Thanks
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
$C = A \otimes B$
\[ C = A \otimes B \]

\[ W = \ominus (X \otimes Y \otimes Z) \]
• 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

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
OpenMP / pthreads / Qthread
OpenACC / LEO
OpenCL / CUDA

C / C++ / Fortran
MPI
PGAs

CPUs,
APUs, Hybrid, FPGA,
GPUs, Accelerators,

and clusters of them configured in shared and distributed memory systems...
Heat Equation in Python / NumPy

```python
def solve(grid, epsilon):
    center = grid[1:-1, 1:-1]
    north = grid[0:-2, 1:-1]
    south = grid[2:, 1:-1]
    east = grid[1:-1, 2:]
    west = grid[1:-1, 0:-2]

    delta = epsilon + 1
    while delta > epsilon:
        work = (center + north + east + west + south) * 0.2
        delta = np.sum(np.absolute(work - center))
        center[:] = work
```
```c
#include <math.h>
solve(int size, double *grid, double epsilon)
{
    int gsize = size+2; //Size + borders.
    double *T = malloc(gsize*gsize*sizeof(double));
    double delta = epsilon+1;
    while(delta > epsilon)
    {
        double *a = grid;
        double *t = T;
        delta = 0;
        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 = a+gsize+1;
            double *t_center = t+gsize+1;
            for(j=0; j<size; ++j)
            {
                *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
                delta += fabs(t_center+center);
            }
            a += gsize;
            t += gsize;
        }
        memcpy(A, T, gsize*gsize*sizeof(double));
    }
```
```c
#include <math.h>
solve(int size, double *grid, double epsilon) {
  int gsize = size+2; // Size + borders.
  double *T = malloc(gsize*gsize*sizeof(double));
  double delta = epsilon+1;
  while(delta > epsilon) {
    delta = 0;
    #pragma omp parallel for shared(grid,T) reduction(+:delta)
    for(i=0; i<size; ++i) {
      int a = i * gsize;
      double *up   = &grid[a+1];
      double *left = &grid[a+gsize];
      double *right = &grid[a+gsize+2];
      double *down  = &grid[a+1+gsize*2];
      double *center = &grid[a+gsize+1];
      double *t_center = &T[a+gsize+1];
      for(j=0; j<size; ++j) {
        *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
        delta += fabs(t_center+center);
      }
    }
    memcpy(grid, T, gsize*gsize*sizeof(double));
  }
}
# Heat Equation in C and OpenMP

```c
#include <math.h>
#include <mpi.h>
solve(int size, double *grid, double epsilon)
{
  int gsize = size + 2; //Size + borders.
  double *T = malloc(gsize*gsize*sizeof(double));
  double delta = epsilon + 1;
  while(delta > epsilon)
  {
    delta = 0;
    #pragma omp parallel for shared(grid,T) reduction(+:delta)
    for(i=1; i<size; ++i)
    {
      int a = i * gsize;
      double *up = &grid[a+1];
      double *left = &grid[a+gsize];
      double *right = &grid[a+gsize+2];
      double *down = &grid[a+gsize+1];
      double *t_center = &T[a+gsize];
      for(j=0; j<size; ++j)
      {
        *t_center = (*center + *up++ + *left++ + *right++ + *down++) * 0.2;
        delta += fabs(t_center-center);
      }
    }
    memcp(grid, T, gsize*gsize*sizeof(double));
  }
  MPI_Allreduce(delta, delta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
}
```

Heat Equation in C and OpenMP and MPI
Heat Equation in C and OpenMP and MPI with Latency Hiding

