Benchmarking Harp-DAAL: High Performance Hadoop on KNL Clusters

IEEE Cloud Computing Conference
June 27, 2017

Judy Qiu

Intelligent Systems Engineering Department, Indiana University
Email: xqiu@indiana.edu
1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work

Indiana University (Fox, Qiu, Crandall, von Laszewski), Rutgers (Jha), Virginia Tech (Marathe), Kansas (Paden), Stony Brook (Wang), Arizona State (Beckstein), Utah (Cheatham)
Motivation for faster and bigger problems

• **Machine Learning (ML)** Needs high performance
  – Big data and Big model
  – **Iterative algorithms** are fundamental in learning a non-trivial model
  – **Model training** and **Hyper-parameter tuning** steps run the iterative algorithms many times

• Architecture for Big Data analytics
  – to understand the algorithms through a **Model-Centric** view
  – to focus on the **computation and communication patterns** for optimizations
  – Trade-offs of efficiency and productivity
    • linear speedup with an increasing number of processors
    • easier to be parallelized on multicore or manycore computers
High Performance – Apache Big Data Stack

MapReduce

Data Centered, QoS

Classic Parallel Runtimes (MPI)

Efficient and Proven techniques

Expand the Applicability of MapReduce to more classes of Applications

Sequential

Map-Only

MapReduce

Iterative MapReduce

MPI and Point-to-Point

Input

map

Output

map

Output

map

reduce

map

reduce

iterations

Pij
Harp is an open-source project developed at Indiana University [6], it has:

- MPI-like collective communication operations that are highly optimized for big data problems.
- Harp has efficient and innovative computation models for different machine learning problems.

DAAL is an open-source project that provides:

- **Algorithms Kernels to Users**
  - Batch Mode (Single Node)
  - **Distributed Mode (multi nodes)**
  - Streaming Mode (single node)

- **Data Management & APIs to Developers**
  - Data structure, e.g., Table, Map, etc.
  - HPC Kernels and Tools: MKL, TBB, etc.
  - Hardware Support: Compiler
Harp-DAAL enable faster Machine Learning Algorithms with Hadoop Clusters on Multi-core and Many-core architectures

- Bridge the gap between HPC hardware and Big data/Machine learning Software
- Support Iterative Computation, Collective Communication, Intel DAAL and native kernels
- Portable to new many-core architectures like Xeon Phi and run on Haswell and KNL clusters
Outline

1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work
General Reduction in Hadoop, Spark, Flink

Comparison of Reductions:
- Separate Map and Reduce Tasks
- Switching tasks is expensive
- MPI only has one sets of tasks for map and reduce
- MPI achieves AllReduce by interleaving multiple binary trees
- MPI gets efficiency by using shared memory intra-node (e.g. multi-/manycore, GPU)

Map Tasks

Reduce Tasks

Follow by Broadcast for AllReduce which is a common approach to support iterative algorithms

For example, paper [7] 10 learning algorithms can be written in a certain “summation form,” which allows them to be easily parallelized on multicore computers.

HPC Runtime versus ABDS distributed Computing Model on Data Analytics

Hadoop writes to disk and is slowest; Spark and Flink spawn many processes and do not support allreduce directly; MPI does in-place combined reduce/broadcast
Illustration of In-Place AllReduce in MPI
Why Collective Communications for Big Data Processing?

• Collective Communication and Data Abstractions
  o Optimization of global model synchronization
    o ML algorithms: convergence vs. consistency
    o Model updates can be out of order
  o Hierarchical data abstractions and operations
• Map-Collective Programming Model
  o Extended from MapReduce model to support collective communications
  o BSP parallelism at Inter-node vs. Intra-node levels
• Harp Implementation
  o A plug-in to Hadoop
### Harp APIs

