This is fairly long, but my hope is to turn this into a blog article so I have captured some ramblings as I go along.

I was preparing to refactor Aparapi to allow the following form of code

**Aparapi.forEach(<range>,**

**(gid)->{ squares[gid]=in[gid]\*in[gid]}**

**);**

So I create a Aparapi helper with an inner SAM type

**public class Aparapi{**

**public interface SAM{**

**run(int gid);**

**}**

**public void forEach(int range, SAM sam){**

**for (int i=0; i<range; i++){**

**sam.run(i);**

**}**

**}**

**}**

And everything works magically!

Here is my test code

**public class Main{**

**public static void main(String[] args){**

**final int in[] = new int[100];**

**final int squares[] = new int[100];**

**// fill in[]**

**Aparapi.forEach(<range>,**

**(gid)->{ squares[gid]=in[gid]\*in[gid]}**

**);**

**// use squares[]**

**}**

**}**

I pass a range (say 100) and a lambda implementation of my Aparapi.SAM type and the Aparapi.SAM.run(int gid) method is indeed executed 100 times , each with a unique value of gid.

Of course for Aparapi I need to grab the bytecode of the run method of the SAM type so that I can convert it to OpenCL. I can then do some buffer magic and execute on the GPU.

When I compile using the –XdlambdaToMethod (using the latest lambda tree from project lambda website) sure enough, I see an anonymous inner class created which has the bytecode for my lambdafied Aparapi.SAM.run() method.

However when I compile with the –XdlambdaToMethod, I get all sorts of goodies in my class file ;) but cannot for the life of me find the bytecode for my lambdafied Aparapi.SAM.run() method.

Links/References

<http://openjdk.java.net/projects/lambda/>

<http://cr.openjdk.java.net/~briangoetz/lambda/lambda-translation.html>