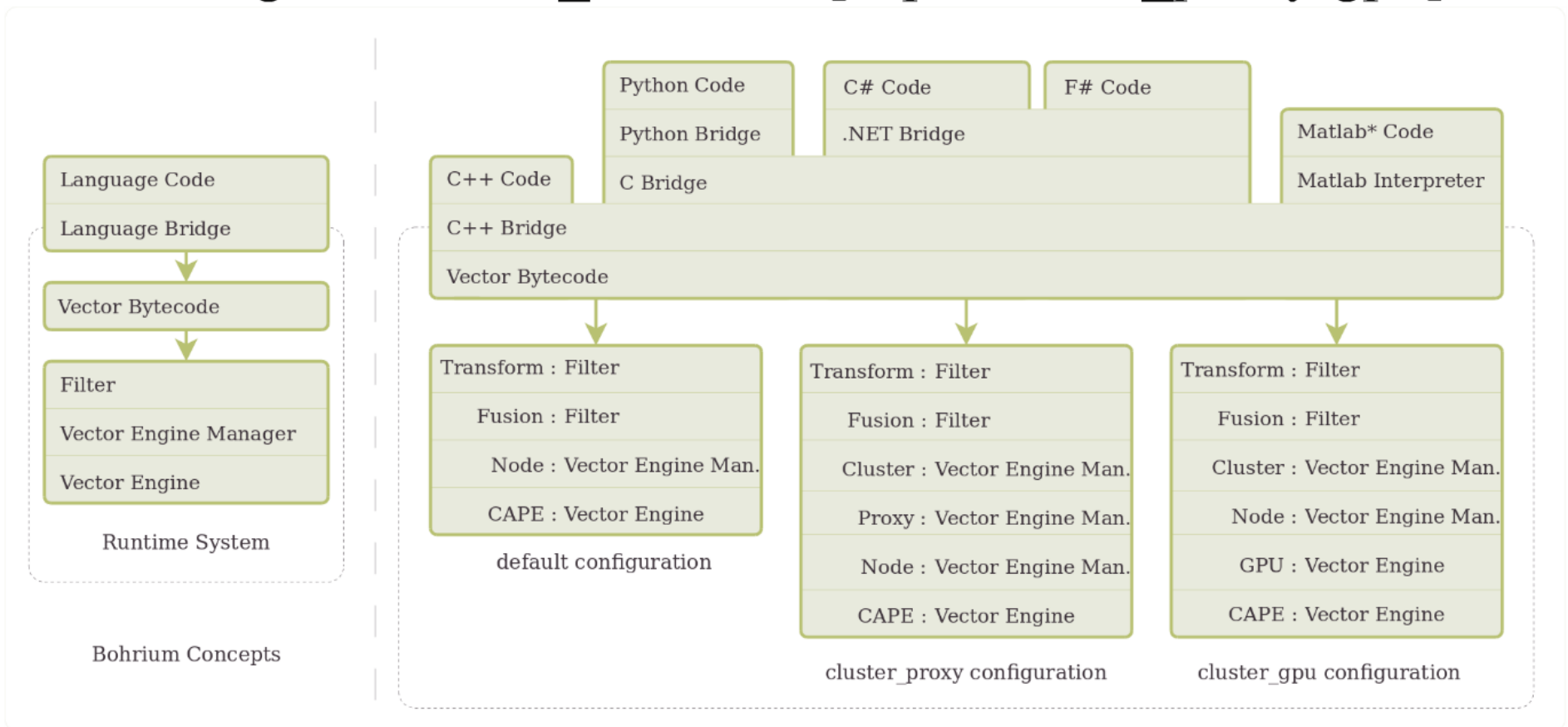
Bohrium: a virtual machine approach to portable parallelism

Mads R.B. Kristensen, Simon A.F. Lund, Troels Blum, Kenneth Skovhede, Brian Vinter.

In proceedings of the Parallel & Distributed Processing Symposium Workshops (IPDPSW14)



Reconfigurable: BH_STACK=[cape,cluster_proxy,gpu]





Array Operation Fusion

```
#define N 1000
double a[N], b[N], T[N];
// Array expression: A += B * A
for (int i=0; i<N; ++i)
  T[i] = B[i] * A[i];
for (int i=0; i<N; ++i)
  A[i] += T[i];
```

```
for (int i=0; i<N; ++i) {
  T[i] = B[i] * A[i];
  A[i] += T[i];
}
```

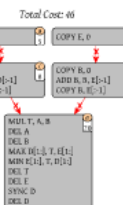
```
for (int i=0; i<N; ++i) {
  double t = B[i] * A[i];
  A[i] += t;
}
```

Fusion Fail

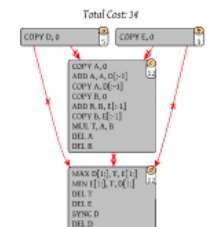
```
#define N 1000
double a[N], b[N], r[N];
int i = 0;
// Array expression: A += reverse(B * A)
for (int i=0; i<N; ++i)
  r[i] = B[i] * A[i];
for (int i=0; i<N; ++i)
  A[i] += r[N-i];
```



Greedy



Optimal



Fusion of Parallel Array Operations

Mads R.B. Kristensen, Simon A.F. Lund, Troels Blum, James Avery.

In proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT16).

