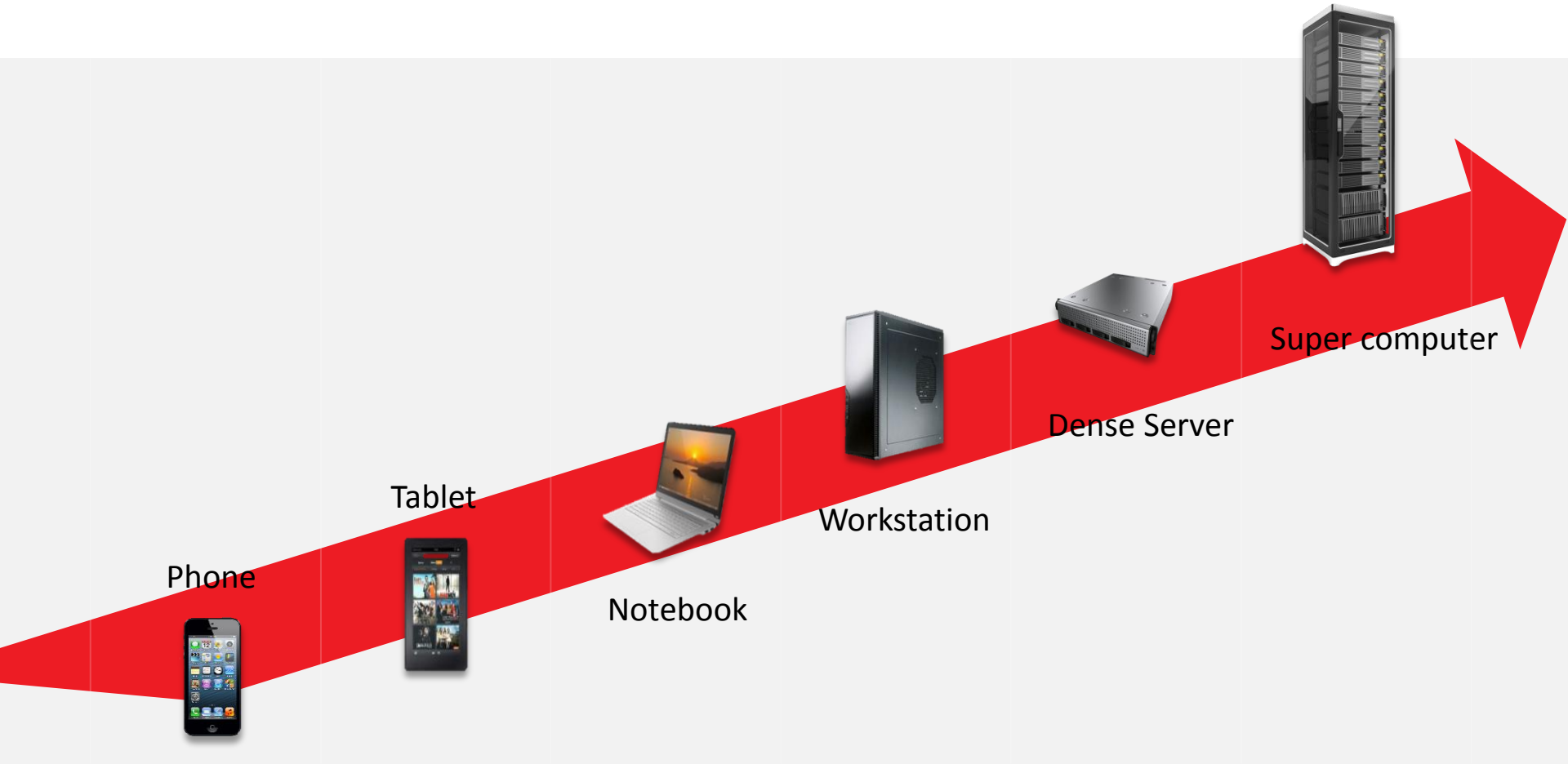
AMD

OPTIMIZING BIG DATA ANALYTICS ON HETEROGENEOUS PROCESSORS

MAYANK DAGA, MAURICIO BRETERNITZ, JUNLI GU
AMD RESEARCH

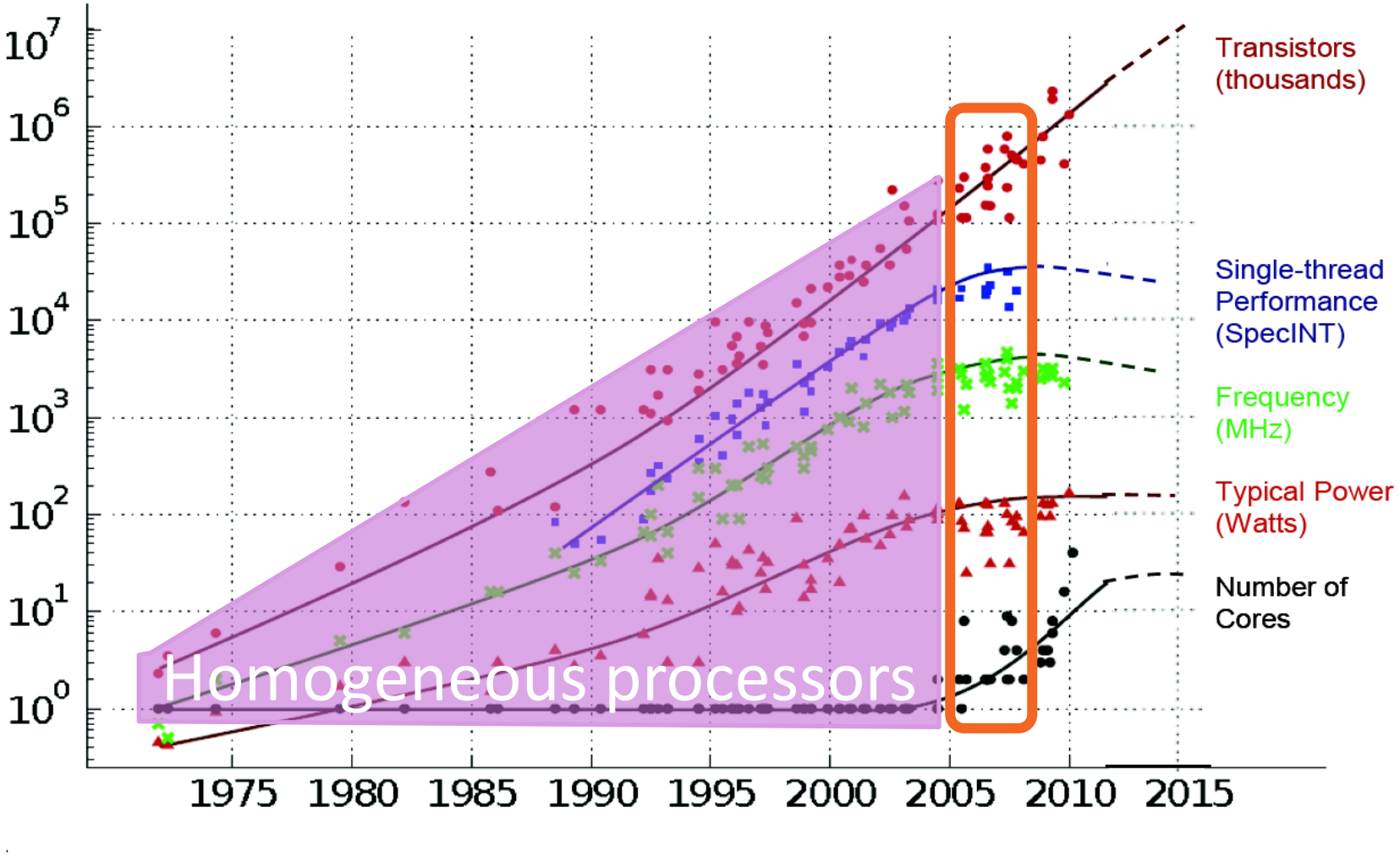
HETEROGENEOUS PROCESSORS - EVERYWHERE

SMARTPHONES TO SUPER-COMPUTERS



From Phil Rogers APU13 Keynote