```c
#include <stdio.h>
#include <mpi.h>

solve(int size, double *grid, double epsilon) {
    int gsize = SIZE; // size = borders;
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &worldsize);
    MPI_Comm comm;
    int period[2] = {0};
    MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize, 1, &period[0], &comm);
    int l = size / worldsize;
    if (myrank == worldsize - 1) l = size / worldsize + 1;
    int h = 1 / (epsilon * size);
    double delta = epsilon * h;

    while(delta > epsilon) {
        int p, src, dest;
        MPI_Request req[3];
        // Initialize send/receive - neighbor above
        MPI_Cart_shift(comm, 1, 0, src, 0, dest);
        MPI_Isend(grid[src], gsize, MPI_DOUBLE, p, dest, &comm, &req[0]);
        MPI_Irecv(grid[dest], gsize, MPI_DOUBLE, p, src, &comm, &req[1]);
        // Initialize send/receive - neighbor below
        MPI_Cart_shift(comm, 1, 0, src, 0, dest);
        MPI_Isend(grid[src], gsize, MPI_DOUBLE, p, dest, &comm, &req[2]);
        MPI_Irecv(grid[dest], gsize, MPI_DOUBLE, p, src, &comm, &req[3]);

        // Handle the non-border elements.
        delta *= 0.8;
        //pragma omp parallel for shared(grid,T) reduction(+delta)
        for(int i = 1; i < size; i++)
            for(int j = 1; j < size; j++)
                { t_center = (*center + *up + *left + *right + *down) / 5;
                  delta += fabs(T_center - t_center);
                }

        MPI_Waitall(3, req, MPI_STATUSES_IGNORE);
        MPI_Allreduce(&delta, delta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    }
}
```
Heat Equation in C and OpenMP and MPI with Latency Hiding

```c
#include <math.h>
#include <mpi.h>

solve(int size, double *grid, double epsilon) {
    int i, j, k, l, M, N, P, Q, R, S;
    double dt = 1e-10; // delta time
    double dx = 1.0 / size; // delta x
    double dx2 = dx * dx;
    double epsilon2 = epsilon * epsilon;

    // Initial conditions
    for (i = 0; i < size; i++)
        grid[i] = std::sin(2 * M_PI * i / size);

    // Main loop
    for (int t = 0; t < T; t++)
        for (i = 1; i < size - 1; i++)
            for (j = 1; j < size - 1; j++)
                grid[i][j] = grid[i][j] + dt * (dx2 * (grid[i+1][j] - 2 * grid[i][j] + grid[i-1][j])
                                                  + dx2 * (grid[i][j+1] - 2 * grid[i][j] + grid[i][j-1])
                                                  + epsilon2 * (grid[i][j+1] + grid[i][j-1] - 2 * grid[i][j])
                                                  + epsilon2 * (grid[i+1][j] + grid[i-1][j] - 2 * grid[i][j])) / dx2;

    // Output
    for (i = 0; i < size; i++)
        for (j = 0; j < size; j++)
            printf("%.6f\t", grid[i][j]);

    MPI_Finalize;
}
```

---

Heat Equation in Python / NumPy

```python
import numpy as np

# Initialize grid
grid = np.zeros((size, size), dtype=float)

# Set initial conditions
for i in range(size):
    for j in range(size):
        grid[i][j] = np.sin(2 * np.pi * i / size)

# Main loop
for t in range(T):
    for i in range(1, size-1):
        for j in range(1, size-1):
            grid[i][j] = grid[i][j] + dt * (dx2 * (grid[i+1][j] - 2 * grid[i][j] + grid[i-1][j])
                                              + dx2 * (grid[i][j+1] - 2 * grid[i][j] + grid[i][j-1])
                                              + epsilon2 * (grid[i][j+1] + grid[i][j-1] - 2 * grid[i][j])
                                              + epsilon2 * (grid[i+1][j] + grid[i-1][j] - 2 * grid[i][j])) / dx2

# Output
for i in range(size):
    for j in range(size):
        print(grid[i][j], end='   ')

# Finalize
```
def solve(grid, epsilon):
    center = grid[1:-1, 1:-1]
    north = grid[0:-2, 1:-1]
    south = grid[2:, 1:-1]
    east = grid[1:-1, 2:]
    west = grid[1:-1, 0:-2]

    delta = epsilon + 1
    while delta > epsilon:
        work = (center + north + east + west + south) * 0.2
        delta = np.sum(np.abs(work - center))
        center[:] = work
Programming Pitfalls

Correctness
- Deadlocks
- Race-conditions
• 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 straightforward 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

<table>
<thead>
<tr>
<th>type</th>
<th>float64</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndim</td>
<td>3</td>
</tr>
<tr>
<td>start</td>
<td>0</td>
</tr>
<tr>
<td>shape</td>
<td>2 2 2</td>
</tr>
<tr>
<td>stride</td>
<td>7 3 1</td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
</tbody>
</table>

Data layout

Outer dimension

Inner dimension

Middle dimension

Skip by stride

1) Dense
2) Strided
3) Strided

Center  North  South  East  West
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]
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