Array Operation Fusion

```
#define N 1000
double A[N], B[N], T[N];
// Array expression: A += B*A
for(int i=0; i<N; ++i)
    T[i] = B[i] * A[i];
for(int i=0; i<N; ++i)
    A[i] += T[i];
```

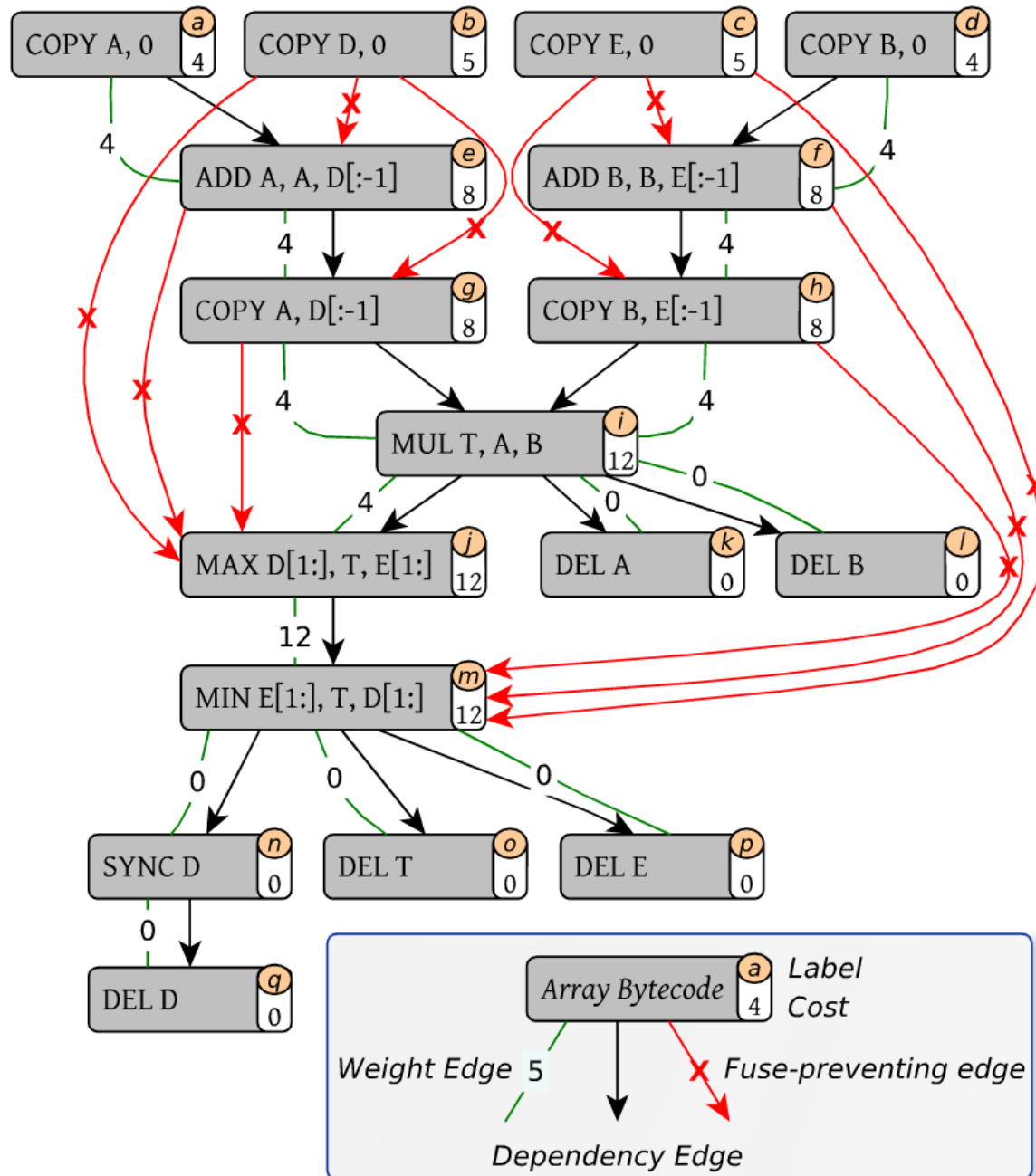
```
for(int i=0; i<N; ++i){
    T[i] = B[i] * A[i];
    A[i] += T[i];
}
```

```
for(int i=0; i<N; ++i){
    double t = B[i] * A[i];
    A[i] += t;
}
```

Fusion Fail

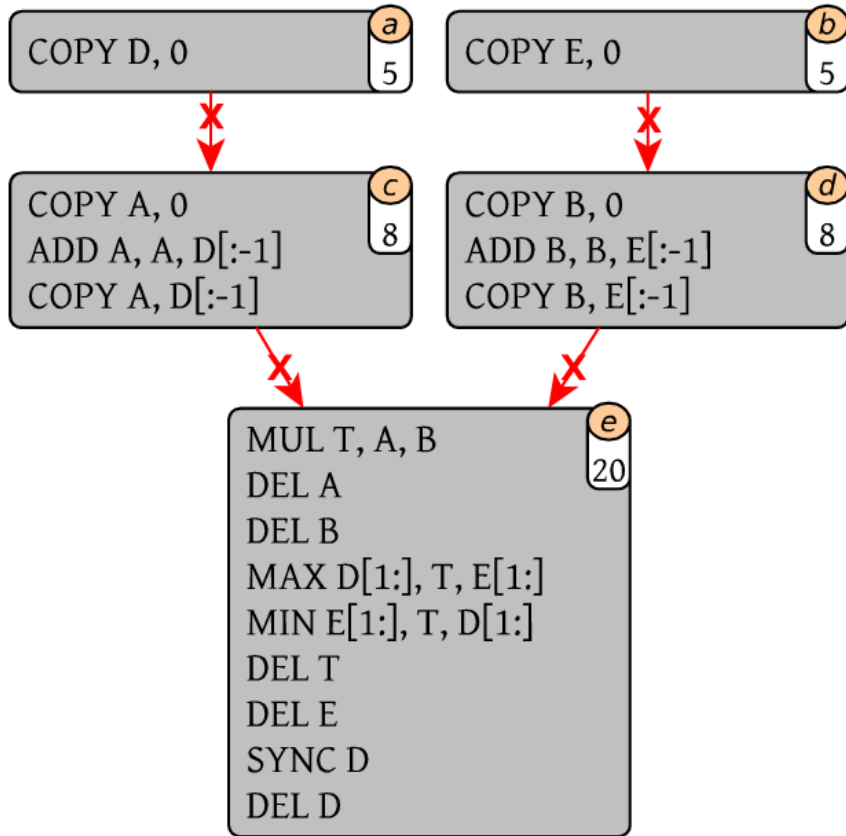
```
#define N 1000
double A[N], B[N], T[N];
int j = N;
// Array expression: A += reverse(B * A)
for(int i=0; i<N; ++i)
    T[i] = B[i] * A[i];
for(int i=0; i<N; ++i)
    A[i] += T[--j];
```