35 YEARS OF MICROPROCESSOR TREND DATA



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

IMPORTANT SHIFTS

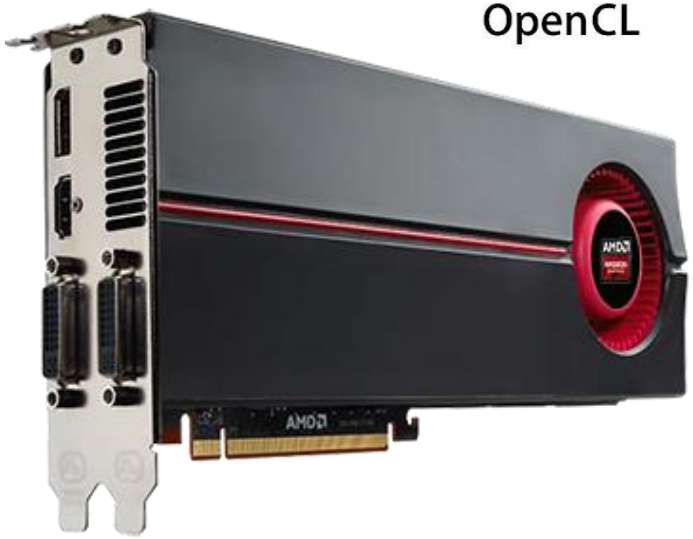
AMD

2004 - 2005

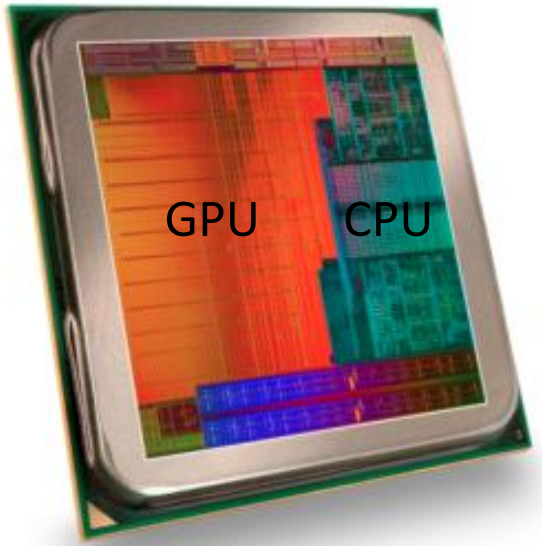
2007 - 2008

CUDA®

OpenCL



2010 - 2011



The Era of Heterogeneous Computing Is Here !

