OpenPOWER Software Stack with Big Data Example

March 2014

Driving industry innovation

The goal of the OpenPOWER Foundation is to create an open ecosystem, using the POWER Architecture to share expertise, investment, and server-class intellectual property to serve the evolving needs of customers.

- Opening the architecture to give the industry the ability to innovate across the full Hardware and Software stack
  - Simplify system design with alternative architecture
  - Includes SOC design, Bus Specifications, Reference Designs, FW OS and Hypervisor Open Source
  - Little Endian Linux to ease the migration of software to POWER
- Driving an expansion of enterprise class Hardware and Software stack for the data center
- Building a complete ecosystem to provide customers with the flexibility to build servers best suited to the Power architecture

Giving ecosystem partners a license to innovate

OpenPOWER will enable data centers to rethink their approach to technology.

Member companies may use POWER for custom open servers and components for Linux based cloud data centers.

OpenPOWER ecosystem partners can optimize the interactions of server building blocks – microprocessors, networking, I/O & other components – to tune performance.

How will the OpenPOWER Foundation benefit clients?

- –OpenPOWER technology creates greater choice for customers
- –Open and collaborative development model on the Power platform will create more opportunity for innovation
- –New innovators will broaden the capability and value of the Power platform

What does this mean to the industry?

- –Game changer on the competitive landscape of the server industry
- –Will enable and drive innovation in the industry
- –Provide more choice in the industry

Platinum Members

ALTERA Google IBM Mellanox Micron NVIDIA SAMSUNG PowerCore TYAN

Proposed Ecosystem Enablement

OpenPOWER™

Power Open Source Software Stack Components

Software

Cloud Software — openstack CLOUD SOFTWARE

Standard Operating Environment (System Mgmt) — Ganglia, KCAT, oVirt

Operating System / KVM — Linux

Firmware — OpenPOWER Firmware

Hardware — OpenPOWER Technology

Existing Open Source Software Communities

New OSS Community

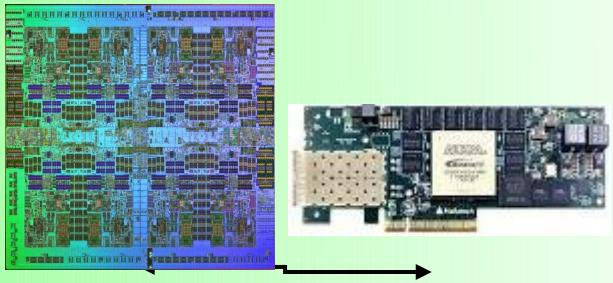
System Operating Environment Software Stack

A modern development environment is emerging based on this type of tools and services

Hardware

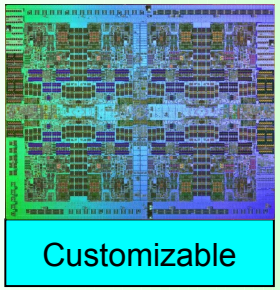
Multiple Options to Design with POWER Technology Within OpenPOWER