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Collective</th>
<th>Event Driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>• DynamicScheduler</td>
<td>• MPI collective communication</td>
<td>• getEvent</td>
</tr>
<tr>
<td>• StaticScheduler</td>
<td>• broadcast</td>
<td>• waitEvent</td>
</tr>
<tr>
<td></td>
<td>• reduce</td>
<td>• sendEvent</td>
</tr>
<tr>
<td></td>
<td>• allgather</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• allreduce</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>MapReduce</strong> “shuffle-reduce”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• regroup with combine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Graph &amp; ML operations</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• “push” &amp; “pull” model parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• rotate global model parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>between workers</td>
<td></td>
</tr>
</tbody>
</table>
## Collective Communication Operations

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Algorithm</th>
<th>Time Complexity&lt;sup&gt;a&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>broadcast</td>
<td>chain</td>
<td>$n^\beta$</td>
</tr>
<tr>
<td></td>
<td>minimum spanning tree</td>
<td>$(\log_2 p)n^\beta$</td>
</tr>
<tr>
<td>reduce</td>
<td>minimum spanning tree</td>
<td>$(\log_2 p)n^\beta$</td>
</tr>
<tr>
<td>allgather</td>
<td>bucket</td>
<td>$pn^\beta$</td>
</tr>
<tr>
<td>allreduce</td>
<td>bi-directional exchange</td>
<td>$(\log_2 p)n^\beta$</td>
</tr>
<tr>
<td>regroup</td>
<td>point-to-point</td>
<td>$n^\beta$</td>
</tr>
<tr>
<td>push &amp; pull</td>
<td>point-to-point plus routing optimization</td>
<td>$n^\beta$</td>
</tr>
<tr>
<td>rotate</td>
<td>exchange data between neighbors on a ring topology</td>
<td>$n^\beta$</td>
</tr>
</tbody>
</table>

<sup>a</sup>Note in “time complexity”, $p$ is the number of processes, $n$ is the number of input data items per worker, $\beta$ is the per data item transmission time, communication startup time is neglected and the time complexity of the “point-to-point” based algorithms are estimated regardless of potential network conflicts.
Taxonomy for Machine Learning Algorithms

Optimization and related issues

- Task level only can't capture the traits of computation
- Model is the key for iterative algorithms. The structure (e.g. vectors, matrix, tree, matrices) and size are critical for performance
- Solver has specific computation and communication pattern

We investigate different computation and communication patterns of important ml algorithms
Parallel Machine Learning Application Implementation Guidelines

Application
• Latent Dirichlet Allocation, Matrix Factorization, Linear Regression…

Algorithm
• Expectation-Maximization, Gradient Optimization, Markov Chain Monte Carlo…

Computation Model
• Locking, Rotation, Allreduce, Asynchronous

System Optimization
• Collective Communication Operations
• Load Balancing on Computation and Communication
• Per-Thread Implementation
Computation Models

(A) Model
   - Synchronized algorithm
   - The latest model

(B) Model 1 → Model 2 → Model 3
   - Synchronized algorithm
   - The latest model

(C) Model
   - Synchronized algorithm
   - Stale model

(D) Model
   - Asynchronous algorithm
   - Stale model

Harp Solution to Big Data Problems

**Computation Models**

*Model-Centric Synchronization Paradigm*

- Computation Models
  - Locking
  - Rotation
  - Allreduce

**Distributed Memory**

- broadcast
- regroup
- sendEvent
- reduce
- push & pull
- getEvent
- allgather
- rotate
- allreduce
- waitEvent

**Communication Operations**

**Shared Memory**

- Schedulers
  - Dynamic Scheduler
  - Static Scheduler

Harp-DAAL High Performance Library
Example: K-means Clustering

The Allreduce Computation Model

Worker → Model → Worker

When the model size is small

When the model size is large but can still be held in each machine’s memory

When the model size cannot be held in each machine’s memory

broadcast
reduce
allreduce
regroup
allgather
push & pull
rotate
Outline

1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work
Hardware specifications