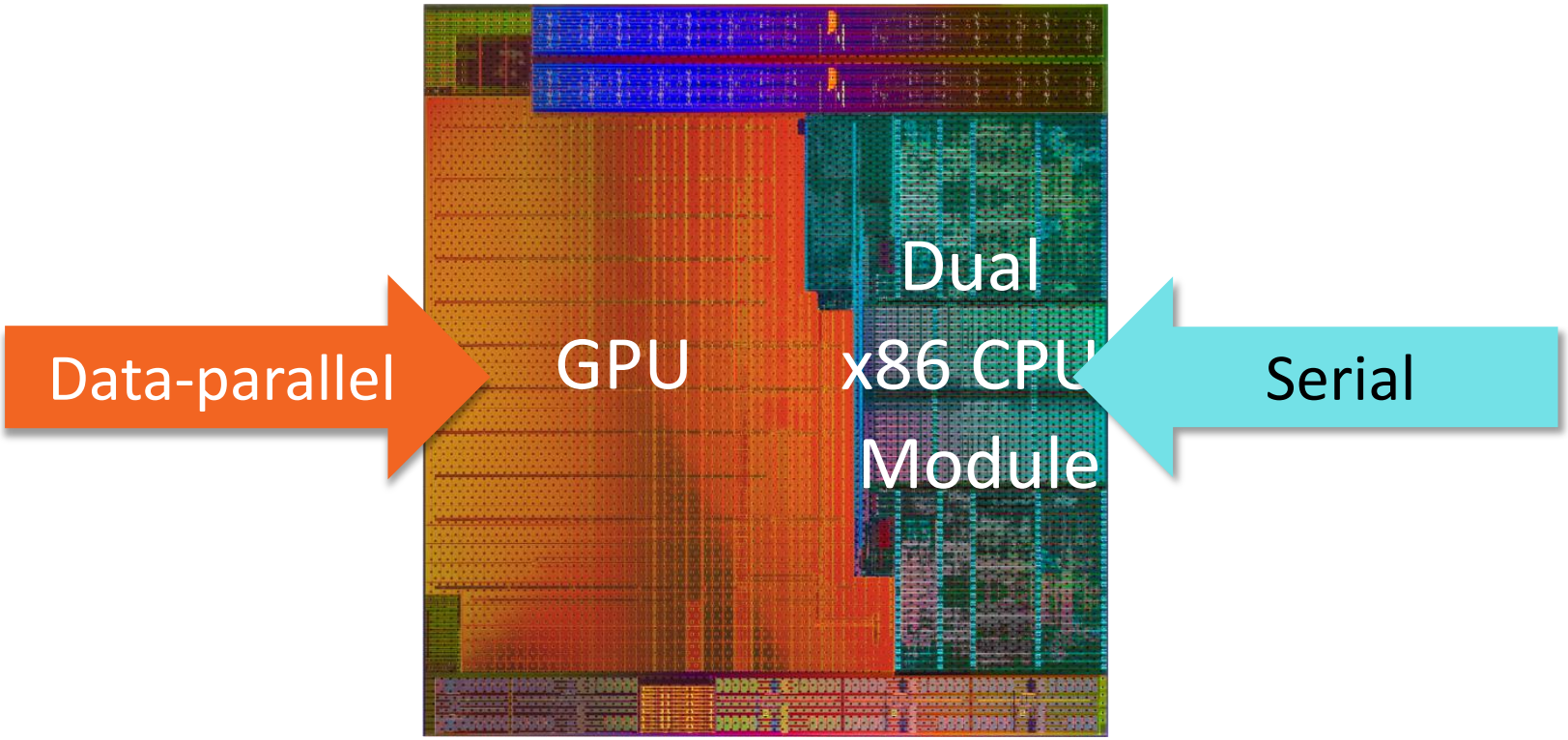
CENTRAL PROCESSING UNIT (CPU)

- Few BIG cores
- Ideal for scalar & control-intensive parts of application

GRAPHICS PROCESSING UNIT (GPU)

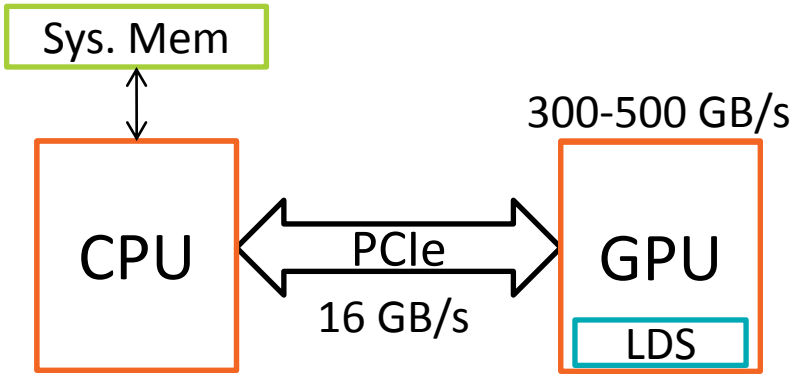
- Lots of small cores
- Resides over the PCIe bus
- Ideal for data-parallel parts of the application

ACCELERATED PROCESSING UNIT (APU)

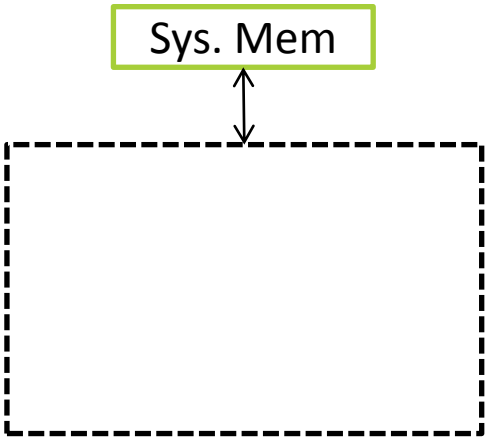


- A ***heterogeneous platform*** *with* CPU+GPU on the same silicon die
- Ideal for ***both serial and data-parallel*** parts of application
- TDP of less than 100 Watts

