cphVB service in MiG

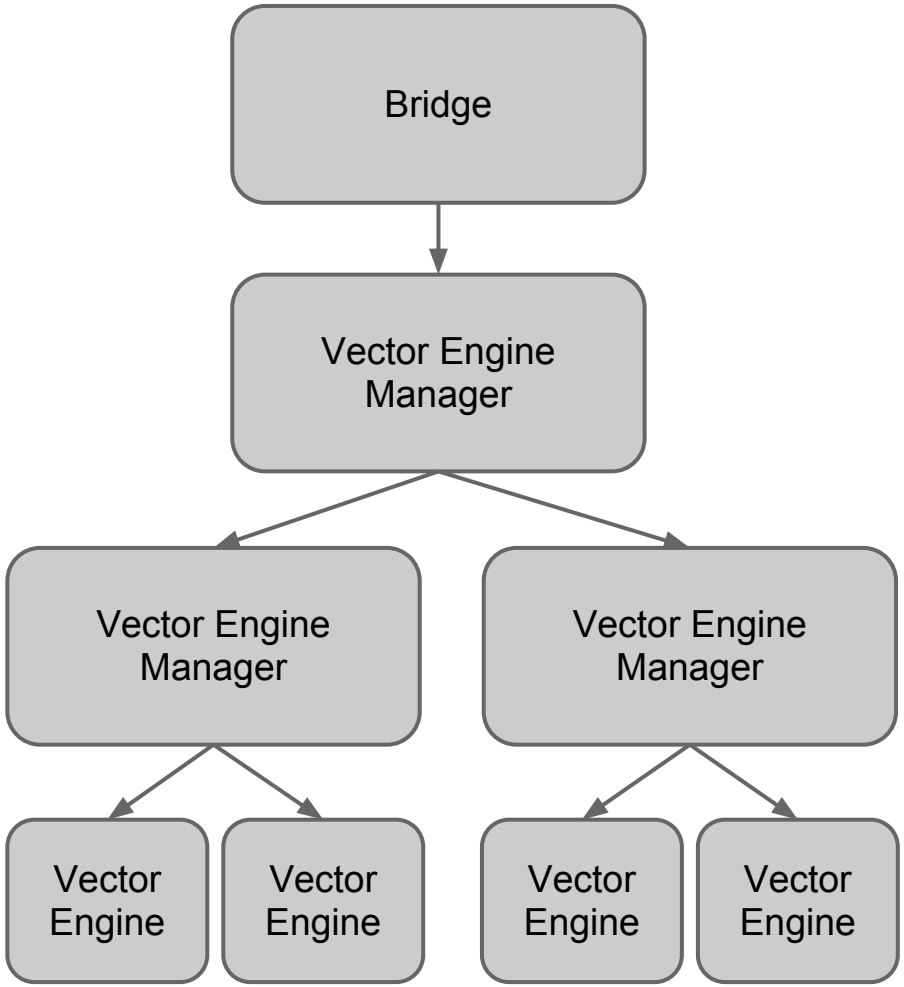
Preliminary ideas about a compute engine for cphVB inside MiG

Kenneth Skovhede
Phd Student
eScience, NBI

cphVB Quick Intro

Forget handcrafting CUDA/OpenCL to utilize your GPU, forget threading, mutexes and locks to utilize your multi-core CPU and forget about MPI and net topology to program your cluster just cphVB!

cphVB Quick Intro



Bridge is language bindings and interface to cphVB, currently for numpy, and CIL (C#, F#, IronPython, etc)

VEM has a simple interface and can support hieracical 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 on GPU or multicore.

cphVB Example - Jacobi in numpy

```
import numpy as np
import cphvbnumpy
import util

B = util.Benchmark()
H = B.size[0]
W = B.size[1]
iterations = B.size[2]

full = np.empty((H+2,W+2), dtype=np.double,
dist=B.cphvb)
work = np.empty((H,W), dtype=np.double,
dist=B.cphvb)
full[:]    = 0.0
full[:,0]  = -273.15
full[:,-1] = -273.15
full[0,:]  =  40.0
full[-1,:] = -273.13
```

