Keith Campbell – IBM Java Technology Center

March 20th, 2015

Hands-on Lab: Accessing the GPU from Java

S5823

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

About me

- A member of the Java Technology Centre in Ottawa, Canada
- Working on various tools & runtime technologies for >20 years
- Experience with open source communities
- Currently focused on bringing the power of GPUs to the Java ecosystem

keithc@ca.ibm.com

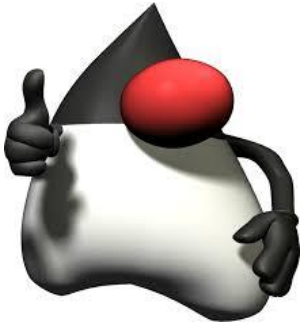
Workshop objectives

- Learn how to take an existing C kernel and build it so that it can be used with CUDA4J
- See how an application can control the process of data transfer and invoke the kernel, all from Java using CUDA4J
- Get a sense of how much JNI code you can avoid writing by using CUDA4J
- See how the Java 8 enablement of lambdas for GPU can remove the need for coding in C altogether

Workshop outline

- 15 minutes of slides which will introduce CUDA4J and Java 8 lambda support
- 40 minutes of hands-on exploration with several implementations of Conway's Game of Life

CUDA4J: Bringing GPU programming to Java

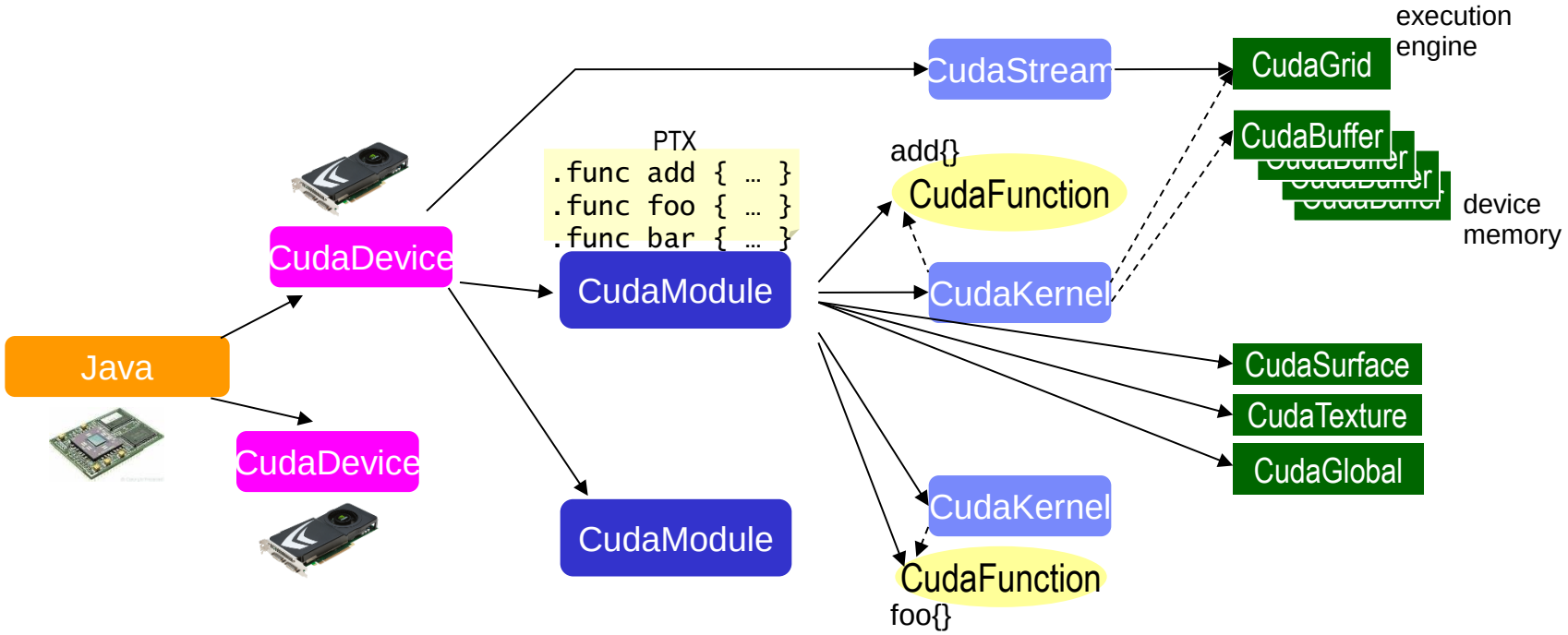


- A Java API that reflects the concepts familiar in CUDA programming
- Makes use of Java idioms: exceptions, automatic resource management, etc.
- Handle copying data to/from the GPU, flow of control from Java to GPU and back, etc.
- Ability to invoke existing GPU code modules from Java applications