Total Partition Cost: 86



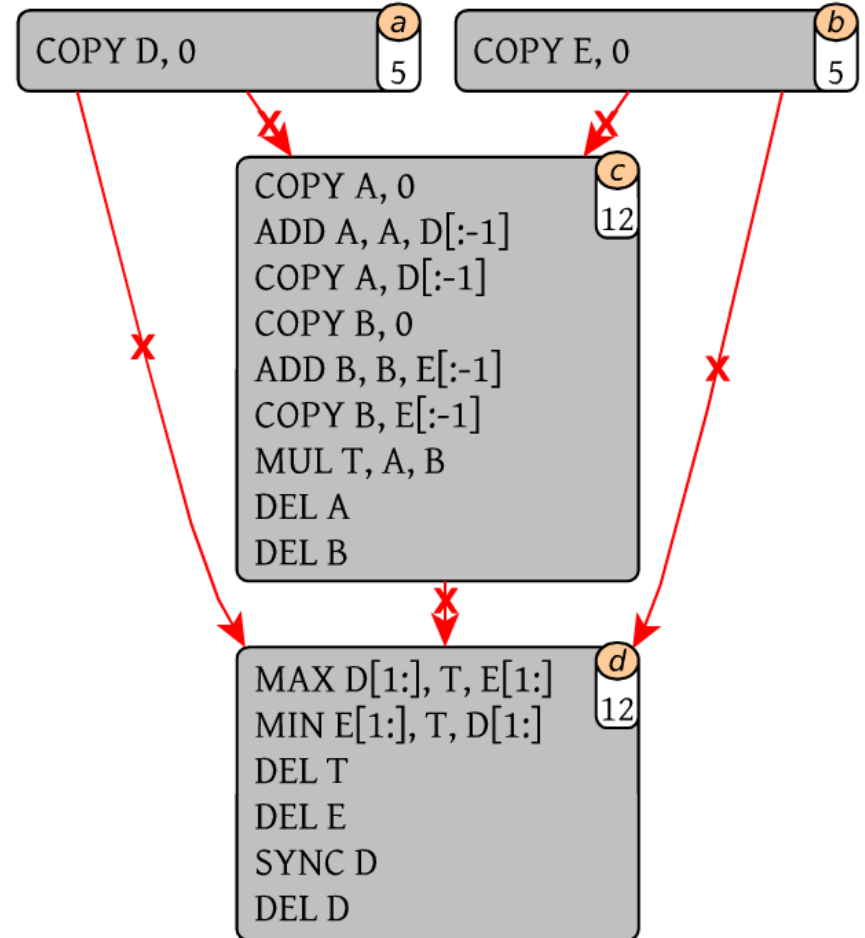
Greedy

Total Cost: 46



Optimal

Total Cost: 34





Array Operation Fusion

```
#define N 1000
double a[N], b[N], t[N];
// Array expression: A += B * A
for (int i=0; i<N; ++i)
  t[i] = b[i] * a[i];
for (int i=0; i<N; ++i)
  a[i] += t[i];
```

```
for (int i=0; i<N; ++i) {
  t[i] = b[i] * a[i];
  a[i] += t[i];
}
```

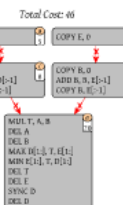
```
for (int i=0; i<N; ++i) {
  double t = b[i] * a[i];
  a[i] += t;
}
```

Fusion Fail

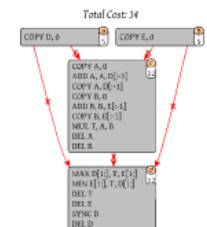
```
#define N 1000
double a[N], b[N], t[N];
int i = 0;
// Array expression: A += reverse(B * A)
for (int i=0; i<N; ++i)
  t[i] = b[i] * a[i];
for (int i=0; i<N; ++i)
  a[i] += t[N-i];
```



Greedy



Optimal



Fusion of Parallel Array Operations

Mads R.B. Kristensen, Simon A.F. Lund, Troels Blum, James Avery.

In proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT16).

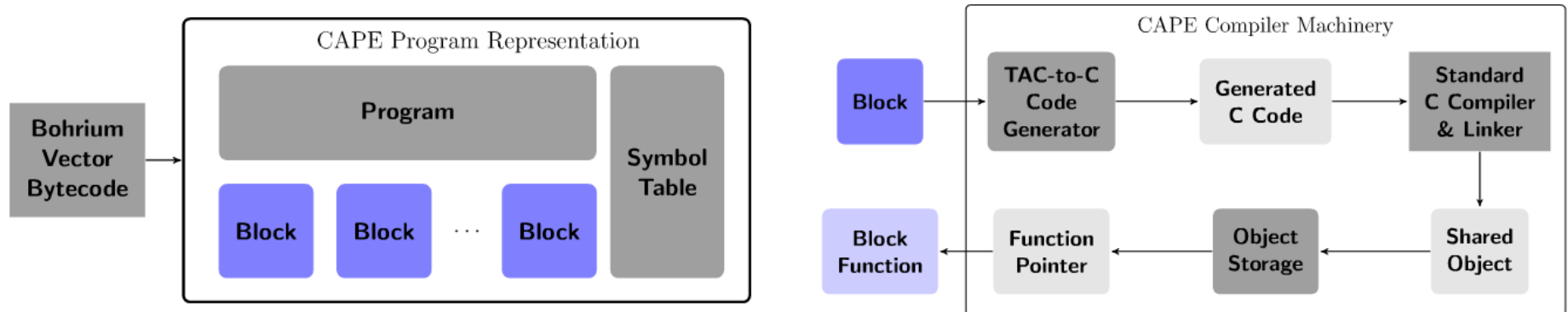


CAPE: C-Targeting Array Processing Engine

- Code generator for array operations with parallelization and composition of multiple array operations
- Caching JIT-Compiler and object storage for array operation kernels
- Runtime instrumenting compilation, buffer management, array operation scheduling and execution