POWER8



CAPI over PCIe

Framework to Integrate System IP on Chip



Industry IP License Model

"Standard POWER Products" – 2014

"Custom POWER SoC" – Future

Software Stack

OpenPOWER Enables Hybrid Cloud

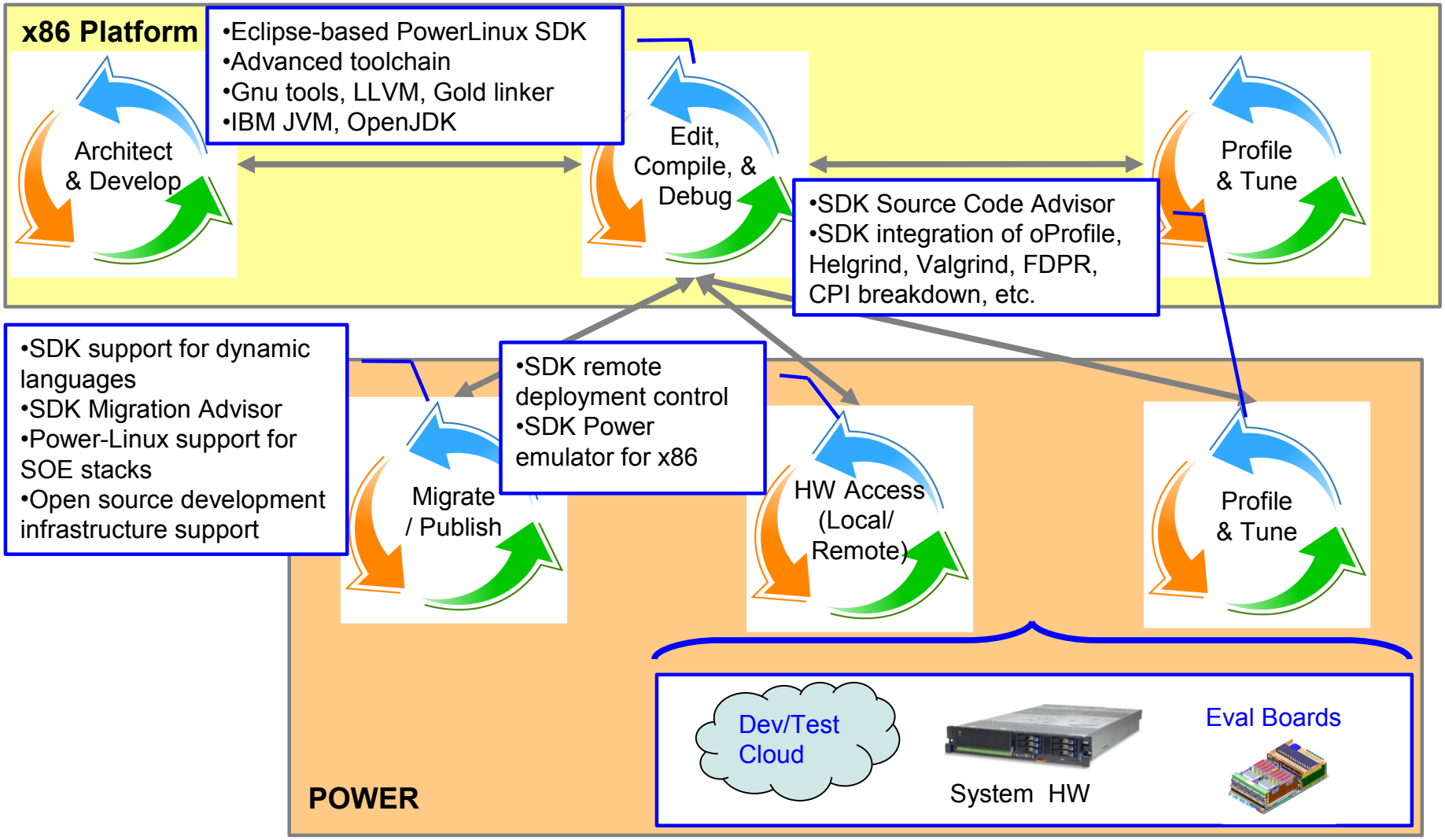
OpenPOWER SW Stack Commonality

- Linux – to provide commonality for:
  - Operating system management
  - Operating system features
  - Application programming model
- Little Endian – to provide source code and data commonality
- KVM – to provide virtualization management and feature commonality
- Firmware interfaces – to provide platform management commonality

Migrating Software to OpenPOWER

- Software written in interpreted languages (Javascript, PHP, Perl, Python, Ruby, Java, etc.)
  - Generally, no work is required.
- Software written in compiled languages (C/C++, Fortran, etc.)
  - Generally, this requires just a simple recompile for POWER.
- Rarely, dependencies on specific behaviors can require source code modification:
  - Multi-threaded applications that don't use standard synchronization models and depend upon specific memory ordering behavior (unusual)
  - Applications that depend upon specific memory page sizes (rare)