new Java APIs

- CudaDevice – a CUDA capable GPU device
- CudaBuffer – a region of memory on the GPU
- CudaModule – user library of kernels to loaded onto the GPU
- CudaKernel – for launching a device function
- CudaFunction – a kernel's entry point
- CudaEvent – for timing and synchronization
- CudaException – for when something goes wrong

Fundamental types in CUDA4J



CudaLinker — Used to combine multiple cubin/fatbin/PTXs into single module

CudaEvent — Device events

Corresponds to a HW feature in GPU

Relationship for generating an instance

Relationship as an argument

Explicit GPU vector addition in Java

```
void add(int[] a, int[] b, int[] c) throws CudaException, IOException {
   CudaDevice device = new CudaDevice(0);
   CudaModule module = new CudaModule(device, getClass().getResourceAsStream("ArrayAdder"));
   CudaKernel kernel = new CudaKernel(module, "Cuda_cuda4j_samples_adder");
   CudaGrid grid = new CudaGrid(512, 512);

   try (CudaBuffer aBuffer = new CudaBuffer(device, a.length * 4);
        CudaBuffer bBuffer = new CudaBuffer(device, b.length * 4)) {
      aBuffer.copyFrom(a, 0, a.length);
      bBuffer.copyFrom(b, 0, b.length);

      kernel.launch(grid, new CudaKernel.Parameters(aBuffer, aBuffer, bBuffer, a.length));

      aBuffer.copyTo(c, 0, a.length);
   }
}
```