HOW IS AN APU DIFFERENT



- Manage data-movement across PCIe®
- Use local scratchpad memory (cache)



Accelerated Processing Unit (APU)

- No data-movement overhead
- Programming is similar to discrete GPU but *simpler*

MODERN WORKLOADS ARE HETEROGENEOUS

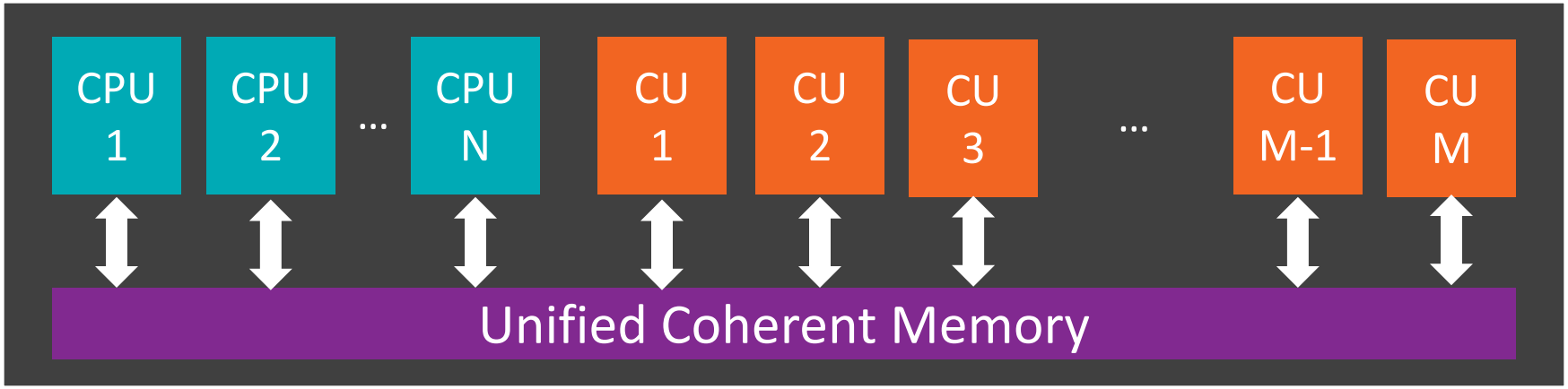
SCALAR CONTENT WITH A GROWING MIX OF PARALLEL CONTENT

- Video is expected to represent two thirds of mobile data traffic by 2017
  - Video processing is inherently parallel … and can be accelerated
- Big data growing exponentially with exabytes of data crawled monthly
  - Map reduce is a heterogeneous workload
- Rapid growth of Sensor Networks
  - Drives exponential increase in data
- Internet of Things (IoT) results in explosion of data sources
  - Another exponential growth in data at local and cloud level

HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

AMD

HARDWARE AND SOFTWARE INTEGRATION TO DELIVER HETEROGENEOUS COMPUTE TO THE MASSES



All processors use same memory addresses

Power efficient

Full access to virtual and physical memory

Easy to program

LARGE-SCALE DATA ANALYTICS & HETEROGENEOUS COMPUTE (HC)

AMD

Volume

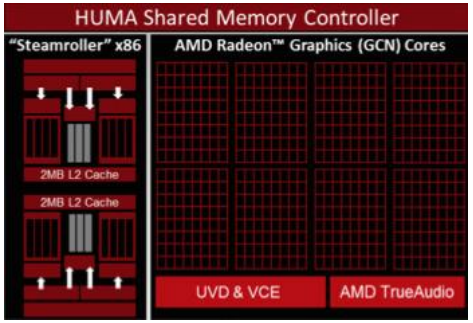
Large amounts of data stored in **cloud**