Development for PowerLinux



LE Toolchain & Development Ecosystem

- Build Tools
  - GCC (+binutils+GLIBC) toolchain (AT native and cross)
  - config.guess, config.sub, M4, automake, autoconf, libtool, pkg-config, …
  - Gcov, rpm, Perf, SystemTap
- Profiling and Tracing Tools
  - Oprofile (operf, ocount, perf-events),
  - Valgrind (memcheck, massiff, cachegrind, hellgrind, itrace)
- PowerLinux Multi-core Runtime
  - TBB, CBB, userRCU, SPHDE, tcmalloc, Boost, ...
- IBM Tools (binary only)
  - FDPR (bprober)
  - Pthread_mon (sync_mon)
  - Mambo (simulator)
- Eclipse SDK Tools
  - PPC64LE JVM (prereq - binary only)
  - PPC64LE Eclipse IES: Native code + CDT, PTP, LTP, ...
  - PPC64LE Eclipse PowerLinux plugins: MA, SCA, CPI, TA, ...

CUDA / GPU / POWER Technology Kit

Technology Kit Available in 3Q2014

- CUDA toolkit and runtime for POWER
- Linux for POWER
- NVIDIA Tesla GPUs
- POWER platform

- Contact – Richard Talbot, IBM
- Email – rdtalbot <at> us.ibm.com

Java GPU Acceleration

Java GPU Acceleration on OpenPOWER

- IBM and NVIDIA partnering to bring GPU acceleration to mainstream Java workloads and enterprise environments on OpenPOWER
- Evolve the Java support to enable both transparent Java acceleration and Java programmer enablement to program explicitly for GPUs from Java
- Key compute-intensive algorithms in Java standard libraries and domain-specific libraries to be accelerated with GPU
- First step was to build a GPU control and runtime framework in Java
  - Allow Java programmers to do data transfers and launch GPU kernels
  - Gain experience in APIs and GPU programming models applied to Java
- Demo!

Java GPU Framework Overview

- Exploring Java equivalents of CUDA concepts
  - Reuse concepts defined by CUDA in Java
  - Improve usability with familiar Java idioms

- Key mappings from Java to CUDA
  - CudaDevice – a CUDA-capable device
  - CudaBuffer – a region of device memory
  - CudaException – for when something goes wrong
  - CudaModule – code loaded onto a device
  - CudaFunction – a kernel entry point
  - CudaKernel – for launching a device function
  - CudaEvent – for timing or inter-stream synchronization
  - CudaStream – a CUDA stream

- Key Benefits
  - Java ‘write once, run anywhere’ extended to include GPU code
  - Using exceptions removes the tedium of error checking and avoids problems of omissions
  - Provides a foundation for building more general class libraries and reusing existing GPU code

Java GPU Framework – Device Management

- // how many devices do we have?
  int deviceCount = CudaDevice.getCount();
- // create a handle for the first device
  CudaDevice device = new CudaDevice(0);
- // how many multiprocessors does the device have?
  int smCount = device.getAttribute(
  CudaDevice.ATTRIBUTE_MULTIPROCESSOR_COUNT);
- // add a Runnable callback (to the default stream)
  device.addCallback(action);
- // wait for a device to complete queued work
  device.synchronize();

Prototype API shown here is subject to change or withdrawal without notice, and represents goals and objectives only.

Java GPU Framework – Memory Management

- // allocate memory on a device
  CudaBuffer buffer = new CudaBuffer(device, byteCount);
- // copy data from Java array to device memory
  buffer.copyFrom(array, startIndex, endIndex);
- // copy data to Java array from device memory
  buffer.copyTo(array, startIndex, endIndex);
- // clear device memory
  buffer.fillByte(0, byteCount);
- // release device memory
  buffer.close();

Java GPU Framework – Launching Kernels

- // load a module onto a device
  CudaModule module = new CudaModule(device, content);
- // locate a kernel function
  CudaKernel kernel = new CudaKernel(module, kernelName);
- // define grid size
  CudaGrid grid = new CudaGrid(gridDim, blockDim);
- // build bundle of launch arguments
  CudaParameters args = new CudaParameters(2);
  args.set(0, buffer);
  args.set(1, n);
- // launch kernel
  kernel.launch(grid, args);