CAPE: C-Targeting Array Processing Engine

```
1 process(bytecode) {
2
3   program, symbol_table, blocks[] = map(
4     bytecode
5   );
6
7   thread_manager.bind();
8
9   for block in blocks {
10    block_function = jit_compiler(block);
11    memory_manager(block); // Allocation
12    block_function(block); // Execution
13    memory_manager(block); // De-allocation
14  }
15 }
```



CAPE: C-Targeting Array Processing Engine

C99

- OpenMP

Experimental

- OpenACC
- LEO

```
1 void KP_IDENTHASH(kp_buffer** buffers ,
2                   kp_operand** operands ,
3                   kp_iterspace* iterspace)
4 {
5     // Buffers (data pointers)
6     double* buf0_data = buffers[0]->data;
7     ...
8
9     // Operands (strides and buffer offset)
10    const int64_t opd0_start = operands[0]->start;
11
12    // Iterspace (shape, layout, #elements)
13    const int64_t iterspace_nelem = iterspace->nelem;
14    ...
15
16    // Parallel section - prequel
17
18    { // Parallel section - entry
19
20        // - operand to buffer mapping
21        double* restrict opd0 = buf0_data + opd0_start;
22        double opd1;
23        // - work distribution
24        // - initialize accumulator variables
25
26        // Parallel section - loop constructs
27        {
28            ... array operations ...
29        }
30        // Parallel section - exit
31    }
32
33    // Parallel section - sequel
34 }
```

CAPE: C-Targeting Array Processing Engine

Codegen specialization

- Flattening
- Array contraction
- Array operation composition
- Array shape \Rightarrow Loop constructs
- Checks buffer-references for aliasing

Memory Management

Allocation and de-allocation of buffers backing array storage

Alignment

GPUs and Accelerators

- data transfer: host <-> device
- data persistence: buffer-reuse on device

Software Victim cache

- Delay de-allocation
- Reuse buffers

```
t1 = malloc();
ufunc_add(t1, center, north)

t2 = malloc()
ufunc_add(t2, t1, east)
free(t1)

t3 = malloc()
ufunc_add(t3, t2, west)
free(t2)

t4 = malloc()
ufunc_add(t4, t3, south)
free(t3)

t5 = malloc()
ufunc_mul(t5, t4, 0.2)
free(t4)

update(center, t5)
free(t5)
```



```
t1 = malloc();
ufunc_add(t1, center, north)

t2 = malloc()
ufunc_add(t2, t1, east)

ufunc_add(t1, t2, west)

ufunc_add(t2, t1, south)

ufunc_mul(t1, t2, 0.2)

update(center, t1)
free(t1)
free(t2)
```

Doubling the Performance of Python/NumPy With Less Than 100 SLOC
Simon A.F. Lund, Kenneth Skovhede, Mads R.B. Kristensen, Brian Vinter. In proceedings of the 3rd
Python for High Performance and Scientific Computing (PyHPC13@SC13)

Software Victim cache

- Delay de-allocation
- Reuse buffers

```
t1 = malloc();
ufunc_add(t1, center, north)

t2 = malloc()
ufunc_add(t2, t1, east)
free(t1)

t3 = malloc()
ufunc_add(t3, t2, west)
free(t2)

t4 = malloc()
ufunc_add(t4, t3, south)
free(t3)

t5 = malloc()
ufunc_mul(t5, t4, 0.2)
free(t4)

update(center, t5)
free(t5)
```



```
t1 = malloc();
ufunc_add(t1, center, north)

t2 = malloc()
ufunc_add(t2, t1, east)

ufunc_add(t1, t2, west)

ufunc_add(t2, t1, south)

ufunc_mul(t1, t2, 0.2)

update(center, t1)
free(t1)
free(t2)
```

Doubling the Performance of Python/NumPy With Less Than 100 SLOC

Simon A.F. Lund, Kenneth Skovhede, Mads R.B. Kristensen, Brian Vinter. In proceedings of the 3rd Python for High Performance and Scientific Computing (PyHPC13@SC13)

Thread Management

Implemented with HWLOC

Multi-core and MIC

- # threads
- Control core/thread affinity

CAPE: C-Targeting Array Processing Engine

Machine:

Processor:	AMD Opteron 6272
Clock:	2.1 GHz
L3 Cache:	16MB
Memory:	128GB DDR3
Network:	Gigabit Ethernet
Compiler:	GCC 4.8.4
Software:	Ubuntu 14.04, Linux 3.13, Python 2.7.6, NumPy 1.8.2

Baseline: Python/NumPy

python [-m bohrium] benchmark.py

CAPE-AC: Without array contraction

CAPE: WITH array contraction

	CAPE-AC		CAPE			CAPE-AC		CAPE	
	1	32	1	32		1	32	1	32
Threads					Threads				
Synthetic Inplace Update	1.4	8.6	14.1	236.9	Black Scholes	3.4	27.4	6.0	84.9
Synthetic Stream Ones	1.5	9.7	10.0	137.3	Game of Life v1	1.2	8.6	1.5	32.2
Synthetic Stream Range	0.9	8.0	1.8	45.5	Game of Life v2	1.2	8.0	2.2	39.6
Synthetic Stream Random	1.0	18.0	1.0	29.8	Heat Equation	1.3	9.1	4.4	41.0
1D Stencil	0.7	8.5	1.1	21.8	Lattice Boltzmann 3D	0.8	4.0	0.8	5.3
2D Stencil	0.6	8.8	3.6	53.9	NBody	4.7	17.2	5.1	23.6
3D Stencil	0.8	11.0	1.8	48.6	NBody Nice	1.2	7.3	0.4	8.1
27 Point Stencil	1.0	5.5	1.2	13.6	SOR	1.4	8.3	1.9	20.1
Jacobi	1.4	10.3	3.9	76.2	Shallow Water	1.5	9.2	6.9	62.2
Gauss Elimination	0.7	1.6	2.1	3.8	Water-Ice Simulation	0.8	5.7	1.6	10.9
LU Factorization	0.7	1.6	1.9	3.4	kNN Naive 1	0.7	10.2	1.0	26.1
Leibnitz PI	1.0	6.1	2.2	48.3					
Monte Carlo PI	1.0	17.7	1.0	30.1					
Matrix Multiplication	0.9	10.1	2.1	47.3					
Rosenbrock	1.5	9.9	12.5	169.4					

