Chapel On Accelerators

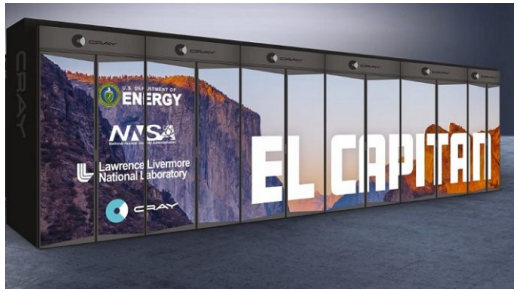
Rahul Ghangas (Australian National University)

Supervisor - Dr. Josh Milthorpe (Australian National University)

A Little About Me

- Rahul Ghangas
- Final semester student at ANU
- Working on Chapel for my honours thesis
- Have been a Chapel enthusiast for a while now

Why GPUs?



CPU - AMD Genoa Epyc
GPU - AMD Radeon

CPU - Xeon Scalable Processor
GPU - Intel Xe

CPU - AMD Milan
GPU - Nvidia Tesla

Source : hpcwire.com

The Two-Language Problem

C Host Code

```
cl_device_id device_id; // compute device id
cl_context context; // compute context
cl_command_queue commands; // compute command queue
cl_program program; // compute program
cl_kernel ko_vadd; // compute kernel

cl_mem d_a;
cl_mem d_b;
cl_mem d_c;
int i = 0;
for(i = 0; i < LENGTH; i++){
h_a[i] = rand() / (float)RAND_MAX;
h_b[i] = rand() / (float)RAND_MAX;
h_e[i] = rand() / (float)RAND_MAX;
h_g[i] = rand() / (float)RAND_MAX;
}

cl_uint numPlatforms;

err = clGetPlatformIDs(0, NULL, &numPlatforms);
checkError(err, "Finding platforms");
if (numPlatforms == 0)
{
printf("Found 0 platforms!\n");
return EXIT_FAILURE;
}

cl_platform_id Platform[numPlatforms];
err = clGetPlatformIDs(numPlatforms, Platform, NULL);
checkError(err, "Getting platforms");
```

Opencl C kernel code

```
const char *KernelSource = "\n" \
"__kernel void vadd(         \n" \
" __global float* a,         \n" \
" __global float* b,         \n" \
" __global float* c,         \n" \
" const unsigned int count) \n" \
"{                           \n" \
" int i = get_global_id(0); \n" \
" if(i < count)              \n" \
" c[i] = a[i] + b[i];        \n" \
"}                           \n";
```

Needs native support for “productivity”

- Chapel’s high level constructs offer a “productive” interface to parallel/distributed computing for any programmer
    1. Reduce Expressions
    2. Operator Promotion
    3. Forall/CoForall Loops

Reduction Operations

- Initial native GPU support, follows up on previous work[1]
- Making it work with the current state of the compiler
- Extending support for complex expressions

AMD Proposal

```
const D = {1..100};
var varA : int;
var varA : [D] int = 13;


on here.GPU do {
varA = + reduce varB;
}
```

Our Proposal

```
const D = {1..100};
var varA : int;
var varB : [D] int = 12;
var varC : [D] int = 13;

on here.GPU do {
varA = + reduce (varB + 3 * varC);
}
```

[1] Michael L. Chu, Ashwin M .Aji, Daniel Lowell, and Khaled Hamidouche. *GPGPU support in Chapel with the Radeon Open Compute Platform.* CHIUW 2017.

Inferred intents for OpenCL kernels

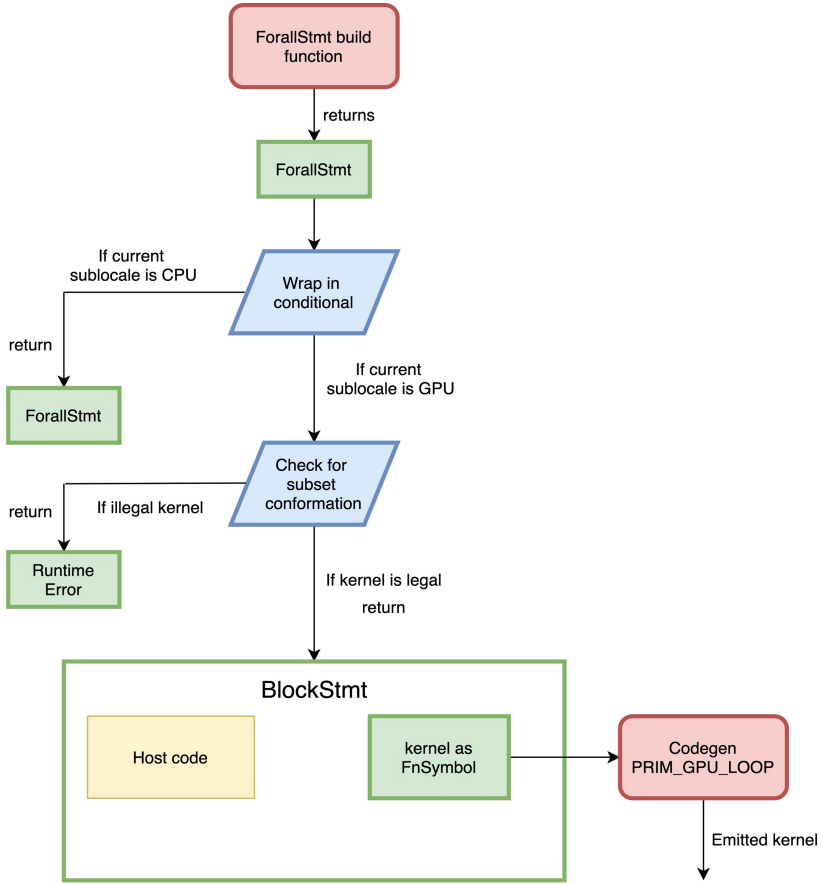
- Infer compile time constants and bake into the kernel
- Similarly, use run-time constants as `__constant`, allowing architectures to utilize cache instead of global memory.
- Another more aggressive improvement (experimental), split and transfer arrays to local memory based on usage for faster access

Current : Using LLVM GPGPU backends

- Chapel has been gradually moving towards making LLVM the default backend
- Our current focus is on NVPTX and AMDGPU backends for NVIDIA and AMD GPUs respectively, and SPIR-V for Intel GPUs
- The idea is to directly emit IR kernels and generate corresponding host code

Forall Loops

- Simply reuse compiler *ForallStmt*
- Minimally invasive on the compiler



Defining a subset of Chapel

- Disallow external function calls
- Disallow on blocks
- Disallow begin/cobegin statements
- Disallow nested parallel loops inside forall loops intended for GPU execution
- Restrict array usage to index expressions only

Defining Work group sizes

Approach 1

- Coforall wrapped forall loops to define the number of work groups spanning each dimension, and in turn, the work group size

Approach 2

- Propose language extensions to support defining work group sizes using with intents

Lastly – A toy implementation GPUArray(s)

```
use GPU;

var a : [1..10] int(32) = 12;
var b : [1..10] int(32) = 13;
var arr = new GPUArray(a);
var arr2 = new GPUArray(b);

var carr : GPUArray(int) = 2 * arr -
                           new GPUArray(b);

var darr : GPUArray(int) = carr * arr2 - arr;

darr.compute();
```

- Lazily evaluated arithmetic operations
- Builds up an intermediate representation (currently Strings)
- Evaluated only for arrays that call `compute()`