Java GPU Framework – Events

- // create a new event
  CudaEvent event = new CudaEvent();
- // queue an event
  device.record(event);
  stream.record(event);
- // wait until an event has occurred
  event.synchronize();
- // check whether an event has occurred
  int errorCode = event.query();
- // query the time between events
  float millis = event.elapsedTimeSince(priorEvent);
- // destroy an event
  event.close();

Java GPU Framework – Streams

- // create a new stream
  CudaStream stream = new CudaStream(device);
- // make stream wait for an event
  stream.waitFor(event);
- // wait until all items have been processed
  stream.synchronize();
- // check whether all items have been processed
  int errorCode = stream.query();
- // destroy a stream
  stream.close();

Automatic Resource Management

- Relevant classes implement AutoCloseable to enable automatic release via Java try-with-resources statement
  - Manual memory management with fewer errors

```
int[] data = ...

try (CudaBuffer buffer = new CudaBuffer(device,
data.length * 4)) {
    // copy from Java to device
    buffer.copyFrom(data, 0, data.length);
    // launch kernels, etc.
    // copy from device to Java
    buffer.copyTo(data, 0, data.length);
} // buffer.close() called automatically
```

GTC Demo

- Motivation
  - Large global retailers collect petabytes of data from customer transactions
  - They want to identify customers with similar behavior, so they can create customized products and more effective marketing programs
  - K-Means is a well-known algorithm for performing clustering analysis that can identify these segments
  - Businesses that react quickly to changes in market segments will have an advantage: we want a faster implementation
  - Apache Mahout is an open-source Java implementation which is applicable to 'Big Data' because it is built on top of Apache Hadoop

- Can we do better using GPUs?

GTC Demo

- K-Means Algorithm Overview

- Input consists of:
  - D – the number dimensions in each data point
  - K – the number of clusters sought
  - a set of N sample data points
  - a set of K points which are approximate representatives of the clusters

- The output is an improved set of cluster representatives

- The process iterates until the set of representatives stabilizes

GTC Demo

- K-Means Map-Reduce Structure

- Each point is logically mapped to a matrix with K rows and 2D+1 columns (the table below assumes D=2)
  - The non-zero row corresponds to the closest input cluster center

Count	X	Y	X^2	Y^2
0	0	0	0	0
…	0	0	0	0
1	x	y	x^2	y^2
…	0	0	0	0
0	0	0	0	0

- The reduction phase computes the simple sum of these matrices.
  - From this, the output centers and radii can be derived

GTC Demo

- K-Means Map-Reduce Structure (continued)

- The map computation can be parallelized (at a finer granularity than managed by Hadoop); a new mapper class was created
  - Batches of points are transferred to a GPU for mapping
  - A partial reduction is returned to the host

- A single Mahout class was modified – only to allow a system property to select the (new) mapper class