Table I  Specification of Xeon Phi 7250 KNL

<table>
<thead>
<tr>
<th>Cores</th>
<th>Memory</th>
<th>Node Spec</th>
<th>Misc Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>DDR4</td>
<td>Network</td>
<td>Omni-path</td>
</tr>
<tr>
<td>Base Freq</td>
<td>MCDRAM</td>
<td>Peak Port Band</td>
<td>100 Gbps</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>DDR4-Band</td>
<td>Socket</td>
<td>1</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>MCDRAM-Band</td>
<td>Disk</td>
<td>1 TB</td>
</tr>
</tbody>
</table>

Table II  Specification of Haswell Xeon E5 2699 v3

<table>
<thead>
<tr>
<th>Cores</th>
<th>Memory</th>
<th>Node Spec</th>
<th>Misc Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>DDR4</td>
<td>Network</td>
<td>InfiniBand</td>
</tr>
<tr>
<td>Base Freq</td>
<td>HBM</td>
<td>Peak Port Band</td>
<td>56 Gbps</td>
</tr>
<tr>
<td>L1/L2 Cache</td>
<td>DDR4-Band</td>
<td>Socket</td>
<td>2</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>HBM-Band</td>
<td>Disk</td>
<td>8 TB</td>
</tr>
</tbody>
</table>

All scalability tests run on the above Haswell (128 node) and KNL (64 node) clusters.
**Harp-SAHad SubGraph Mining**

VT and IU collaborative work

Relational sub-graph isomorphism problem: find sub-graphs in G which are isomorphic to the given template T. **SAHAD** is a challenging graph application that is both data intensive and communication intensive. **Harp-SAHad** is an implementation for sub-graph counting problem based on SAHAD algorithm and Harp framework.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Web-google</td>
<td>0.9</td>
<td>4.3</td>
<td>65</td>
</tr>
<tr>
<td>Miami</td>
<td>2.1</td>
<td>51.2</td>
<td>740</td>
</tr>
<tr>
<td>Nyc</td>
<td>18</td>
<td>480</td>
<td>7856</td>
</tr>
</tbody>
</table>

**Table IV** Networks of Graph Applications

Harp-SAHad Performance Results

VT and IU collaborative work

Figure Speedup on nyc dataset

Figure Running time break-down on u3-1 template.
# Test Plan and Datasets

## Table III  
DATASETS USED IN K-MEANS, MF-SGD, AND ALS

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kmeans-Single</th>
<th>Kmeans-Multi</th>
<th>Movielens</th>
<th>Netflix</th>
<th>Yahoomusic</th>
<th>Enwiki</th>
<th>Hugewiki</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Training</td>
<td>50000000</td>
<td>20000000</td>
<td>9301274</td>
<td>99072112</td>
<td>252800275</td>
<td>609700674</td>
<td>3074875354</td>
</tr>
<tr>
<td>#Test</td>
<td>none</td>
<td>none</td>
<td>698780</td>
<td>1408395</td>
<td>4003960</td>
<td>12437156</td>
<td>365998592</td>
</tr>
<tr>
<td>#centroid</td>
<td>10000</td>
<td>10000</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Dim</td>
<td>100</td>
<td>100</td>
<td>40</td>
<td>40</td>
<td>100</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>λ</td>
<td>none</td>
<td>none</td>
<td>0.05</td>
<td>0.05</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>γ</td>
<td>none</td>
<td>none</td>
<td>0.003</td>
<td>0.002</td>
<td>0.0001</td>
<td>0.001</td>
<td>0.004</td>
</tr>
</tbody>
</table>
Harp-DAAL Applications

- Clustering
- Vectorized computation
- **Small model data**
- Regular Memory Access

- Matrix Factorization
- **Huge model data**
- Random Memory Access
- Easy to scale up
- Hard to parallelize

- Matrix Factorization
- **Huge model data**
- Regular Memory Access
- Easy to parallelize
- Hard to scale up

Computation models for K-means

Harp-DAAL-Kmeans

• Inter-node: Allreduce, Easy to implement, efficient when model data is not large