HC makes the cloud **energy-efficient**

Velocity

Needs to be analyzed **quickly**

47% GPU



GPU provides the computational horsepower

Variety

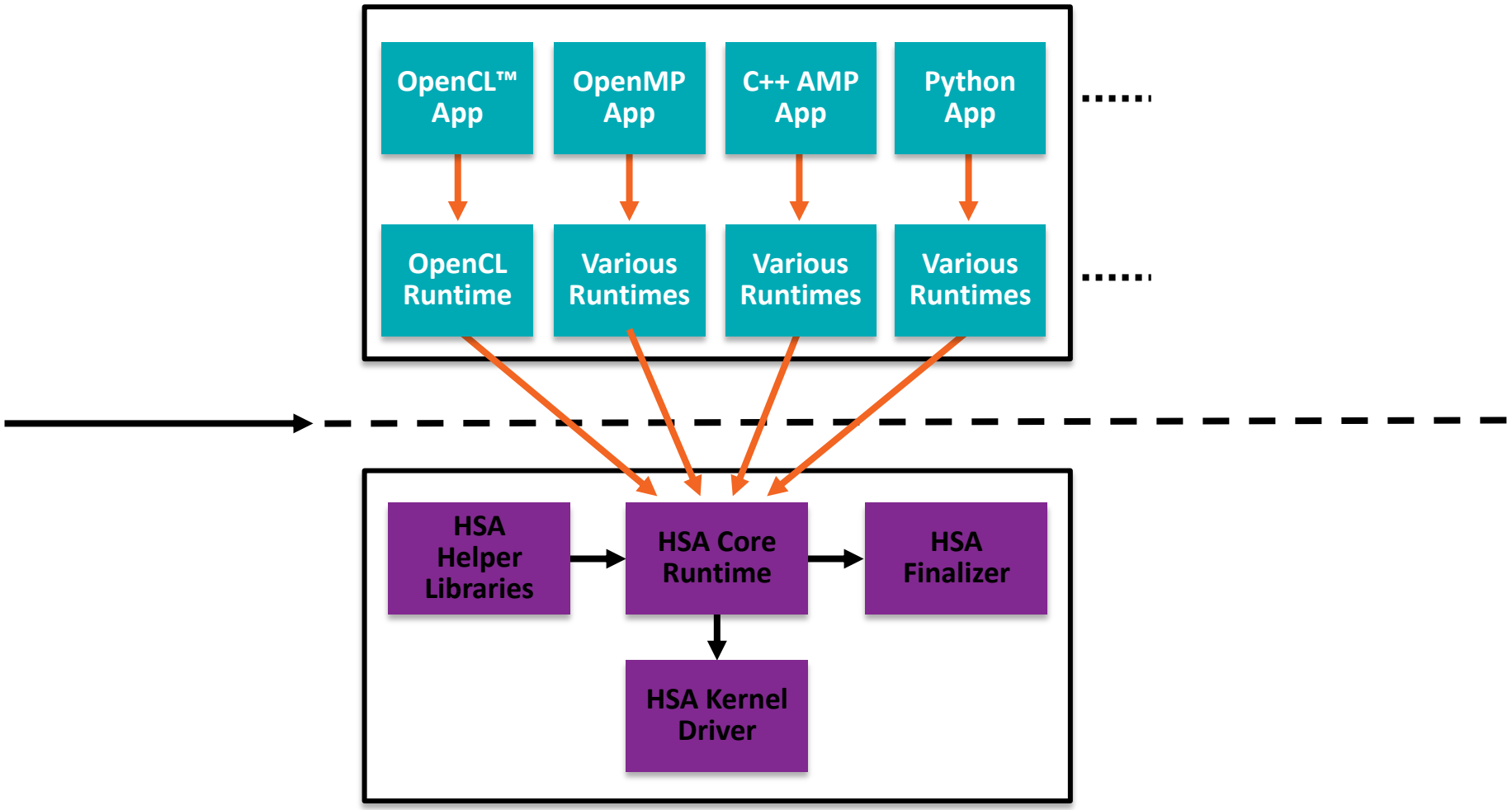
Different types of structured and unstructured data