Baseline: Serial C99

	C++ / OpenMP		NumPy / CAPE		C++ / CAPE	
	1	32	1	32	1	32
Threads						
Black Scholes +O	0.9	29.1	4.8	67.3	4.9	118.9
Black Scholes -O	0.9	29.1	1.1	26.1	1.1	31.7
Heat Equation	0.6	7.1	0.7	7.0	0.8	7.6
Leibnitz PI	1.0	22.6	0.6	14.6	0.6	15.2
Monte Carlo PI	1.0	29.8	1.0	27.8	0.9	28.2
Mxmul	1.0	9.5	1.0	14.9	1.1	15.1
Rosenbrock	1.0	21.0	1.2	15.8	1.2	21.4
Shallow Water	0.5	9.1	0.7	6.6	0.7	10.9

<https://github.com/bh107/benchpress.git> rev. 0aa2942

<https://github.com/bh107/bohrium.git> rev. b4d3586

www.erda.dk/public/archives/YXJjaGl2ZS0xSWWhQSmU=/published-archive.html

Baseline: Python/NumPy

```
python [-m bohrium] benchmark.py
```

CAPE-AC: Without array contraction

CAPE: WITH array contraction

	CAPE-AC		CAPE	
Threads	1	32	1	32
Synthetic Inplace Update	1.4	8.6	14.1	236.9
Synthetic Stream Ones	1.5	9.7	10.0	137.3
Synthetic Stream Range	0.9	8.0	1.8	45.5
Synthetic Stream Random	1.0	18.0	1.0	29.8
1D Stencil	0.7	8.5	1.1	21.8
2D Stencil	0.6	8.8	3.6	53.9
3D Stencil	0.8	11.0	1.8	48.6
27 Point Stencil	1.0	5.5	1.2	13.6
Jacobi	1.4	10.3	3.9	76.2
Gauss Elimination	0.7	1.6	2.1	3.8
LU Factorization	0.7	1.6	1.9	3.4
Leibnitz PI	1.0	6.1	2.2	48.3
Monte Carlo PI	1.0	17.7	1.0	30.1
Matrix Multiplication	0.9	10.1	2.1	47.3
Rosenbrock	1.5	9.9	12.5	169.4

	CAPE-AC		CAPE	
Threads	1	32	1	32
Black Scholes	3.4	27.4	6.0	84.9
Game of Life v1	1.2	8.6	1.5	32.2
Game of Life v2	1.2	8.0	2.2	39.6
Heat Equation	1.3	9.1	4.4	41.0
Lattice Boltzmann 3D	0.8	4.0	0.8	5.3
NBody	4.7	17.2	5.1	23.6
NBody Nice	1.2	7.3	0.4	8.1
SOR	1.4	8.3	1.9	20.1
Shallow Water	1.5	9.2	6.9	62.2
Water-Ice Simulation	0.8	5.7	1.6	10.9
kNN Naive 1	0.7	10.2	1.0	26.1

Baseline: Serial C99

	C++ / OpenMP		NumPy / CAPE		C++ / CAPE	
Threads	1	32	1	32	1	32
Black Scholes +O	0.9	29.1	4.8	67.3	4.9	118.9
Black Scholes -O	0.9	29.1	1.1	26.1	1.1	31.7
Heat Equation	0.6	7.1	0.7	7.0	0.8	7.6
Leibnitz PI	1.0	22.6	0.6	14.6	0.6	15.2
Monte Carlo PI	1.0	29.8	1.0	27.8	0.9	28.2
Mxmul	1.0	9.5	1.0	14.9	1.1	15.1
Rosenbrock	1.0	21.0	1.2	15.8	1.2	21.4
Shallow Water	0.5	9.1	0.7	6.6	0.7	10.9

Conclusion

Question: *Is it possible to construct a language agnostic backend for high-level languages without sacrificing performance?*



User knows his high-level array-oriented programming



Knows the configuration of the computing system



Combined: the high-level array-oriented programming model and its declarative nature provides implicit data-parallel operations and freedom for the backend to decide how to efficiently compute them.

Future / Ongoing Work

niels

numerically intensive expression language for science

Command-line interface via docopt

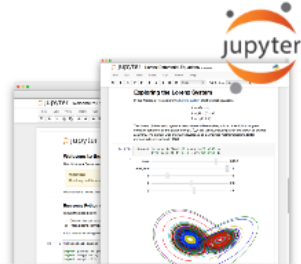
```
PI approximation using the Leibnitz formula.
-d --domain Number of samples
-t --timesteps Number of times to run

Options:
  h --help Show this screen.

Usage:
  niels <args.domain>
  niels <args.domain> <args.timesteps>

res ← 0.0
while(timestep=0) {
  timestep ← timestep + 1
  pi ← 4 * +reduce 1.0/(4.0*smp1+1.0) - 1.0/(4.0*smp1+3.0)
}
res
```

