GPU support for Scala with Tasty

Student

Reto Ehrensperger

Ausgangslage: Graphics processing units (GPUs) can be used to speed up programs for general purposes. Although frameworks for programming the GPU are available, it is still a difficult task to implement such programs. In addition, the well-known CUDA and OpenCL programming models are only available for C or C++, making it even more difficult for developers of other languages, such as Scala, to accelerate their programs using the GPU. The Firepile library addresses this limitation by providing methods for Scala programs that developers can use to accelerate their code. It creates an abstract syntax tree of bytecode that is then parsed to generate the OpenCL kernels. Since the release of Scala 3, the related compiler generates additional files with the extension .tasty, which already contain the abstract syntax tree.

Ziel der Arbeit: We describe an approach to improving Firepile by using these generated tasty files instead of recreating the abstract syntax tree from the bytecode. The goal is to make the library more independent from third-party libraries, making it easier to maintain in the future. This experiment provides an initial system for simplifying GPU acceleration for Scala developers, focusing on kernel building rather than performance improvement.

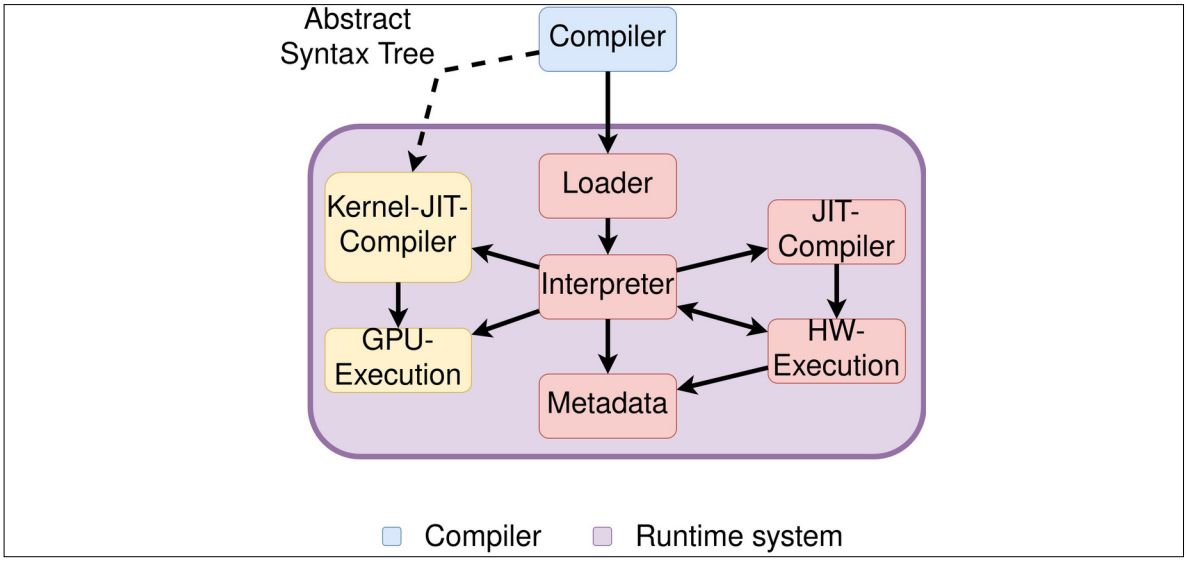
Implementation of a Matrix multiplication
Eigene Darstellung

```
1  val m = 8
2  val n = 4
3  val k = 2
4
5  result = Space.spawn(A,B,C){
6    val i = Space.global_id_0
7    val j = Space.global_id_1
8    if(i < m){
9      if(j < n){
10       var sum = 0.0f
11       for(x <- 0 until k) {
12         sum = sum + A(i*k+x) * B(x*n+j)
13       }
14       C(i*n+j) = sum
15     }
16   }
17   C
18 }
```

Resulting kernel for the matrix multiplication
Eigene Darstellung

```
1  __constant int m = 8;
2  __constant int n = 4;
3  __constant int k = 2;
4
5  __kernel void matrixMul(__global float* A,__global
       float* B,__global float* C){
6    int i = get_global_id(0);
7    int j = get_global_id(1);
8    if (i <m){
9      if (j <n){
10       float sum=0.0;
11       for(int x = 0; x < k; x++){
12         sum = sum + A[i *k +x] * B[x *n +j];
13       }
14       C[i *n +j] = sum;
15     }
16   }
17 }
```

Procedure to generate kernel code
Eigene Darstellung



Referentin
Prof. Dr. Mitra Purandare

Themengebiet
Computer Science