A **heterogeneous platform** for the heterogeneous data

AMD

Programming Heterogeneous Processors

PROGRAMMING LANGUAGES PROLIFERATING ON APU



AMD

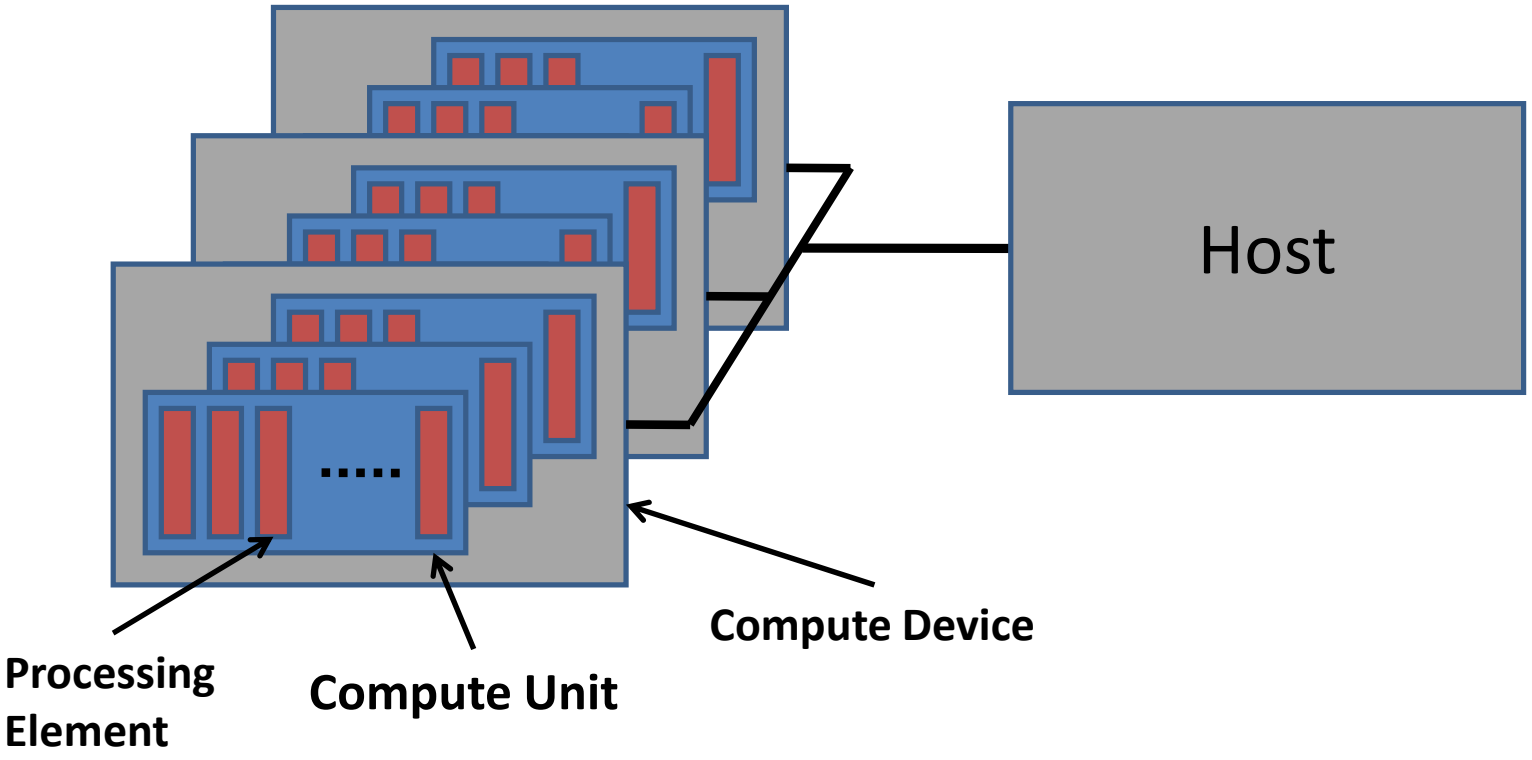
OpenCL *Performance*

C++ AMP *Productivity*

WHAT IS OPENCL

AMD

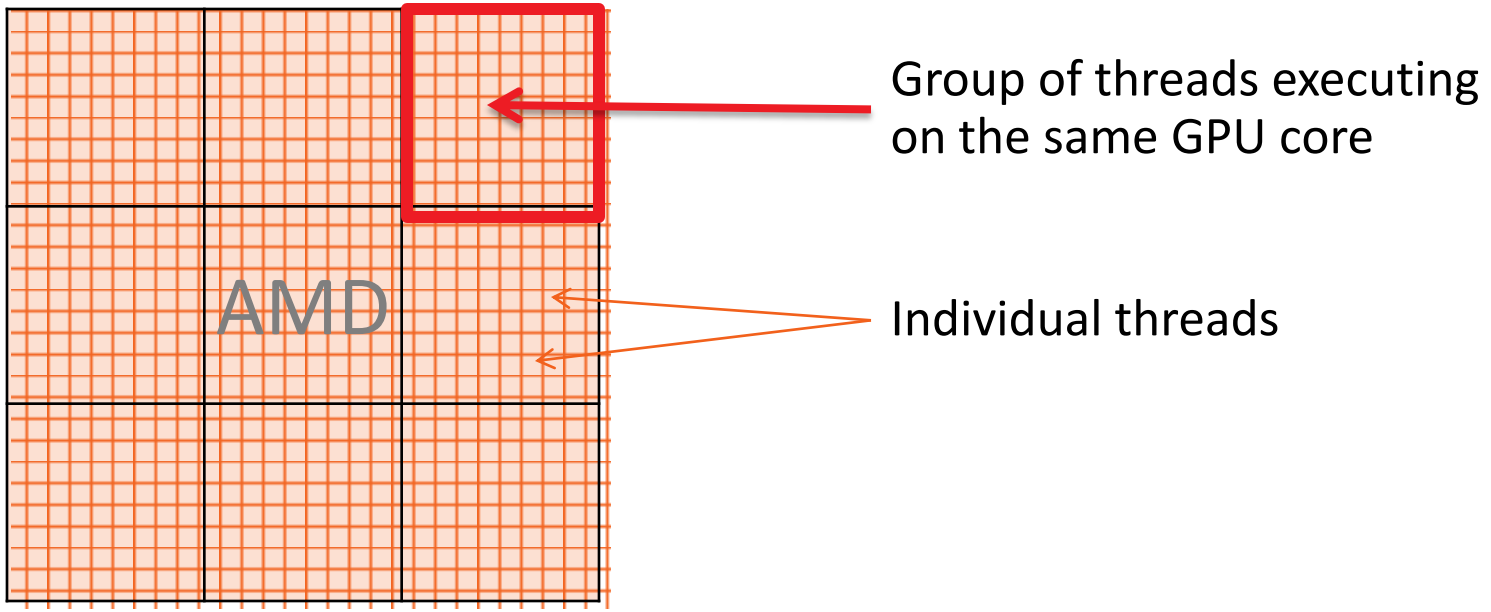
- **OPEN C**omputing **L**anguage
  - Open standard managed by the Khronos Group
- Platform Agnostic --- CPUs, GPUs, FPGAs, DSPs



OPENCL: EXECUTION MODEL

AMD

- Host Program
  - Executes on the host (*usually a CPU*)
  - Sends commands to the compute devices using a queue
- Kernel
  - Basic unit of executable code which runs on compute devices
  - A grid of parallel threads execute the kernel on the compute device



CODE EXAMPLE

```
for loop {
  // do work;
}
```

1. OpenCL Initialization
2. Allocate memory
3. Data_copy_GPU
4. Launch GPU Kernel
5. Data_copy_Host

Host-side code

```
kernel {
  // do work;
}
```

Device-side code

GETTING STARTED RESOURCES (OPENCL)

AMD

- AMD APP Programming SDK
  - http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/
- AMD APP Programming Guide
  - http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_OpenCL_Programming_User_Guide2.pdf
  - http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_OpenCL_Programming_Optimization_Guide2.pdf
- Works for both Windows and Linux

C++ AMP

AMD

- **C++**, not C
- **Mainstream**: programmed by millions
- **Minimal**: just one language extension
- **Portable**: mix and match hardware from any vendor
- **General and Future Proof**: designed to cover full-range of heterogeneity

CODE EXAMPLE

```
parallel_for_each( loop{ threads, [=] (t_idx) {
  // do work;
} );
```

```
for loop {
  // do work;
}
```