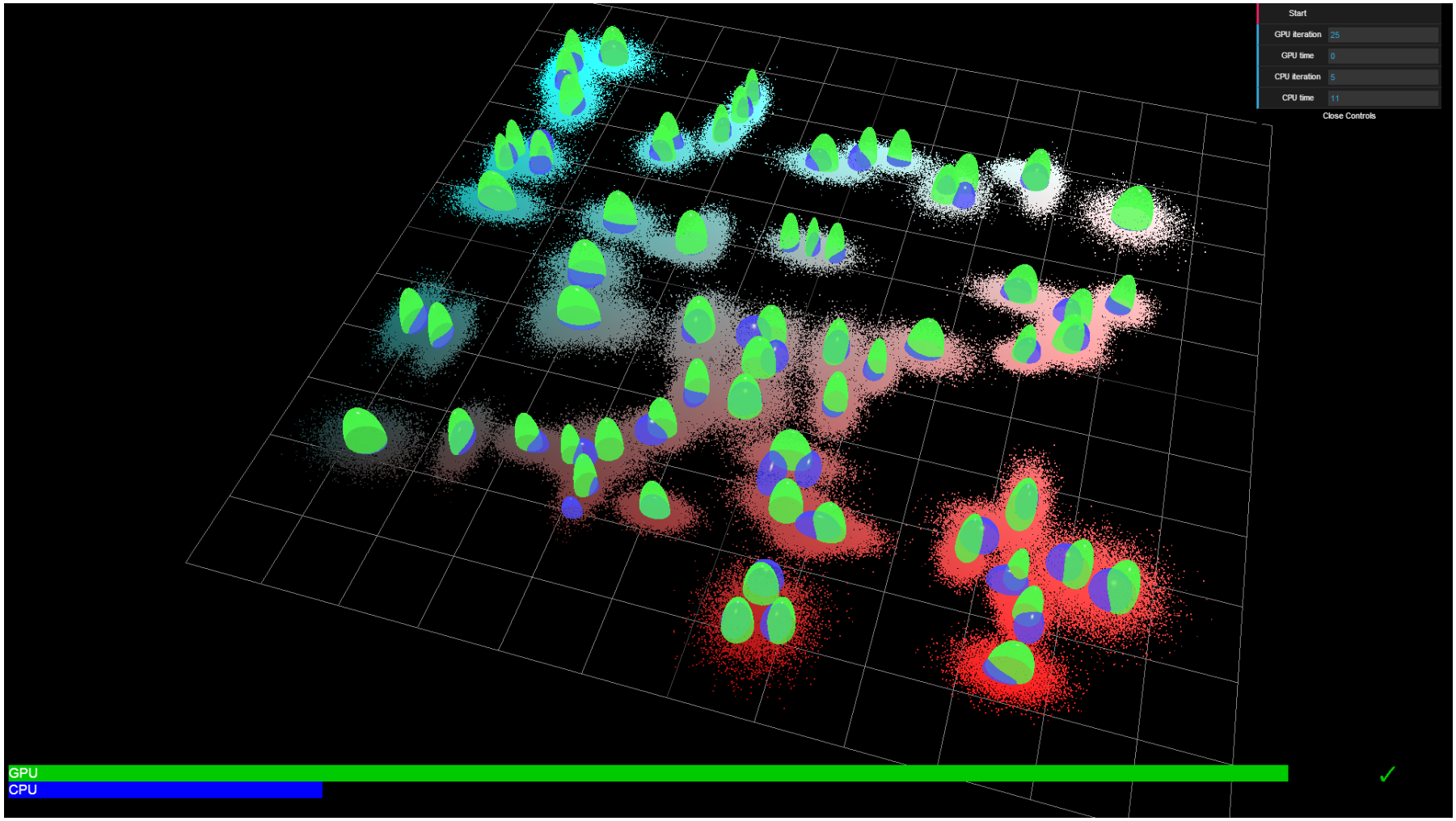
GTC Demo

- Performance Comparison
  - NVIDIA K40 GPU Speed-up factor vs. modern CPU (higher is better)
  - Baseline is standard Mahout implementation
  - Comparison excludes Hadoop framework and I/O

	D=1	D=2	D=4	D=8	D=16	D=32	D=64
K=2	197	120	85	56	38	29	20
K=4	183	119	92	61	40	27	19
K=8	286	134	104	66	42	26	18
K=16	400	160	117	77	44	24	17
K=32	529	261	155	92	49	22	22
K=64	661	388	201	113	55	34	24

Measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.

GTC Demo – Clustering of Big Data



What's Next?

- Java 8 platform adding lambda support
  - Concise syntax for writing small snippets of code used in bulk operations (kernels)
  - Perfect for GPUs, parallel operation and high core count machines
- Standard class libraries will leverage GPUs where it makes sense
  - Bulk operations (map, reduce) on Java collections
  - New parallel operations defined as part of a standard JCP process
- Java platform needs to evolve for best performance
  - optimal layout and control of memory needed for highest performance
    - IBM PackedObjects, Oracle value types evolving for future Java platform definitions
- Demo in booth #921 in the Exhibit Hall
- Related talks:
  - NVIDIA compiling Java kernels to PTX as a proof of concept – see their talks
    - S4830 – CUDA 6 and Beyond
    - S4939 – Accelerating Java on GPUs

OpenPOWER™

Questions?