```c
#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];
```

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

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

COPY A, 0
COPY D, 0
COPY E, 0
COPY B, 0

ADD A, A, D[:-1]
ADD B, B, E[:-1]

COPY A, D[:-1]
COPY B, E[:-1]

MUL T, A, B

MAX D[1:], T, E[1:]
DEL A
DEL B

MIN E[1:], T, D[1:]
SYNC D
DEL T
DEL E

Array Bytecode
Label Cost
Weight Edge 5
Fuse-preventing edge
Dependency Edge
**Greedy**

*Total Cost: 46*

- COPY D, 0
- COPY E, 0
- COPY A, 0
- ADD A, A, D[−1]
- COPY A, D[−1]
- MUL T, A, B
- DEL A
- DEL B
- MAX D[1:], T, E[1:]
- MIN E[1:], T, D[1:]
- DEL T
- DEL E
- SYNC D
- DEL D

**Optimal**

*Total Cost: 34*

- COPY D, 0
- COPY E, 0
- COPY A, 0
- ADD A, A, D[−1]
- COPY A, D[−1]
- COPY B, 0
- ADD B, B, E[−1]
- COPY B, E[−1]
- MUL T, A, B
- DEL A
- DEL B
- MAX D[1:], T, E[1:]
- MIN E[1:], T, D[1:]
- DEL T
- DEL E
- SYNC D
- DEL D
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

```java
process(bytecode) {
    program, symbol_table, blocks[] = map(
        bytecode
    );
    thread_manager.bind();
    for block in blocks {
        block_function = jit_compiler(block);
        memory_manager(block);  // Allocation
        block_function(block);   // Execution
        memory_manager(block);   // De-allocation
    }
}
```
CAPE: C-Targeting Array Processing Engine

C99
- OpenMP

Experimental
- OpenACC
- LEO

```c
void KP_IDENTHASH(kp_buffer** buffers,
                  kp_operand** operands,
                  kp_iterspace* iterspace)
{
  // Buffers (data pointers)
  double* buf0_data = buffers[0]->data;
  ...

  // Operands (strides and buffer offset)
  const int64_t opd0_start = operands[0]->start;

  // Iterspace (shape, layout, #elements)
  const int64_t iterspace_nelem = iterspace->nelem;
  ...

  // Parallel section — prequel

  // Parallel section — entry
  // — operand to buffer mapping
  double* restrict opd0 = buf0_data + opd0_start;
  double opd1;
  // — work distribution
  // — initialize accumulator variables

  // Parallel section — loop constructs
  {
    ...array operations...
  }

  // Parallel section — exit

  // Parallel section — sequel
}
```
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

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

Thread Management

Implemented with HWLOC

Multi-core and MIC
  • # threads
  • Control core/thread affinity
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

Double the Performance of Python/NumPy With Less Than 100 SLOC
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)
```

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

Baseline: Python/NumPy

CAPE: Without array contraction
CAPE: WITH array contraction

<table>
<thead>
<tr>
<th>Baseline: Serial C99</th>
<th>C++ / OpenMP</th>
<th>NumPy / CAPE</th>
<th>C++ / CAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>1</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>Black Scholes +O</td>
<td>0.9</td>
<td>29.1</td>
<td>4.8</td>
</tr>
<tr>
<td>Black Scholes -O</td>
<td>0.9</td>
<td>29.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Heat Equation</td>
<td>0.6</td>
<td>7.1</td>
<td>0.7</td>
</tr>
<tr>
<td>Leibnitz PI</td>
<td>1.0</td>
<td>22.6</td>
<td>0.6</td>
</tr>
<tr>
<td>Monte Carlo PI</td>
<td>1.0</td>
<td>29.8</td>
<td>1.0</td>
</tr>
<tr>
<td>Mxmul</td>
<td>1.0</td>
<td>9.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>1.0</td>
<td>21.0</td>
<td>1.2</td>
</tr>
<tr>
<td>Shallow Water</td>
<td>0.5</td>
<td>9.1</td>
<td>0.7</td>
</tr>
</tbody>
</table>