1: procedure
2: Given \((x^1, x^2, \ldots, x^m)\), \(\forall i, x^i \in \mathbb{R}^n\)
3: Initialize centroids randomly: \(\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n\)
4: Repeat until convergence
5: \(\forall i, c^i := \arg\min_j \|x^i - \mu_j\|^2\)
6: \(\forall j, \mu_j := \frac{\sum_{i=1}^{m} 1\{c^i = j\} x^i}{\sum_{i=1}^{m} 1\{c^i = j\}}\)
7: End Repeat
8: end procedure

• Intra-node: Shared Memory, matrix-matrix operations, xGemm: aggregate vector-vector distance computation into matrix-matrix multiplication, higher computation intensity (BLAS-3)

\[
C \leftarrow \alpha \text{op}(A) \text{op}(B) + \beta C
\]
Computation models for MF-SGD

- Inter-node: Rotation
- Intra-node: Asynchronous

Rotation: Efficient when the mode data is large, good scalability

Asynchronous: Random access to model data
Good for thread-level workload balance.

procedure

\[
R \in \mathbb{R}^{m \times n}, P \in \mathbb{R}^{k \times m}, \text{ and } Q \in \mathbb{R}^{k \times n}
\]

while true do

select randomly a point \( r_{ij} \) from \( R \)

\[
e_{ij} = r_{ij} - p_i^T q_j
\]

\[
p_i \leftarrow p_i + \gamma (e_{ij} q_j - \lambda p p_i)
\]

\[
q_j \leftarrow q_j + \gamma (e_{ij} p_i - \lambda Q q_j)
\]

if \( P, Q \) converged then

Exit While loop

end if

end while

end procedure
Computation Models for ALS

- Inter-node: Allreduce

```
procedure
Load R, R^T
Initialize X, Y
repeat
  for i = 1, 2, \ldots, n do
    V_i = Y_i R^T(i, I_i)
    A_i = Y_i Y_i^T + \lambda n_{x_i} E
    x_i = A_i^{-1} V_i
  end for
  for j = 1, 2, \ldots, m do
    U_j = X_{I_j} R(I_j, j)
    B_j = X_{I_j} X_{I_j}^T + \lambda n_{m_j} E
    y_j = B_j^{-1} U_j
  end for
until convergence
end procedure
```

- Intra-node: Shared Memory, Matrix operations

xSyrk: symmetric rank-k update

\[
C \leftarrow \alpha A A^T + \beta C
\]
\[
A \leftarrow \alpha x x^T + A
\]
Performance on KNL Single Node

Harp-DAAL-Kmeans vs. Spark-Kmeans:

~ 20x speedup
1) Harp-DAAL-Kmeans invokes MKL matrix operation kernels at low level
2) Matrix data stored in contiguous memory space, leading to regular access pattern and data locality

Harp-DAAL-SGD vs. NOMAD-SGD

1) Small dataset (MovieLens, Netflix): comparable perf
2) Large dataset (Yahoomusic, Enwiki): 1.1x to 2.5x, depending on data distribution of matrices

Harp-DAAL-ALS vs. Spark-ALS

20x to 50x speedup
1) Harp-DAAL-ALS invokes MKL at low level
2) Regular memory access, data locality in matrix operations

Harp-DAAL has much better single node performance than Java solution (Spark-Kmeans, Spark-ALS) and comparable performance to state-of-arts C++ solution (NOMAD-SGD)
Performance on KNL Multi-Nodes

**Harp-DAAL-Kmeans:**
- **15x to 20x speedup** over Spark-Kmeans
  1) Fast single node performance
  2) Near-linear strong scalability from 10 to 20 nodes
  3) After 20 nodes, insufficient computation workload leads to some loss of scalability

**Harp-DAAL-SGD:**
- **2x to 2.5x speedup** over NOMAD-SGD
  1) Comparable or fast single node performance
  2) Collective communication operations in Harp-DAAL outperform point-to-point MPI communication in NOMAD