+ Combination of library and extensions to C++ standard
+ Single-source
+ Substantially boosts programmer productivity

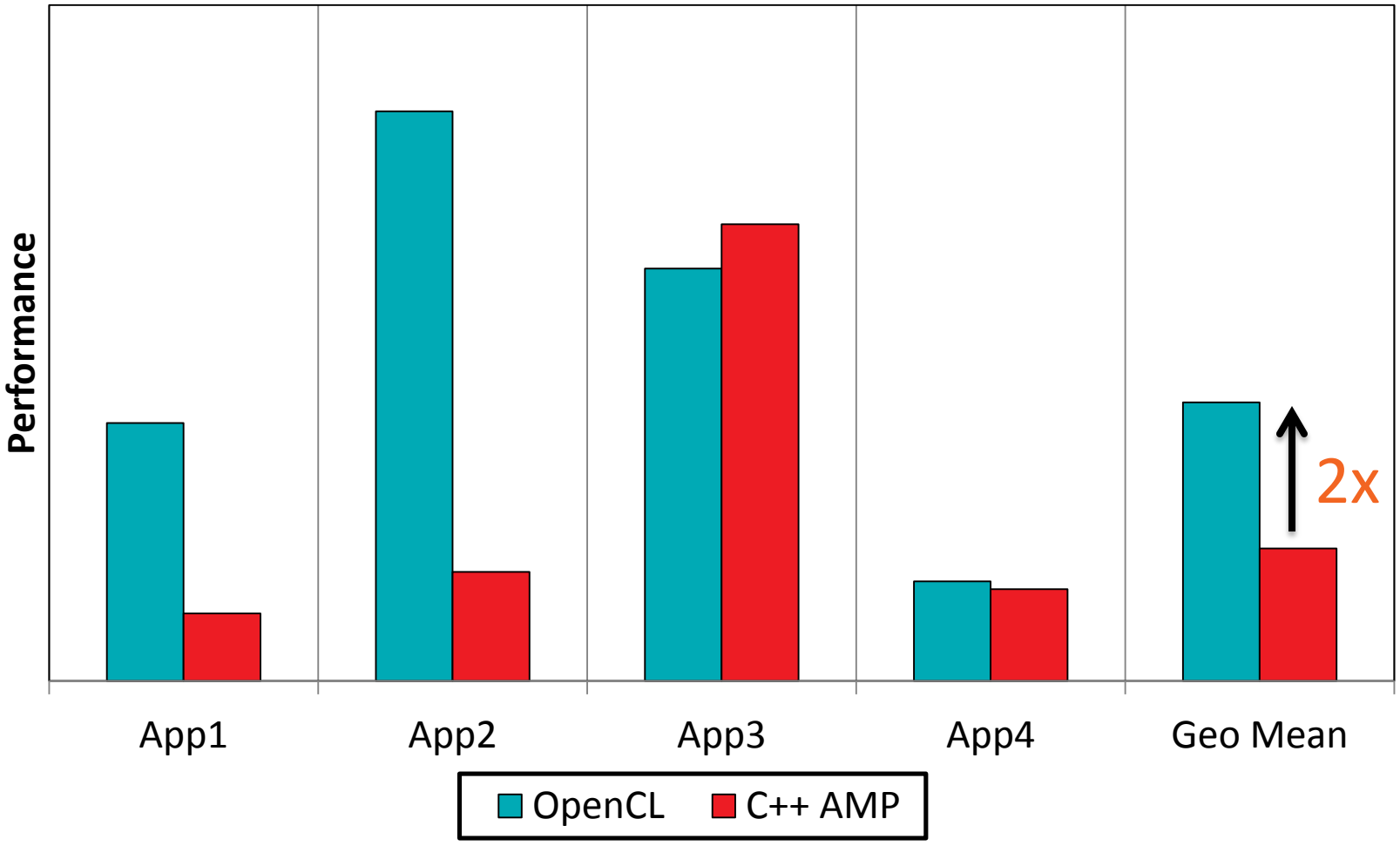
- No asynchronous data-transfers

GETTING STARTED RESOURCES (C++ AMP)