first GPU device

native module containing the kernel

grid of kernels that will execute this task

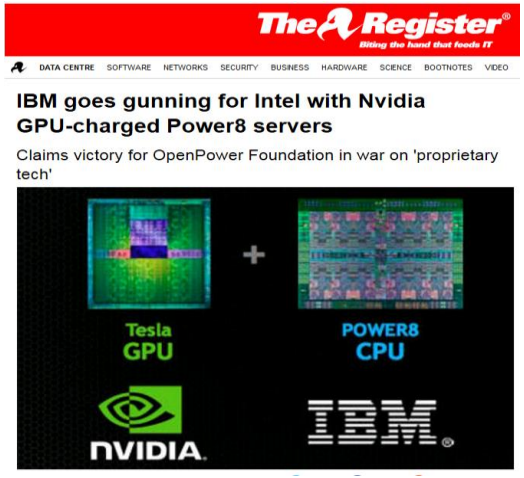
move data from Java heap to device

invoke the task

move the result back to the Java heap

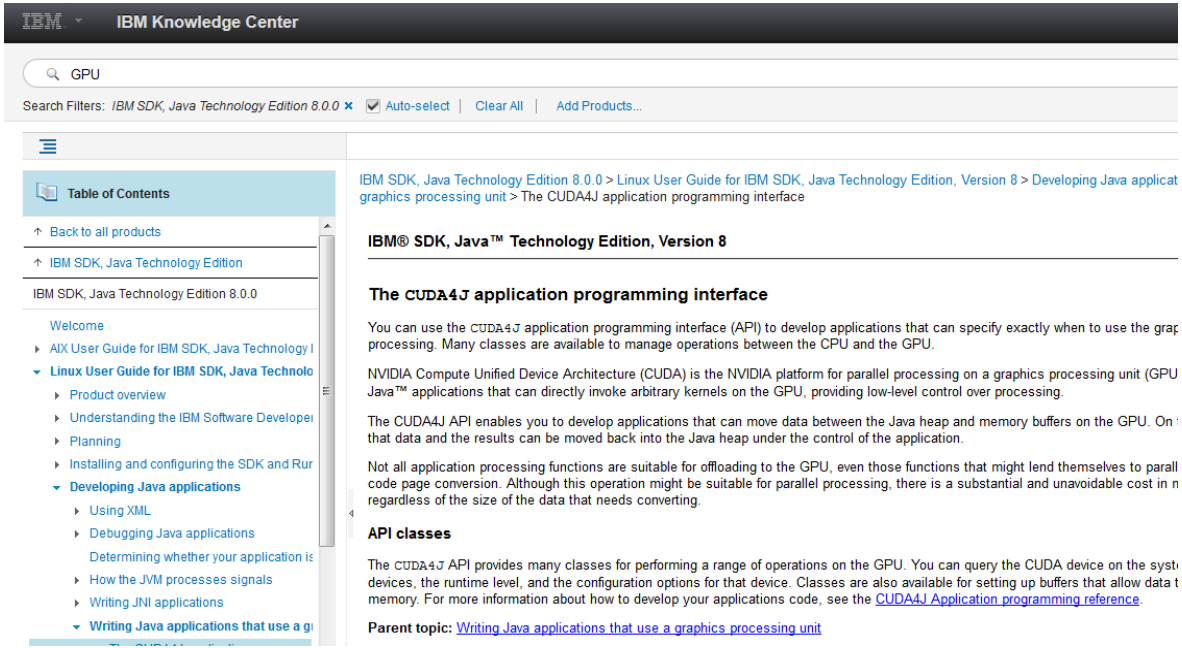
Availability

- Currently available in Java 7.1 and Java 8 on POWER 8 Little Endian, running Ubuntu 14.10 and CUDA 5.5-54
- Download Java at http://www.ibm.com/developerworks/java/jdk/linux/download.html
- Supported hardware is POWER 8 model 824L with one or two Tesla K40m GPUs



CUDA4J documentation

- http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/user/gpu_developing_cuda4j.html



Beyond specific APIs – Java 8 streams

- Streams allow developers to express computation as aggregate parallel operations on data
- For example:

```
IntStream.range(0, N).parallel().forEach(i -> c[i] = a[i] + b[i]);
```

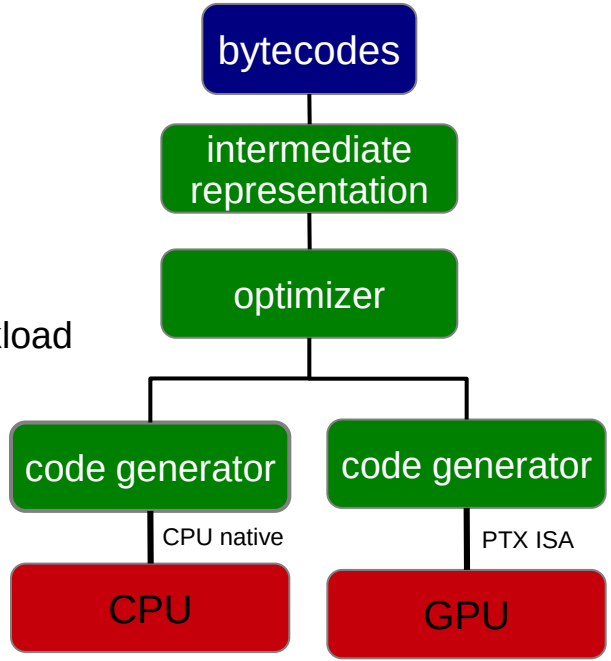
creates a stream whose operations can be executed in parallel

- What if we could recognize the terminal operation and conduct it on the GPU?
  - √ reuses standard Java idioms, so no code changes required
  - √ no knowledge of GPU programming model required by the application developer
  - – but no low-level manipulation of the device – the Java implementation has the controls
  - – future smarts introduced into the JIT do not require application code changes

JIT optimized GPU acceleration

- As the JIT compiles a stream expression we can identify candidates for GPU off-loading
  - arrays copied to and from the device implicitly
  - Java operations mapped to GPU kernel operations
  - preserves the standard Java syntax and semantics

- Early steps
  - Recognize a limited set of operations within lambda expressions,
    - notably, no object references maintained on GPU
  - Default grid dimensions and operating parameters for the GPU workload
  - Redundant/pessimistic data transfer between host and device
    - not using GPU shared memory
  - Limited heuristics about when to invoke the GPU and when to generate CPU instructions