**Harp-DAAL-ALS:**
- **25x to 40x speedup** over Spark-ALS
  1) Fast single node performance
  2) ALS algorithm is not scalable (high communication ratio)

*Harp-DAAL combines the benefits from local computation (DAAL kernels) and communication operations (Harp), which is much better than Spark solution and comparable to MPI solution.*
**Breakdown of Intra-node Performance**

**Thread scalability:**
- Harp-DAAL best threads number: 64 (K-means, ALS) and 128 (MF-SGD), more than 128 threads no performance gain
  - communications between cores intensify
  - cache capacity per thread also drops significantly
- Spark best threads number 256, because Spark could not fully Utilize AVX-512 VPU
- NOMAD-SGD could use AVX VPU, thus has 64 its best thread as that of Harp-DAAL-SGD
Spark-Kmeans and Spark-ALS dominated by Computation (retiring), no AVX-512 to reduce retiring Instructions, Harp-DAAL improves L1 cache bandwidth utilization due to AVX-512
Code Optimization Highlights

Two ways to store data using DAAL Java API

- Keep Data on JVM heap
  - no contiguous memory access requirement
  - Small size DirectByteBuffer and parallel copy (OpenMP)

- Keep Data on Native Memory
  - contiguous memory access requirement
  - Large size DirectByteBuffer and bulk copy

Data Conversion

- Harp Data
- DAAL Java API
- DAAL Native Kernel

- Table<Obj>
- NumericTable
- MicroTable

- Data on JVM Heap
- Data on JVM heap
- Data on Native Memory

A single DirectByteBuffer has a size limite of 2 GB
Data Structures of Harp & Intel’s DAAL

Table<Obj> in Harp has a three-level data Hierarchy

- Table: consists of partitions
- Partition: partition id, container
- Data container: wrap up Java objs, primitive arrays

Data in different partitions, non-contiguous in memory

NumericTable in DAAL stores data either in Contiguous memory space (native side) or non-contiguous arrays (Java heap side)

Data in contiguous memory space favors matrix operations with regular memory accesses.

DAAL Table has different types of Data storage
Two Types of Data Conversion

**JavaBulkCopy:**
- Dataflow: Harp Table<Obj> ----
- Java primitive array ---- DirectByteBuffer ----
- NumericTable (DAAL)
- Pros: Simplicity in implementation
- Cons: high demand of DirectByteBuffer size

**NativeDiscreteCopy:**
- Dataflow: Harp Table<Obj> ----
- DAAL Java API (SOANumericTable)
- ---- DirectByteBuffer ---- DAAL native memory
- Pros: Efficiency in parallel data copy
- Cons: Hard to implement at low-level kernels
Outline

1. Introduction: HPC-ABDS, Harp (Hadoop plug in), DAAL
2. Optimization Methodologies
3. Results (configuration, benchmark)
4. Code Optimization Highlights
5. Conclusions and Future Work
Conclusions

• Identification of **Apache Big Data Software Stack** and integration with **High Performance Computing Stack** to give **HPC-ABDS**
  - ABDS (Many Big Data applications/algorithms need HPC for performance)
  - HPC (needs software model productivity/sustainability)

• Identification of **4 computation models** for machine learning applications
  - Locking, Rotation, Allreduce, Asynchronous

• **HPC-ABDS**: High performance **Hadoop** (with Harp-DAAL) on KNL and Haswell clusters
**Hadoop/Harp-DAAL: Prototype and Production Code**

Open Source Available at https://dsc-spidal.github.io/harp

- Source codes became available on Github in February, 2017.
- Harp-DAAL follows the same standard of DAAL’s original codes
- Six Applications
  - Harp-DAAL Kmeans
  - Harp-DAAL MF-SGD
  - Harp-DAAL MF-ALS
  - Harp-DAAL SVD
  - Harp-DAAL PCA
  - Harp-DAAL Neural Networks
## Scalable Algorithms implemented using Harp