CONTINUED --->

```
B.start()
for i in xrange(iterations):
  work[:] = full[1:-1, 1:-1]
  work += full[1:-1, 0:-2]
  work += full[1:-1, 2:  ]
  work += full[0:-2, 1:-1]
  work += full[2:  , 1:-1]
  work *= 0.2
  diff = np.absolute(full[1:-1, 1:-1] -
work)
  delta = np.add.reduce(diff)
  delta = np.add.reduce(delta)
  full[1:-1, 1:-1] = work
B.stop()

B.pprint()
```

cphVB Example - Jacobi in C#

```
public static void Solve(long width, long height) {
     var full = Generate.Zeroes(height + 2, width + 2);
     var work = Generate.Zeroes(height, width);
     var diff = Generate.Zeroes(height, width);

     var cells = full[R.Slice(1, -1), R.Slice(1, -1)];
     var up =    full[R.Slice(1, -1), R.Slice(0, -2)];
     var left =  full[R.Slice(0, -2), R.Slice(1, -1)];
     var right = full[R.Slice(2,  0), R.Slice(1, -1)];
     var down =  full[R.Slice(1, -1), R.Slice(2,  0)];

     full[R.All, R.El(0)] += -273.5f;
     full[R.All, R.El(-1)] += -273.5f;
     full[0] += 40f;
     full[-1] += -273.5f;

     T epsilon = width * height * 0.002f;
     T delta = epsilon + 1;
```

```
while (epsilon < delta)
{
     work[R.All] = cells;

     work += up;
     work += left;
     work += right;
     work += down;
     work *= 0.2f;*/

     Sub.Apply(cells, work,
diff);
     Abs.Apply(diff, diff);
     delta = diff.Sum();
}
}
```

cphVB Example - BlackScholes in F#

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

    let L = X.Abs()
    let K = 1.0f / (1.0f + 0.2316419f * L)
    let w = 1.0f - 1.0f / ((float32(sqrt(2.0 * System.Math.PI)))) * (L.Negate() * L / 2.0f).Exp() * (a1 * K + a2 * (K.Pow(2.0f)) + a3 * (K.Pow(3.0f)) + a4 * (K.Pow(4.0f)) + a5 * (K.Pow(5.0f)));

    let mask1 = X.Apply<LessThan>()
    let mask2 = X.Apply<GreaterThanOrEqual>()

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

cphVB - Implementation details

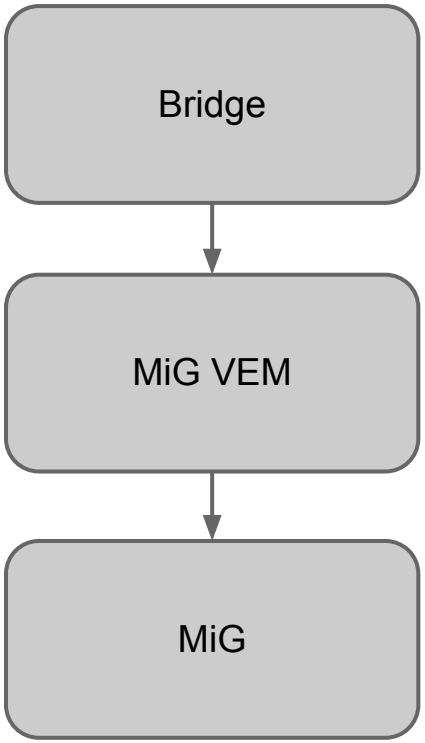
- Elementwise operations defined as vector bytecode
- Operands defined as ndarray like views
- Complex operations implemented as userdefined functions (e.g. matrix multiplication)
- Bridge collects a batch of instructions, and ask for synchronous execution
- VEM can split/forward as required
- VE can produce kernels from batches

cphVB and MiG

Overall goal:

Run any cphVB targeted program unmodified on a MiG resource

cphVB and MiG - How?



Bridge level is unaware of a special type of VEM.

VEM is aware of MiG and schedules tasks on MiG rather than executing them locally on VE's.

cphVB and MiG - Benefits

- Ability to use a large set of compute resources from seemingly sequential programs
- An extreme application of "develop and test locally, run on cluster without modification"

- Benefits cphVB because it extends the range of machines that can be utilized
- Benefits MiG because the grid service becomes more accesible to non-technical users

cphVB and MiG - Challenges

- Suitable for special problem characteristics:
  - Sufficient size kernels
  - Sufficient amount of parallel jobs
- Need extra possibilities in cphVB:
  - Ability to read and write files through cphVB
  - The MiG VEM needs to be aware of the resource physical layout and capabilitites
  - Ability to JIT compile a suitable kernel with operations
  - Endianness, floating point representation, etc

cphVB and MiG - Where are we

- Idea phase!

- Bridge is functional
- Kernel bundling reuseable from GPU VE
- MiG API is well known
- May need some heavy operations, such as FFT implemented in cphVB