Interactive environment via



 **Bohrium** for efficient hardware utilization
1101011

 **Bohrium**
1101011

- Collaborative effort
- There is even more to it

Bohrium Processing Unit

Goal: ASIC for executing Bohrium Bytecode

- High flops-to-watt ratio
- Low latency
- FPGA prototype

niels

numerically intensive expression language for science

Command-line interface via docopt

```
/*
  PI-approximation using the Leibnitz formular.
  -d --domain      Number of samples
  -t --timesteps  Number of times to run

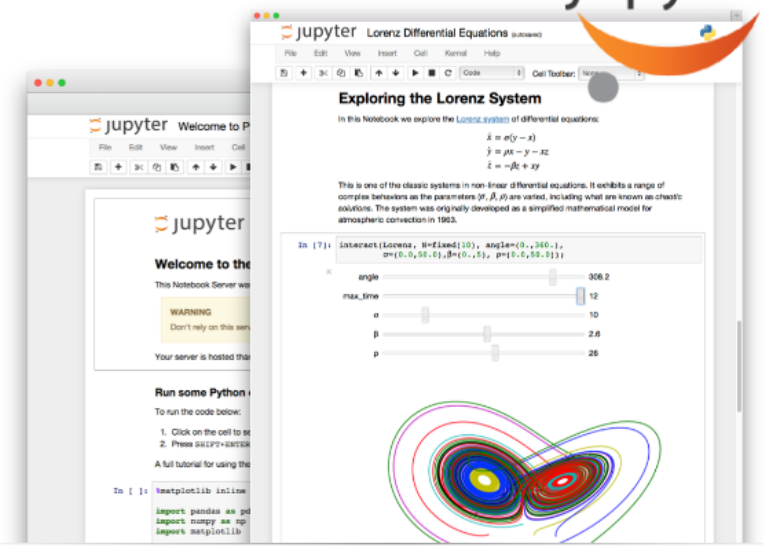
Options:
  -h --help      Show this screen.
*/

n      <- args.domain
smpls <- [0..n]
tstep <- args.timesteps

res <- 0.0
while(tstep>0) {
  tstep <- tstep-1
  pi    <- 4 * +reduce 1.0/(4.0*smp1+1.0) - 1.0/(4.0*smp1+3.0)
}

? res
```

Interactive environment via



for efficient hardware utilization

Bohrium Processing Unit

Goal: ASIC for executing Bohrium Bytecode

- High flops-to-watt ratio
- Low latency
- FPGA prototype

Future / Ongoing Work

niels

numerically intensive expression language for science

Command-line interface via docopt

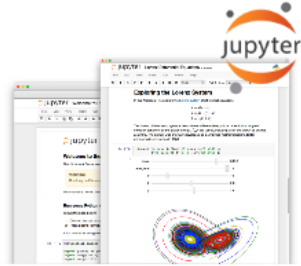
```
PI approximation using the Leibnitz formula.
-d --domain Number of samples
-t --timesteps Number of times to run

Options:
  h --help Show this screen.

Usage:
  niels <args.domain>
  niels <args.domain> <args.timesteps>

res <-> 0.0
while(timestep=0) {
  timestep = timestep + 1
  pi = 4 * +reduce 1.0/(4.0*smp1+1.0) - 1.0/(4.0*smp1+3.0)
}
res
```

Interactive environment via



 **Bohrium** for efficient hardware utilization
1101011

 **Bohrium**
1101011

- Collaborative effort
- There is even more to it

Bohrium Processing Unit

Goal: ASIC for executing Bohrium Bytecode

- High flops-to-watt ratio
- Low latency
- FPGA prototype

Goal

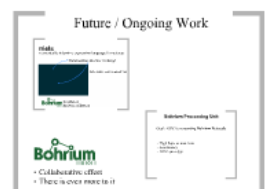
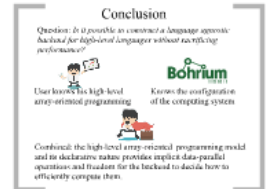
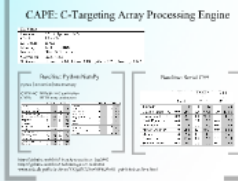
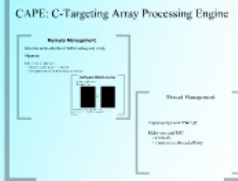
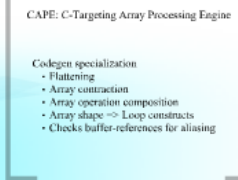
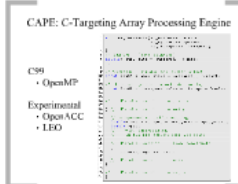
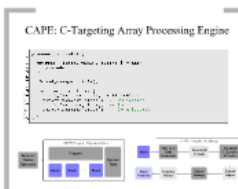
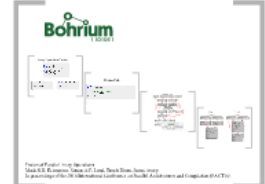
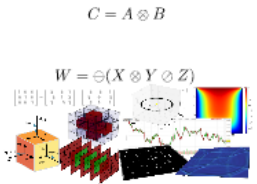
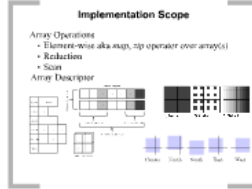
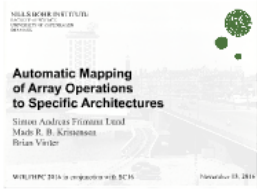
Previous

Current

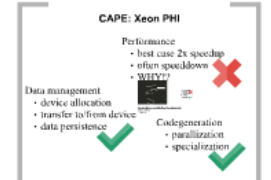
The End

What

Why



Encore



Thanks

CAPE: Xeon PHI

Performance

- best case 2x speedup
- often speeddown
- WHY!?



