The Bohrium Processing Unit
A FPGA backend for Bohrium

CPA 2013, Edinburgh
The Bohrium Processing Unit
A FPGA backend for Bohrium

Kenneth Skovhede
eScience
Niels Bohr Institute
University of Copenhagen

CPA 2013, Edinburgh
Presentation topics

Bohrium

FPGA

BPU

Simulation

Future Work
Motivation
Stencil example in Matlab

#Parameters
1 %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:l,
    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
// 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++) \n            / 5.0;
        }
        a += gsize;
        t += gsize;
    }
    memcpy(A, T, gsize*gsize*sizeof(double));
}
Stencil example with inefficient MPI

```c
// Computation
int size = SIZE; // Size of the matrix
MPI_Comm lineback(MPI_COMM_WORLD, &size); // MPI_COMM_WORLD, &size
int periods[] = {0}; // Periods
MPI Comm comm;
MPI Create(comm, &periods, &size);
for (int i = 0; i < size; i++)
{
    // Send/Receive - neighbor above
    MPI_Sendrecv(matrix[i], size, MPI_DOUBLE, 0, 1, &size, MPI_DOUBLE, MPI_Bcast, matrix[i], size, MPI_DOUBLE, MPI_Bcast, 0, MPI_COMM_WORLD, &status);
}
for (int i = 0; i < size; i++)
{
    // Send/Receive - neighbor below
    MPI_Sendrecv(matrix[i], size, MPI_DOUBLE, 0, 1, &size, MPI_DOUBLE, MPI_Bcast, matrix[i], size, MPI_DOUBLE, MPI_Bcast, 0, MPI_COMM_WORLD, &status);
}
for (int i = 0; i < size; i++)
{
    // Send/Receive - neighbor left
    MPI_Sendrecv(matrix[i], size, MPI_DOUBLE, 0, 1, &size, MPI_DOUBLE, MPI_Bcast, matrix[i], size, MPI_DOUBLE, MPI_Bcast, 0, MPI_COMM_WORLD, &status);
}
for (int i = 0; i < size; i++)
{
    // Send/Receive - neighbor right
    MPI_Sendrecv(matrix[i], size, MPI_DOUBLE, 0, 1, &size, MPI_DOUBLE, MPI_Bcast, matrix[i], size, MPI_DOUBLE, MPI_Bcast, 0, MPI_COMM_WORLD, &status);
}
}
```
/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[1] = {0};
MPI_Cart_create(MPI_COMM_WORLD, 1, &worldsize,
    periods, 1, &comm);
int L = SIZE / worldsize;
if(myrank == worldsize-1)
    L += 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
# 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++
# Monte Carlo PI

Sequential code

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

Vector bytecode

```plaintext
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.
Bridge is language bindings and interface to Bohrium, currently for NumPy

VEM has a simple interface and can support hierarchical setups. The VEM can distribute and load-balance as required.

Node level VEM knows about hardware features and schedules operations optimally on hardware.

VE's are the workhorses and know how to implement elementwise operations and composite operations, currently on CPU and GPU.
Black-Scholes
n-body
Shallow water
High frequency
low latency trading
FPGA

Behrius as a front-end
- Gain all the benefits of a high-level programming language
- Gain all the benefits of FPGA hardware
- No need to work with the low-level FPGA details

Why not use FPGAs for everything?
- Very low level programming
- Need to worry about signal propagation
- More expensive than GPGPU

Why use FPGA?
- Can work in stand-alone mode, with no host machine
- Low power
- Higher durability
- Low latency network processing

Full adder in VHDL
Why use FPGA?

- Can work in stand-alone mode, with no host machine
  - Low power
  - Higher durability
- Low latency network processing
Why not use FPGAs for everything?

- Very low level programming
- Need to worry about signal propagation
- More expensive than GPGPU
4bit full adder in VHDL

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
useIEEE.STD_LOGIC_UNSIGNED.ALL;

entity Adder4 is
port (A,B : in bit_vector(3 downto 0); ----inputs
     Cin : in bit ;
     S : out bit_vector(3 downto 0); ------Outputs
     Co : out bit);
end Adder4;

architecture Behavioral of Adder4 is
component FullAdder
port (X,Y,Cin : in bit;
     Cout,Sum : out bit);
end component;

signal C: bit_vector(3 downto 1);
begin
---Instantiate four copies of the full adder
FA0: FullAdder port map (A(0),B(0),Cin,C(1),S(0));
FA1: FullAdder port map (A(1),B(1),C(1),C(2),S(1));
FA2: FullAdder port map (A(2),B(2),C(2),C(3),S(2));
FA3: FullAdder port map (A(3),B(3),C(3),Co,S(3));
end Behavioral;
Bohrium as a front-end

- Gain all the benefits of a high-level programming language
- Gain all the benefits of FPGA hardware
- No need to work with the low-level FPGA details
Bohrium Processing Unit

Why a processing unit?

Basic design:

Trippel buffer register files
Quad kernel memories
Simulation

FPGA development
- Simulation
- Structural
- Behavioral

Required components
- ALU
  - How wide?
  - Registers
  - ALU width, but how large?
  - Tripple buffer for registers
    - 3 is nice :)
  - Kernel memories
    - How many?

Essentially an optimisation problem:
We know how many gates we have, but
not how to utilize them best.
Required components

- ALU
  - How wide?
- Registers
  - ALU width, but how large?
  - Trippel buffer registers
    - 3 is nice :)
- Kernel memories
  - How many?

Essentially an optimization problem:
We know how many gates we have, but not how to utilize them best
FPGA development

- Simulation
- Structural
- Behavioral
Rising edge

Faling edge

Time

Hardware clock

Process

Read channel

Write channel

PyCSP model
Future work

- More examples
- Prototype
- Recursive refinement
Thanks

www.bh107.org