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Category</th>
<th>Applications</th>
<th>Features</th>
<th>Computation Model</th>
<th>Collective Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-means</td>
<td>Clustering</td>
<td>Most scientific domain</td>
<td>Vectors</td>
<td>AllReduce</td>
<td>allreduce, regroup+allgather, broadcast+reduce, push+pull</td>
</tr>
<tr>
<td>Multi-class Logistic Regression</td>
<td>Classification</td>
<td>Most scientific domain</td>
<td>Vectors, words</td>
<td>Rotation</td>
<td>regroup, rotate, allgather</td>
</tr>
<tr>
<td>Random Forests</td>
<td>Classification</td>
<td>Most scientific domain</td>
<td>Vectors</td>
<td>AllReduce</td>
<td>allreduce</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>Classification, Regression</td>
<td>Most scientific domain</td>
<td>Vectors</td>
<td>AllReduce</td>
<td>allgather</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>Classification</td>
<td>Image processing, voice recognition</td>
<td>Vectors</td>
<td>AllReduce</td>
<td>allreduce</td>
</tr>
<tr>
<td>Latent Dirichlet Allocation</td>
<td>Structure learning (Latent topic model)</td>
<td>Text mining, Bioinformatics, Image Processing</td>
<td>Sparse vectors; Bag of words</td>
<td>Rotation</td>
<td>rotate, allreduce</td>
</tr>
<tr>
<td>Matrix Factorization</td>
<td>Structure learning (Matrix completion)</td>
<td>Recommender system</td>
<td>Irregular sparse Matrix; Dense model vectors</td>
<td>Rotation</td>
<td>rotate</td>
</tr>
<tr>
<td>Multi-Dimensional Scaling</td>
<td>Dimension reduction</td>
<td>Visualization and nonlinear identification of principal components</td>
<td>Vectors</td>
<td>AllReduce</td>
<td>allgather, allreduce</td>
</tr>
<tr>
<td>Subgraph Mining</td>
<td>Graph</td>
<td>Social network analysis, data mining, fraud detection, chemical informatics, bioinformatics</td>
<td>Graph, subgraph</td>
<td>Rotation</td>
<td>rotate</td>
</tr>
<tr>
<td>Force-Directed Graph Drawing</td>
<td>Graph</td>
<td>Social media community detection and visualization</td>
<td>Graph</td>
<td>AllReduce</td>
<td>allgather, allreduce</td>
</tr>
</tbody>
</table>
Future Work

• **Harp-DAAL** machine learning and data analysis applications with optimal performance.

• **Online Clustering** with **Harp** or **Storm** integrates parallel and dataflow computing models

• Start HPC Cloud incubator project in Apache to bring HPC-ABDS to community
Six Computation Paradigms for Data Analytics

(1) Map Only
- Pleasingly Parallel
  - BLAST Analysis
  - Local Machine Learning
  - Pleasingly Parallel

(2) Classic Map-Reduce
- High Energy Physics (HEP) Histograms, Web search, Recommender Engines

(3) Iterative Map Reduce or Map-Collective
- Expectation Maximization
  - Clustering
  - Linear Algebra
  - PageRank

(4) Point to Point or Map-Communication
- Classic MPI
  - PDE Solvers and Particle Dynamics
  - Graph

(5) Map-Streaming
- Streaming images from Synchrotron sources, Telescopes, Internet of Things

(6) Shared memory Map-Communication
- Difficult to parallelize
  - asynchronous parallel Graph

These 3 Paradigms are our Focus
We gratefully acknowledge support from NSF, IU and Intel Parallel Computing Center (IPCC) Grant.

Langshi Cheng  Bingjing Zhang  Bo Peng  Kannan Govindarajan
Supun Kamburugamuve  Yiming Zhou  Ethan Li  Mihai Avram  Vibhatha Abeykoon

Intelligent Systems Engineering
School of Informatics and Computing
Indiana University