Challenges in Transition

Keynote talk at International Workshop on Software Engineering Methods for Parallel and High Performance Applications (SEM4HPC 2016)

Kazuaki Ishizaki

IBM Research – Tokyo
kiszk@acm.org
What is this talk about?

- How we make a HPC platform consumable for non-HPC people?
  - For machine learning (ML) and deep learning (DL)

- This talk is not a solid research proposal, but what I am recently thinking about.
Takeaways

- Users, applications, and HWs are always in transition
  - Programming is becoming hard

- Let us build an end-to-end runtime system for ML and DL
  - Leave each layer to the specialist to do the best
  - Each layer should know everything for optimizations
    - Should not be isolated

- How we can make state-of-the-art technologies consumable in the system?
  - Our research is here!
My History (mostly commercial, sometimes HPC)

1990-1992  Network HW interface for parallel computer
1992-1995  Static compiler for High Performance Fortran
1996-now  Just-in-time compiler for IBM Developers Kit for Java
  - 1996-2000  Benchmark and GUI applications
  - 2000-2010  Web and Enterprise applications
  - 2012-
    - 2014-  Java language with GPUs
    - 2015-  Apache Spark (in-memory data processing framework) with GPUs
Outline of this talk

- Review transition in HPC
- What are problems in this transition?
- How we will address these problems?
Performance Trend of TOP500

- Great performance improvements for Linpack, at 33.86PFlops
TOP #1 Systems in TOP500

1975: Cray-1
1993: CM-5
1997: ASCI Red
2004: BlueGene/L
2008: RoadRunner
2010: Tianhe-1A
2013: Tianhe-2

Source: TOP500
TOP #1 Systems in TOP500

- 1024 processors
- 7264 processors
- 65536 processors
- 1024 processors

- Cell processor
- GPU
- Xeon Phi

Vector processor

199306 199511 199806 200011 200306 200511 200806 201011 201306 201511
Three Eras in HPC

**Vector Era** (-1993)
- 1024 processors
- Vector processor

**MPP Era** (1990-)
- 7264 processors

**Accelerator Era** (2008 -)
- 65536 processors
- Cell processor
- GPU
- Xeon Phi

* MPP: Massively Parallel Processing
Review for Each Era

- What applications were executed?
- Who wrote these applications?
- What research we did?
- What was commodity HW?
Vector Era

Vector Era (-1993)

Vector processor

*MPP: Massively Parallel Processing*
Vector Era (-1993)

- How we can exploit a vector machine for specific applications

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
<td>Slow scalar processor with vector facility</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td>Weather, wind, fluid, and physics simulations</td>
</tr>
<tr>
<td><strong>Programmers</strong></td>
<td>Limited # of programmers who are well-educated for HPC (Ninja</td>
</tr>
<tr>
<td></td>
<td>programmers)</td>
</tr>
<tr>
<td><strong>Research</strong></td>
<td>Automatic vectorization techniques</td>
</tr>
<tr>
<td></td>
<td>Enhancement of vector HW features (e.g. sparse array support)</td>
</tr>
<tr>
<td><strong>Commodity HW</strong></td>
<td>Slow scalar processor</td>
</tr>
</tbody>
</table>
MPP Era

*MPP: Massively Parallel Processing*
MPP Era (1990 -)

- How we can hide latencies between nodes

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Massive commodity processors with special network I/F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Simulations for wider areas (e.g. chemical synthesis)</td>
</tr>
<tr>
<td>Programmers</td>
<td>Limited # of programmers who are well-educated for HPC</td>
</tr>
</tbody>
</table>
| Research   | Improvements on MPI implementations  
Parallelization and optimization of given applications by hand |
| Commodity HW | Fast scalar processors |
Accelerator Era (2008 - )

*MPP: Massively Parallel Processing
Innovations in System Software

- CUDA/OpenCL make powerful computing resource accessible
Accelerator Era (2008 - )

- How we can exploit GPUs in our applications

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Massive commodity processors with HW accelerators</td>
</tr>
<tr>
<td>Applications</td>
<td>Simulations for wider areas (e.g. chemical synthesis)</td>
</tr>
<tr>
<td>Programmers</td>
<td>Limited # of programmers who are well-educated for HPC</td>
</tr>
<tr>
<td>Research</td>
<td>GPU-friendly rewriting of given applications by hand</td>
</tr>
<tr>
<td></td>
<td>GPU-oriented algorithms</td>
</tr>
<tr>
<td>Commodity HW</td>
<td>Desktop PC with GPU cards</td>
</tr>
</tbody>
</table>
Innovations in Programming Environment

- MapReduce makes parallel programming easy
Innovations in Infrastructure

- Cloud makes a cluster of machines easily accessible
Big innovations in Applications

- Machine learning and deep learning are big FP consumers

Source: The analytic store, Deep Learning with GPUs
Accelerator Era 2.0 (2012 - )

- HPC meets machine learning and deep learning with big data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Massive commodity processors with HW accelerators</td>
</tr>
<tr>
<td>Applications</td>
<td>Machine learning (ML) and deep learning (DL) with big data</td>
</tr>
<tr>
<td>Programmers</td>
<td>Data scientists who are non-familiar with HPC</td>
</tr>
<tr>
<td>Research</td>
<td>How we can effectively use GPUs?</td>
</tr>
<tr>
<td></td>
<td>How about accuracy of a new ML/DL algorithm with big data?</td>
</tr>
<tr>
<td>Commodity HW</td>
<td>A cluster of machines with GPUs on cloud</td>
</tr>
</tbody>
</table>
Summary of Transition

- Majorities of applications are changing
  - From simulations to machine learning (ML)/deep learning (DL) with big data

