Efficiency + Scalability = High-Performance Data Computing

Zhiwei Xu
Institute of Computing Technology (ICT)
Chinese Academy of Sciences

http://novel.ict.ac.cn/zxu/
zxu@ict.ac.cn
Fundamental Challenge: First time in 70 years

- Energy efficiency improvement lags behind speed growth → research goal: 1000 Gop/J
Big Data Computing Does Not Emphasize Efficiency

- Big data computing cares about five nonfunctional metrics
  - **Fault tolerance**: how well are faults handled
    - Now provides good-enough availability for 100 PB of data
  - **Usability**: how easy is it to write a scalable application
    - Now a sort program on Hadoop only needs a few dozens of lines of code, and need not change with data size (1GB, 1TB, 1PB) or system size (1 node, 100 nodes)
  - Performance metrics
    - **Scalability**: how much data can be handled
      - Now 10 PB data can be sorted in hours; EB data managed
    - **Speed**: 1 / seconds for a given data size
    - **Speed Efficiency**: (sustained) speed / peak speed
      Or **Energy Efficiency**: speed / W = operations / Joule
Four Stages of Jim Gray’s Data Challenge

1. Terasort challenge raised (sort 1TB in 1 minute)
2. Speed growth: TB/minute in 2009 (ad hoc methods)
3. Data size growth: TB→PB→10PB (Map-Reduce)
4. Efficiency: 1PB sorted on 190 AWS instances in 2014
Data Computing Efficiency Is Bad

- Sort 4GB: performance & power consumption
  - SW: Hadoop 0.20.2 over CentOS 5.3
  - HW: one cluster node (2 4-core E5620 CPUs, 16GB mem, 150GB disk)

### Speed Efficiency
(Sustained/Peak)
- Payload: 0.002%
- Linpack: 94.5%
- Total op: 4.22%
- Instruction: 4.72%

### Energy Efficiency
(Operations per Joule)
- Payload: $1.55 \times 10^4$
- Linpack: $7.26 \times 10^8$
- Total op: $2.20 \times 10^7$
- Instruction: $2.45 \times 10^7$
Three Ways to Increase Efficiency

• Functional sensing
  – Significantly reduce amounts of data collected

• Acceleration by elastic processors
  – Dynamically transform hardware to match applications characteristics

• DataMPI
  – Transfer HPC knowledge to data computing
    • Buffer-to-buffer to key-value communication
Functional Sensing Model in the Ternary Universe

Human decides the functionality according to the needs, and defines it with a functionality mapping $\Pi_2$.

We cannot get true value $x$, but only an observation $y_1$.

Build an inferring algorithm $\Lambda$ to calculate an approximate value of $y_2$, i.e., $\Lambda(y_1) \approx y_2 = \Pi_2(x)$.
On-Off State Sensing of $N$ Appliances in a Household with 1 Sensor

- Single Sensor ($\Pi_1$)
- Electric Meter

Total power ($y_1$)

HMM decoder ($\Lambda$)

Output $\Lambda(y_1)$

Error of sensing

On-Off state for each appliance ($y_2$)

Definition of On-Off states ($\Pi_2$)

Full and precise information for each and every individual appliance ($x$)
Bayesian Current Disaggregating (BCD) Results for a Laptop and a Fridge

<table>
<thead>
<tr>
<th></th>
<th>RMSE of OCD</th>
<th>RMSE of BCD</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>42.01 mA</td>
<td>11.86 mA</td>
<td>14.58 mA</td>
</tr>
<tr>
<td>Fridge</td>
<td>50.43 mA</td>
<td>34.49 mA</td>
<td>603.07 mA</td>
</tr>
</tbody>
</table>

**Laptop**

**Fridge**
Inferring Cell-scale Signaling Representational Networks

Cell-scale signaling network ($x$)

Measurements over a specific timescale ($\Pi_1$)

Concentrations of proteins ($y_1$)

The CCELL algorithm ($\Lambda$)

The inferred transition matrix of the representational network ($\Lambda(y_1)$)

RMSE

[0.8  ...  0.1]

[ ...  ...  ...]

[0.0  ...  0.9]

The representational network over a specific timescale ($y_2$)

Vollmeister et al. 2012

en.wikipedia.org/wiki/ELISA
The Experimental Results of CCELL

- The synthetic cell-scale signaling network
  - JAK-STAT, GR, ERK and p38
  - 300 proteins
- **150 antibodies** (tried 30~300 antibodies)
- Concentration variation inference


Elastic Processor

- A new architecture style (FISC)
  - Featuring **function instructions** executed by programmable ASIC accelerators
  - Targeting 1000 GOPS/W = 1 Top/J

**Chip types:**
- Intel X86: 10s
- ARM: 1K
- FISC: 10K

**Power:**
- 10~100W
- 1~10W
- 0.1~1W

**Apps/chip:**
- 10M
- 100K
- 10K
DianNao: A Neural Network Accelerator

- Support multiple neural network algorithms, such as DNN, CNN, MLP, SOM
- Pre-tape-out simulation results: 0.98GHz, 452 GOPS, 0.485W, 931.96 GOPS/W @ 65nm
- ASPLOS 2014 Best Paper

IC Layout in GDSII

700 speedup over Intel Core i7
Three More Accelerators

- **DaDianNao**: An NN supercomputer containing up to 64 chips
  - MICRO’14 best paper
  - 100-250 GOPS/W (@28nm)

- **PuDianNao**: A polyvalent machine learning accelerator
  - ASPLOS’15
  - 300-1200 GOPS/W

- **ShiDianNao**: A vision accelerator for embedded devices (cameras)
  - ISCA’15
  - 2000-4000 GOPS/W (16-bit)

- Compared to **931 GOPS/W @65nm** for DianNao
DaDianNao: An NN Supercomputer

- In average, 450x speedup and 150x energy saving over K20 GPU for a 64-chip system
PuDianNao

- Area: 3.51 mm²
- Power: 596 mW
- Freq: 1 GHz
- Supporting a dozen types of ML algorithms: CNN/DNN, LR, Kmeans, SVM, NB, KNN, CT, ...