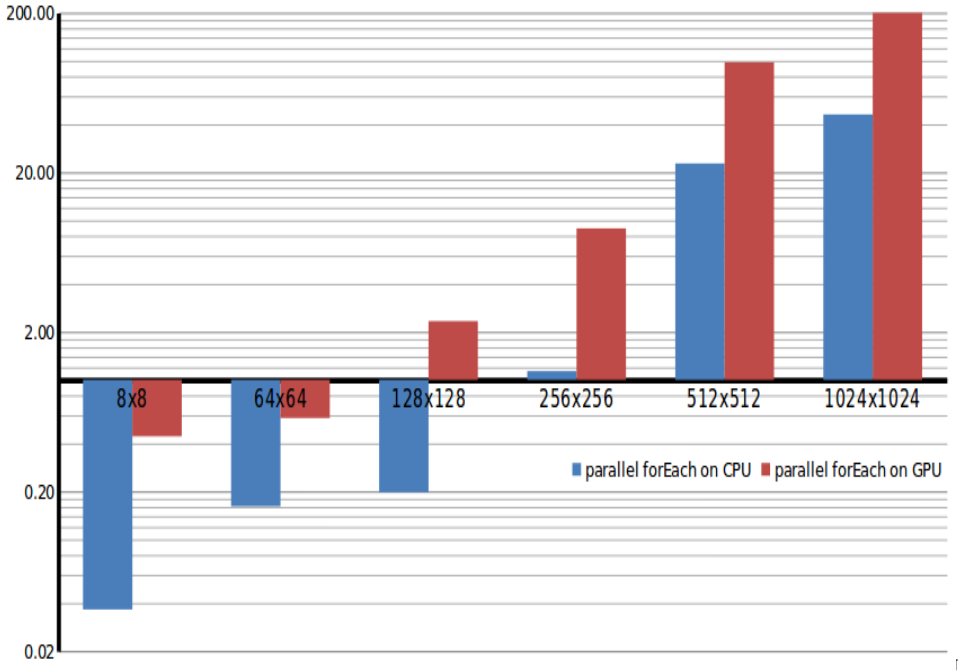
JIT / GPU optimization of Lambda expression

JIT recognized Java code for matrix multiplication using Java 8 parallel stream

```
public void multiply() {
  IntStream.range(0, COLS*COLS).parallel().forEach(
    id -> {
      int i = id / COLS;
      int j = id % COLS;
      int sum = 0;

      for (int k=0; k<COLS; k++) {
         sum += left[i*COLS + k] * right[k*COLS + j]
      }
      output[i*COLS + j] = sum;
  });
}
```

Speed-up factor when run on a GPU enabled host



JIT heuristics for when a lambda can compile for the GPU

- Your mileage may vary, but the intention is:
- Supported:
  - variables and one-dimensional arrays of all Java primitive types
  - locals, parameters, and instance variables. Static variables generally will not be supported but in some cases can be handled by JIT
  - all Java operators except “instanceof”
  - all Java expressions and statements except for new, throw, and method invocations
  - standard Java exceptions: NPE, AOB, arithmetic
- Not supported:
  - method invocations, although some method invocations might be handled by JIT
  - intermediate operations like map, filter, etc.
  - user exceptions

Enabling the JIT compilation of code for the GPU

- See http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendixes/cmdline/xjit.html%23xjit__gpu

-Xjit:enableGPU={default|enforce|verbose} (and combinations)
where:

- *default* enables graphics processor unit (GPU) support based on performance heuristics
- *enforce* attempts to send all parallel.forEach() calls to the GPU irrespective of performance heuristics
- *verbose* generates output that reports which parallel.forEach() calls are sent to the GPU

For example:

-Xjit:enableGPU={enforce|verbose}

Hands-on

- Conway's Game of Life
  - cellular automata
  - hence so-called “embarrassingly” parallel
- Suggested activities:
  - Make sure you understand what Game of Life is:
    - http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
    - check out evolution of acorn at http://pmav.eu/stuff/javascript-game-of-life-v3.1.1/
    - 5206 generations to stabilize on a 300 x 280 grid → ~437M cell calculations
  - Compile the CUDA4J implementation and run the C kernel with it
  - Tinker to your heart's content with the kernel. For example, replace the double-nested loop over neighbors with a single non-looping statement that accumulates the sum of the nine cells. Rerun. Depending on how busy the machine is, you may be able to observe a performance effect.
  - Optional: Run the lambda version. Use the necessary JIT options to enable the lambda to run on the GPU and observe (see run-lambda.sh).

Copyright and Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., and registered in many jurisdictions worldwide.

Other product and service names might be trademarks of IBM or other companies.

A current list of IBM trademarks is available on the Web – see the IBM “Copyright and trademark information” page at URL: www.ibm.com/legal/copytrade.shtml