Allocation
~250MB/s 
Transfer
~5.5GB/s 

<https://github.com/safl/offload/tree/master/mic>
make broken
./broken

Data management

- device allocation
- transfer to/from device
- data persistence



Codegeneration

- parallization
- specialization



WHY!?

```
int main (int argc, char **argv)
{
    int device = 0; // The mic device id
    int iter = 3;
    int nelem = 5000000; // Number of elements in the arrays
    int offload = argc > 1 ? atoi(argv[1]) : 1; // Control offloading

    void* handle;
    char* error;

    printf("Initializing device(%d)... offload(%d)?\n", device, offload);
    mic_get_max_threads(device, offload); // Initialize device
    printf("Allocating on host...\n");
    // Allocate memory on host
    double* a = (double*)mmap(0, nelem*sizeof(double), PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
    double* b = (double*)mmap(0, nelem*sizeof(double), PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
    double* c = (double*)mmap(0, nelem*sizeof(double), PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
    printf("Initializing on host...\n");
    for(int i=0; i<nelem; ++i) { // Initialize data on host
        a[i] = 1;
        b[i] = 1;
        //c[i] = 1;
    }
    printf("Allocating on device...\n");
    mic_alloc(a, nelem, device, offload); // Allocate data on device
    mic_alloc(b, nelem, device, offload);
    mic_alloc(c, nelem, device, offload);
    printf("Copying to device...\n");
    mic_push(a, nelem, device, offload); // Copy input-data to device
    mic_push(b, nelem, device, offload);
    printf("Doing some work on device...\n");
    for (int i=0; i<iter; ++i) { // Call the
        voodoo(a, b, c, nelem, device, offload);
    }
    printf("Retrieving data from and deallocating data on device...\n");
    mic_free(a, nelem, device, offload); // Deallocate on device
    mic_free(b, nelem, device, offload); // Deallocate on device
    mic_pull_free(c, nelem, device, offload); // Copy result back and free on device
    printf("Bla bla...\n");
    printf("c[nelem/2] = %f\n", c[nelem/2]);
    printf("Deallocating on host...\n");
    munmap(a, nelem*sizeof(double)); // Deallocate memory on host
    munmap(b, nelem*sizeof(double));
    munmap(c, nelem*sizeof(double));

    printf("Bye!\n");
    return 0;
}
```

[Var]	mthreads	OUT
[State]	Target->host	copyout data 4
[CPU Time]	0.498360	(seconds)
[CPU->MIC Data]	0	(bytes)
[MIC Time]	0.032039	(seconds)
[MIC->CPU Data]	4	(bytes)

[CPU Time]	1.283028	(seconds)
[CPU->MIC Data]	0	(bytes)
[MIC Time]	0.000108	(seconds)
[MIC->CPU Data]	0	(bytes)

[Var]	data 1064 vs15	NOCOPY
[State]	Target->host	copyout data 0
[CPU Time]	1.273670	(seconds)
[CPU->MIC Data]	0	(bytes)
[MIC Time]	0.000227	(seconds)
[MIC->CPU Data]	0	(bytes)

[CPU Time]	1.385024	(seconds)
[CPU->MIC Data]	0	(bytes)
[MIC Time]	0.000108	(seconds)
[MIC->CPU Data]	0	(bytes)

[CPU Time]	0.072744	(seconds)
[CPU->MIC Data]	400000000	(bytes)
[MIC Time]	0.000000	(seconds)
[MIC->CPU Data]	0	(bytes)

[CPU Time]	0.071028	(seconds)
[CPU->MIC Data]	400000000	(bytes)
[MIC Time]	0.000000	(seconds)
[MIC->CPU Data]	0	(bytes)

[CPU Time]	1.823075	(seconds)
[CPU->MIC Data]	48	(bytes)
[MIC Time]	1.814970	(seconds)
[MIC->CPU Data]	8	(bytes)

[CPU Time]	0.000791	(seconds)
[CPU->MIC Data]	8	(bytes)
[MIC Time]	0.000180	(seconds)
[MIC->CPU Data]	0	(bytes)

[CPU Time]	0.000791	(seconds)
[CPU->MIC Data]	8	(bytes)
[MIC Time]	0.000180	(seconds)
[MIC->CPU Data]	0	(bytes)

[CPU Time]	0.064531	(seconds)
[CPU->MIC Data]	8	(bytes)
[MIC Time]	0.000183	(seconds)
[MIC->CPU Data]	400000000	(bytes)

WHY!?

```
int main (int argc, char **argv)
{
    int device = 0; // The mic device id
    int iter = 3;
    int nelelem = 50000000; // Number of elements in the arrays
    int offload = argc > 1 ? atoi(argv[1]) : 1; // Control offloading

    void* handle;
    char* error;

    printf("Initializing device(%d)... offload(%d)?\n", device, offload);
    mic_get_max_threads(device, offload); // Initialize device

    printf("Allocating on host...\n");
    // Allocate memory on host
    double* a = (double*)mmap(0, nelelem*sizeof(double), PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
    double* b = (double*)mmap(0, nelelem*sizeof(double), PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
    double* c = (double*)mmap(0, nelelem*sizeof(double), PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);

    printf("Initializing on host...\n");
    for(int i=0; i<nelelem; ++i) { // Initialize data on host
        a[i] = i;
        b[i] = i;
        c[i] = i;
    }

    printf("Allocating on device...\n");
    mic_alloc(a, nelelem, device, offload); // Allocate data on device
    mic_alloc(b, nelelem, device, offload);
    mic_alloc(c, nelelem, device, offload);

    printf("Copying to device...\n");
    mic_push(a, nelelem, device, offload); // Copy input-data to device
    mic_push(b, nelelem, device, offload);

    printf("Doing some work on device...\n");
    for (int i=0; i<iter; ++i) { // Call the
        voodoo(a, b, c, nelelem, device, offload);
    }

    printf("Retrieving data from and deallocating data on device...\n");
    mic_free(a, nelelem, device, offload); // Deallocate on device
    mic_free(b, nelelem, device, offload);
    mic_pull_free(c, nelelem, device, offload); // Copy result back and free on device

    printf("Bla bla...\n");
    printf("c[nelelem/2] = %f\n", c[nelelem/2]);

    printf("Deallocating on host...\n");
    munmap(a, nelelem*sizeof(double)); // Deallocate memory on host
    munmap(b, nelelem*sizeof(double));
    munmap(c, nelelem*sizeof(double));

    printf("Bye!\n");
    return 0;
}
```

[Var]	mthreads	OUT
[State]	Target->host copyout data	4
[CPU Time]		0.498360(seconds)
[CPU->MIC Data]		0 (bytes)
[MIC Time]		0.032039(seconds)
[MIC->CPU Data]		4 (bytes)