DianNao

- Area: 3.02 mm²
- Power: 485 mW
- Freq: 0.98 GHz
- Supporting CNN/DNN
ShiDianNao: An vision accelerator for embedded devices

Speedup:
- 46.38x vs. CPU
- 28.94x vs. GPU
- 1.87x vs. DianNao

Energy saving:
- 4688.13x vs. GPU
- 63.48x vs. DianNao
Hadoop Efficiency Is Low

- Lacks a high-performance communication substrate
  - Use HTTP, RPC, direct Sockets over TCP/IP to communicate
  - Can MPI be used for big data?

**Speed Efficiency**
(Sustained/Peak)

- **Payload:** 0.002%
- **Linpack:** 94.5%
- **Total op:** 4.22%
- **Instruction:** 4.72%

**Energy Efficiency**
(Operations per Joule)

- **Payload:** 1.55x10^4
- **Linpack:** 7.26x10^8
- **Total op:** 2.20x10^7
- **Instruction:** 2.45x10^7
Direct MPI Use not Easy or Scalable

//MapReduce
map (String lineno, String contents) {
    for each word w in contents {
        EmitIntermediate(w, 1);
    }
}

reduce (String key, int value) {
    increment(key, value);
}

//MPI
process mapper:
1st> load input
2nd> parse token
3rd> MPI_Send (serialization)
...

process reducer:
1st> MPI_Recv (Deserialization)
2nd> increment
3rd> save output
...

WordCount via MapReduce: Scalable over 1GB, 1TB, 1PB …
Desired Sort Code via DataMPI: Scalable and Easy to Write

```java
public class Sort {
    public static void main(String[] args) {
        try {
            int rank, size;
            Map<String, String> conf = new HashMap<String, String>();
            conf.put(MPI_D.Constants.KEY_TYPE, java.lang.String.class.getName());
            conf.put(MPI_D.Constants.VALUE_TYPE, java.lang.String.class.getName());
            MPI_D.Init(args, MPI_D.Mode.Common, conf);
            if (MPI_D.COMM_BIPARTITE_0 != null) {
                rank = MPI_D.Comm_rank(MPI_D.COMM_BIPARTITE_0);
                size = MPI_D.Comm_size(MPI_D.COMM_BIPARTITE_0);
                String[] keys = loadKeys(rank, size);
                if (keys != null) {
                    for (int i = 0; i < keys.length; i++) {
                        MPI_D.Send(keys[i], "");
                    }
                }
            } else {
                rank = MPI_D.Comm_rank(MPI_D.COMM_BIPARTITE_A);
                size = MPI_D.Comm_size(MPI_D.COMM_BIPARTITE_A);
                Object[] keyValue = MPI_D.Recv();
                while (keyValue != null) {
                    System.out.println("Task " + rank + " of " + size + " key is "
                            + ((String) keyValue[0]) + ", value is " + (long) keyValue[1]));
                    keyValue = MPI_D.Recv();
                }
            }
            MPI_D.Finalize();
        } catch (MPI_D.Exception e) {
            e.printStackTrace();
        }
    }
}
```

33 lines of code

1 GB, 1 TB, 1 PB
DataMPI.ORG

- **Core**
  - Execution pipeline
  - Key-value communication
  - Native direct IO for buffer management
  - Key-value based checkpoint

- **Profiles**
  - Additional code sets
  - Each profile for a typical mode

- **mpidrun**
  - Communicator creation
  - Dynamic process management
  - Data-centric task scheduling

Command:  
```
$ mpidrun -f <hostfile> -O <n> -A <m> -M <mode> -jar <jarname> <classname> <params>
```

Example:  
```
$ mpidrun -f ./conf/hostfile -O 64 -A 64 -M COMMON -jar ./benchmark/dmb-benchmark.jar dmb.Sort/text/64GB-text/text/64GB-output
```
Hive on DataMPI

A first attempt to propose a general design for fully supporting and accelerating data warehouse systems with MPI

- **Functionality & Productivity & Performance**
  - Support Intel HiBench (2 micro benchmark queries) & TPC-H (22 app queries)
  - Only 0.3K LoC modified in Hive
  - HiBench: 30% performance improvement on average
  - TPC-H: 32% improvement on average, up to 53%

Lu Chao, Chundian Li, Fan Liang, Xiaoyi Lu, Zhiwei Xu. *Accelerating Apache Hive with MPI for Data Warehouse Systems*. ICDCS 2015, Columbus, Ohio, USA, 2015
References

• Functional Sensing

• Elastic Processing

• DataMPI
  – Lu Chao, Chundian Li, Fan Liang, Xiaoyi Lu, Zhiwei Xu. Accelerating Apache Hive with MPI for Data Warehouse Systems. ICDCS 2015, Columbus, Ohio, USA, 2015
  – Xiaoyi Lu, Bing Wang, Li Zha, Zhiwei Xu: Can MPI Benefit Hadoop and MapReduce Applications? ICPP Workshops 2011: 371-379

• Energy Efficient Ternary Computing

• http://novel.ict.ac.cn/zxu/#PAPERS
谢谢！
Thank you!

zxu@ict.ac.cn    http://novel.ict.ac.cn/zxu/