https://github.com/bh107/benchpress.git rev. 0aa2942
https://github.com/bh107/bohrium.git rev. b4d3586
www.erdak.dk/public/archives/YXJjaGl2ZS0xSWhQSmU=/published-archive.html
Baseline: Python/NumPy

python [-m bohrism] benchmark.py

CAPE-AC: Without array contraction
CAPE: WITH array contraction

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

<table>
<thead>
<tr>
<th></th>
<th>CAPE-AC</th>
<th>CAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>Black Scholes</td>
<td>3.4</td>
<td>27.4</td>
</tr>
<tr>
<td></td>
<td>6.0</td>
<td>84.9</td>
</tr>
<tr>
<td>Game of Life v1</td>
<td>1.2</td>
<td>8.6</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>32.2</td>
</tr>
<tr>
<td>Game of Life v2</td>
<td>1.2</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>39.6</td>
</tr>
<tr>
<td>Heat Equation</td>
<td>1.3</td>
<td>9.1</td>
</tr>
<tr>
<td></td>
<td>4.4</td>
<td>41.0</td>
</tr>
<tr>
<td>Lattice Boltzmann 3D</td>
<td>0.8</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>5.3</td>
</tr>
<tr>
<td>NBody</td>
<td>4.7</td>
<td>17.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>23.6</td>
</tr>
<tr>
<td>NBody Nice</td>
<td>1.2</td>
<td>7.3</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>8.1</td>
</tr>
<tr>
<td>SOR</td>
<td>1.4</td>
<td>8.3</td>
</tr>
<tr>
<td></td>
<td>1.9</td>
<td>20.1</td>
</tr>
<tr>
<td>Shallow Water</td>
<td>1.5</td>
<td>9.2</td>
</tr>
<tr>
<td></td>
<td>6.9</td>
<td>62.2</td>
</tr>
<tr>
<td>Water-Ice Simulation</td>
<td>0.8</td>
<td>5.7</td>
</tr>
<tr>
<td></td>
<td>1.6</td>
<td>10.9</td>
</tr>
<tr>
<td>kNN Naive 1</td>
<td>0.7</td>
<td>10.2</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>26.1</td>
</tr>
</tbody>
</table>
## Baseline: Serial C99

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>OpenMP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Threads</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td>Black Scholes +O</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>29.1</td>
</tr>
<tr>
<td>Black Scholes −O</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>29.1</td>
</tr>
<tr>
<td>Heat Equation</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>7.1</td>
</tr>
<tr>
<td>Leibnitz PI</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>22.6</td>
</tr>
<tr>
<td>Monte Carlo PI</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>29.8</td>
</tr>
<tr>
<td>Mxmul</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>9.5</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>21.0</td>
</tr>
<tr>
<td>Shallow Water</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>9.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>CAPE</td>
</tr>
<tr>
<td>Threads</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td>Black Scholes +O</td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>118.9</td>
</tr>
<tr>
<td>Black Scholes −O</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>31.7</td>
</tr>
<tr>
<td>Heat Equation</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>7.6</td>
</tr>
<tr>
<td>Leibnitz PI</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>15.2</td>
</tr>
<tr>
<td>Monte Carlo PI</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>28.2</td>
</tr>
<tr>
<td>Mxmul</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>15.1</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>21.4</td>
</tr>
<tr>
<td>Shallow Water</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>10.9</td>
</tr>
</tbody>
</table>
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 doceopt
Interactive environment via jupyter

Bohrium Processing Unit

Goal: ASIC for executing Bohrium Bytecode

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

- Collaborative effort
- There is even more to it
niels
numerically intensive expression language for science

Command-line interface via docopt

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 doceopt

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

- Collaborative effort
- There is even more to it
Thanks
CAPE: Xeon PHI

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

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

Codegeneration
- parallization
- specialization

https://github.com/safi/offload/tree/master/inic
make broken
broken
CAPE: Xeon PHI

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

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

Codegeneration
- parallelization
- 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.