[CPU Time]	1.283028(seconds)
[CPU->MIC Data]	0 (bytes)
[MIC Time]	0.000108(seconds)
[MIC->CPU Data]	0 (bytes)

[State]	data from device copyout	8
[CPU Time]	1.273079(seconds)	
[CPU->MIC Data]	0 (bytes)	
[MIC Time]	0.300927(seconds)	
[MIC->CPU Data]	0 (bytes)	
[CPU Time]	1.385024(seconds)	
[CPU->MIC Data]	0 (bytes)	
[MIC Time]	0.000188(seconds)	
[MIC->CPU Data]	0 (bytes)	

[CPU Time]	0.072744(seconds)
[CPU->MIC Data]	40000000 (bytes)
[MIC Time]	0.000000(seconds)
[MIC->CPU Data]	0 (bytes)
[CPU Time]	0.071028(seconds)
[CPU->MIC Data]	40000000 (bytes)
[MIC Time]	0.000000(seconds)
[MIC->CPU Data]	0 (bytes)

[CPU Time]	1.823875(seconds)
[CPU->MIC Data]	48 (bytes)
[MIC Time]	1.814576(seconds)
[MIC->CPU Data]	8 (bytes)

[CPU Time]	0.000791(seconds)
[CPU->MIC Data]	8 (bytes)
[MIC Time]	0.000160(seconds)
[MIC->CPU Data]	0 (bytes)
[CPU Time]	0.000791(seconds)
[CPU->MIC Data]	8 (bytes)
[MIC Time]	0.000160(seconds)
[MIC->CPU Data]	0 (bytes)
[CPU Time]	0.064531(seconds)
[CPU->MIC Data]	8 (bytes)
[MIC Time]	0.000183(seconds)
[MIC->CPU Data]	40000000 (bytes)

Allocation
~250MB/s
Transfer
~5.5GB/s

<https://github.com/safl/offload/tree/master/mic>
make broken
./broken

CAPE: Xeon PHI

Performance

- best case 2x speedup
- often speeddown
- WHY!?



Allocation
~250MB/s 
Transfer
~5.5GB/s 

<https://github.com/safl/offload/tree/master/mic>
make broken
./broken

Data management

- device allocation
- transfer to/from device
- data persistence



Codegeneration

- parallization
- specialization



SIMD utilization

CAPE codegen takes SIMD into consideration, however, current implementation relies on auto-vectorization by the backend C-compiler

Brittle, example:

gcc often fails where commercial compilers prevail.

Simplistic approach by using `#pragma omp simd [...]` did not yield expected results

Investigate further and possibly expand codegen with intrinsics or explicit means of ensuring the compiler that vectorization makes sense.

Goal

Previous

Current

The End

What

Why

Automatic Mapping of Array Operations to Specific Architectures
Simon Andrew Fritzsche-Lund, Math E. B. Kressner, Brian Vinter
VLSI/ASIC 2015 in conjunction with SC15, November 13, 2015

Tools of the trade
Productivity
Performance
MATLAB, Python, C++
Progressive Language

Implementation Scope
Array Operations
• Element-wise aka. map, slip operator over arrays()
• Reduction
• Scan
Array Descriptor
Order, Rank, Strides, Step, Mask

$C = A \otimes B$
 $W = \ominus(X \otimes Y \otimes Z)$

HIPERFIT
• Improves mathematical models for Fortran
• Express them in vendor-specific Languages (DSLs)
• Executes them efficiently on High-Performance Systems

Backend Design Criteria
Language agnostic
• Repetitive programming model for a specific language
• Productivity Model
• High-level
• Declarative
• Array-oriented
Language intrinsic for internal data representation
Efficient
• Target performance: comparable to single-compiled hand-written C/C++ for its own application
Question: Is it possible to describe a language agnostic Backend for high-level language without sacrificing performance?

Bohrium
Language Integration
• Map abstractions
• Vector/Strides -> internal data representation
Internal Representation DSL
• Abstracted vector/Strides
Transformations
• Optimization
• Normalization
• Fusion, preserving bytecode sequences

Bohrium
Reconfigurable Backend [capec, fuser, peasy, gnu]

Bohrium
Backend Architecture

Bohrium
CAPE: C-Targeting Array Processing Engine
• Code generator for array operations with parallelization and optimization of multiple array operations
• Calling JIT-Compiler and other targets for array operation kernels
• Backend homogenizing capabilities, better management, array operation scheduling and creation

CAPE: C-Targeting Array Processing Engine

CAPE: C-Targeting Array Processing Engine
CSI
• OpenMP
Experimental
• OpenACC
• LED

CAPE: C-Targeting Array Processing Engine
Codegen specialization
• Flattening
• Array contraction
• Array operation composition
• Array shape => Loop constructs
• Checks buffer-references for aliasing

CAPE: C-Targeting Array Processing Engine
Memory Management
• Manual Management
• Automatic Management

CAPE: C-Targeting Array Processing Engine
Performance Analysis

Conclusion
Question: Is it possible to construct a language agnostic Backend for high-level language without sacrificing performance?
User knows the high-level array-oriented programming
Knows the configuration of the computing system
Confirms: the high-level array-oriented programming model and its declarative nature provides implicit data-parallel operations and fusions for the backend to decide how to efficiently compute them

Future / Ongoing Work
Bohrium Processing Unit

Encore

CAPE: Xeon PHI
Performance
• best case 2x speedup
• often speedupdown
• WIPV? ~~X~~
Data management
• device allocation
• transfer to/from device
• data persistence
Codegeneration
• parallelization
• specialization ~~X~~

SIMD utilization
CAPE codegen takes SIMD into consideration, however, current implementation relies on auto-vectorization by the backend/Compiler
Bench: example
gap after 90% where commercial compilers prevail
Synthetic approach by using OpenMP loop unroll -1 did not yield expected results
Investigate further and possibly expand codegen with manual or explicit means of creating the complex data reconstruction and/or code

Thanks