- HPC HW is becoming commodity
  - GPUs are available on desktop and cloud
  - Cloud provides a cluster of GPUs as a commodity

- Programmers are changing
  - From Ninja programmers to data scientists
Outline of this talk

- Review transition in HPC
- What are problems in this transition?
- How we will address these problems?
Details in Application

- Data is becoming rapidly larger
  - 1000x from 2010 to 2015

- The number of applications is rapidly growing
  - arXiv.org is hosting many papers
    - On 2014, hit 1 million articles
    - On 2015, 105,000 new submission and over 139 million downloads
  - github.com is hosting many programs
    - On 2014 4Q, more than 15M updates (pushes) to 2.2M repositories
Details in Programming Languages

- Data Scientists love Python and R (e.g. high level languages)
  - Python and R make programming easy
    - Scientific computing operations and libraries (e.g. Numpy)
  - Programs do not scale to a cluster of machines
    - Perform pre-filterings to reduce data size for a machine
    - Spend much time to rewrite it for a cluster
  - It is not easy to write a program optimized for a target architecture
Details in Infrastructure

- **Accelerators that matter**
  - Processing units
    - GPU, FPGA, ASIC (e.g. Tensorflow Processing Unit), ...
  - Storage
    - Non-volatile memory, phase change memory, ...
  - Communication
    - Communication between accelerators (e.g. NVLINK), optical interconnect, ...
Problems in Future

- Data will be too large to store on fast memory
  - Memory hierarchy is becoming deep

- Programming will be hard
  - Hard to program HW accelerators
  - New applications rapidly appear

- Optimization and deployment will be hard
  - Emerging HW accelerators will appear
Outline of this talk

- Review transition in HPC
- What are problems in this transition?
- How we will address these problems?
My Proposal: Build an End-to-End System

- From an algorithm to hardware
- Leave each layer to the specialist to do the best
  - Easy to develop new algorithms
  - Easy to exploit parallelism from the algorithm
  - Easy to generate accelerator code
    - We should avoid complex tasks (e.g. analysis)
- Each layer should know everything
  - What parts of the algorithm are parallel?
  - What happens at hardware
    - We should not make each layer isolated
Similar Research 1

The Big-ML “Stack” - More than just software

- **Representation**: Compact and informative features
- **Model**: Generic building blocks: loss functions, structures, constraints, priors...
- **Algorithm**: Parallelizable and stochastic MCMC, VI, Opt, Spectrum...
- **Programming model & Interface**: High: Matlab/A, Medium: C/JAVA, Low: MPI
- **System**: Distributed architecture: DFS, KV-store, task scheduler...
- **Hardware**: GPU, flash storage, cloud...

Source: A New Look at the System, Algorithm and Theory Foundations of Distributed Machine Learning
Similar Research 2

System ML

An algorithm written in R subset is translated to an optimized Apache Spark program with information.

\[
HS = t(U) \%\% (W \ast (U \%\% S)) + \text{lambda} \ast S;
\]
Our Recent Research: Exploit GPUs at High Level

- Compile a Java Program for GPUs
  - A parallel stream loop, which explicitly expresses a parallelism, can be offloaded to GPUs by our just-in-time compiler without any GPU specific code

```java
IntStream.range(0, N).parallel().forEach(i -> {
    b[i] = a[i] * 2.0;
});
```
Our Recent Research: Exploit GPUs at High Level

- Apache Spark with GPUs [http://github.com/IBMSparkGPU](http://github.com/IBMSparkGPU)
  - Drive GPU code from an Apache Spark program transparently from a user

```scala
// rdd: resilient distributed dataset is distributed over nodes
rdd = sc.parallelize(1 to 10000, 2) // node0: 1-5000, node1: 5001-10000
rdd1 = rdd.map(i => i * 2)
sum = rdd1.reduce((x, y) => (x + y))
```

Image Source: NVIDIA

```scala
// Example code snippet
rdd
```

```
<table>
<thead>
<tr>
<th>Node</th>
<th>Range</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>1-5000</td>
<td>i * 2</td>
</tr>
<tr>
<td>Node 1</td>
<td>5001-10000</td>
<td>i * 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x+y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sum</td>
</tr>
</tbody>
</table>
```
How We Create this Proposal?

- Will we just pile up existing products?
How We Create this Proposal?

- Will we just pile up existing products?
  - No, it would invent a naïve FAT stack
How We Create this Proposal?

- Will we just pile up existing products?
  - No, it would invent a naïve FAT stack

- I like an abstraction, but do not like to execute it “as-is”
  - Run as an optimized THIN stack with end-to-end optimizations
    - before an execution
    - during an execution
    - among executions

- Do not guess: Each layer should know everything
Our Research Challenges (1/2)

- **Programming environment**
  - Algorithm should be written declaratively without losing high level information

- **Framework / libraries**
  - Resource scheduling
  - Communication-avoiding algorithm
  - Loosely-synchronized execution model
  - Localization (e.g. tiling)

Current ML/DL frameworks have not optimized than HPC software stacks yet
Our Research Challenges (2/2)

- **Programming languages / system software**
  - Make HW accelerators consumable without specific code
  - Dynamic compilation or deployment for new HW accelerators
  - Automatic tuning
    - Deep learning may help too many tuning knobs in system
  - Appropriate feedbacks from HW to programming

- **Debugging**
  - Reproduce a bug for some converged algorithms
Recap: Takeaways

- Users, applications, and HWs are always in transition
  - Programming is becoming hard

- Let us build an end-to-end runtime system for ML and DL
  - Leave each layer to the specialist to do the best
  - Each layer should know everything for optimizations
    - Should not be isolated

- How we can make state-of-the-art technologies consumable in the system
  - Our research is here!