AMD

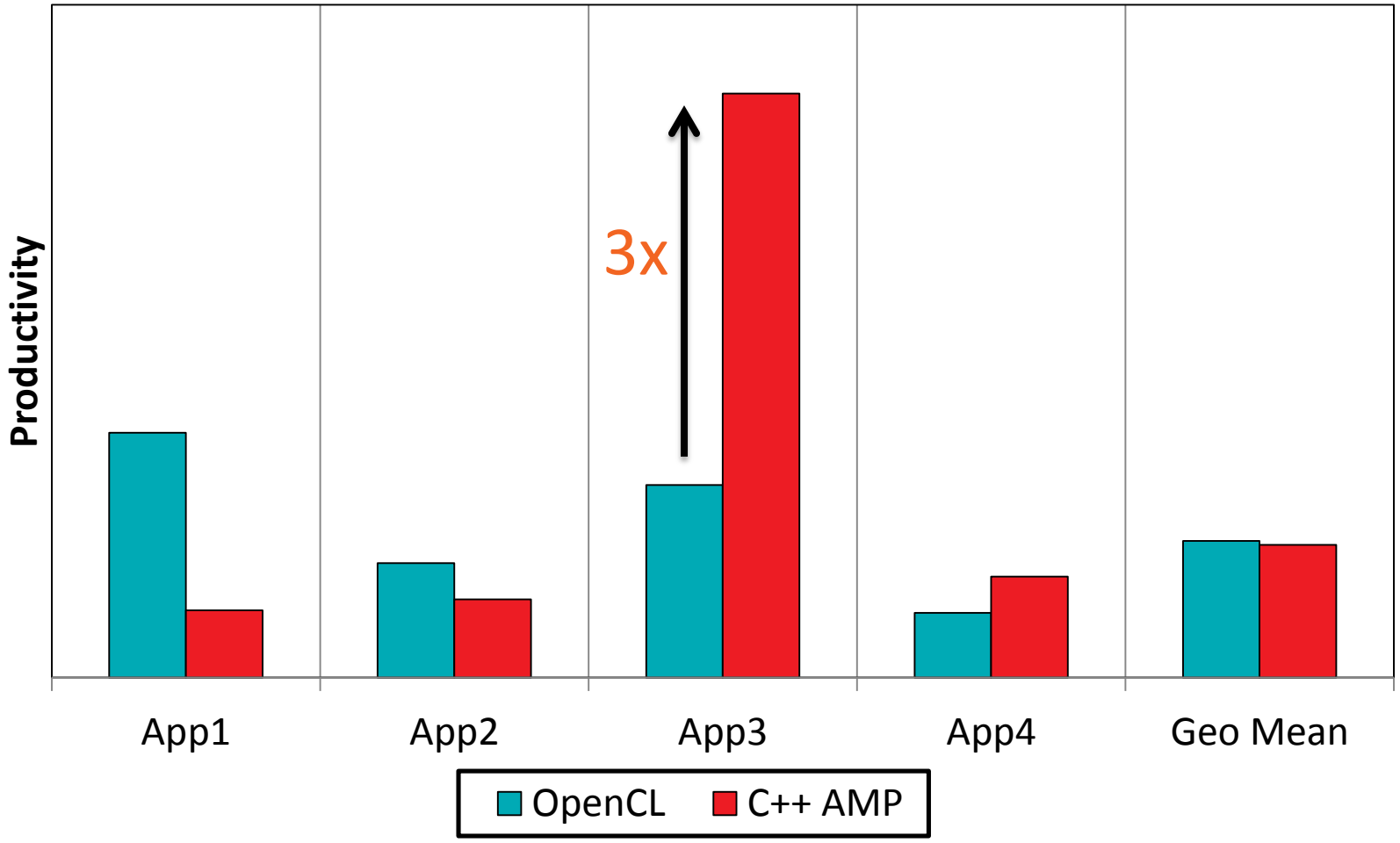
- Compiler and runtime
  - https://bitbucket.org/multicoreware/cppamp-driver-ng/wiki/Home
  - Microsoft Visual Studio
- Programming Guide
  - https://msdn.microsoft.com/en-us/library/hh265136.aspx
- Works for both Windows and Linux

PERFORMANCE



Daga et al. IISWC 2015

PRODUCTIVITY



Daga et al. IISWC 2015

SUPPORT FOR POPULAR LIBRARIES

AMD

- Computer Vision
  - OpenCV
- Data Science
  - SciPy
  - NumPy
- Image Processing
  - ImageMagick
- Parallel Standard Template Library
  - Bolt
- Linear Algebra Library
  - AMD clBLAS

OUR THREE-PRONGED APPROACH TO DATA ANALYTICS

AMD

- Enhancing Programming Model
  - **HadoopCL and Apache Spark:** flexibility, reliability, and programmability of Hadoop accelerated by OpenCL
- Enhancing Data Operations
  - **Deep Neural Networks:** achieved 2x energy efficiency on the APU than discrete GPUs
  - **Breadth-first Search:** fastest single-GPU Graph500 implementation (June 2014)
  - **SpMV:** state-of-the-art CSR-based SpMV (13x faster than prior CSR-SpMV and 2x faster than other storage formats
- Enhancing Data Organization
  - **In-Memory B+ Trees:** efficient memory reorganization to achieve 3x speedup on the APU over a multicore implementation

NESTED PROCESSING OF MACHINE LEARNING MAP-REDUCE, BIG DATA ON APUS

MAURICIO BRETERNITZ, PH.D.
AMD RESEARCH
TECHNOLOGY & ENGINEERING GROUP
AMD

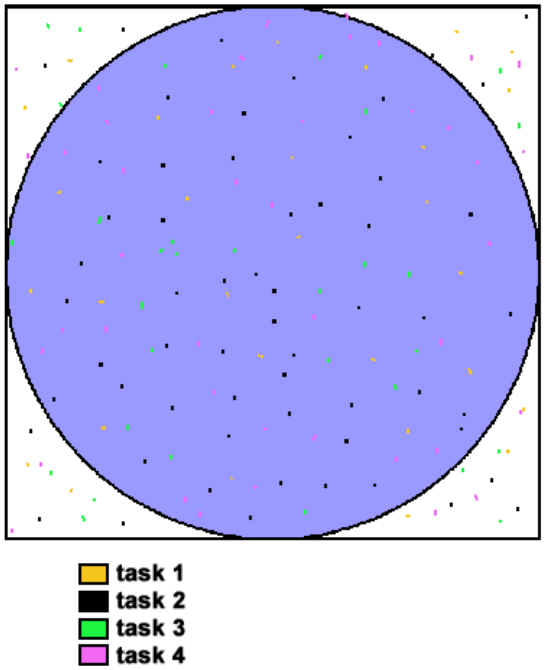
thanks: Max Grossman, Vivek Sarkar

MapReduce in 30 seconds: Estimating PI

AMD

1- pick random points in unit rectangle

2- count fraction inside circle



Area: π/ 4

Map-Reduce:

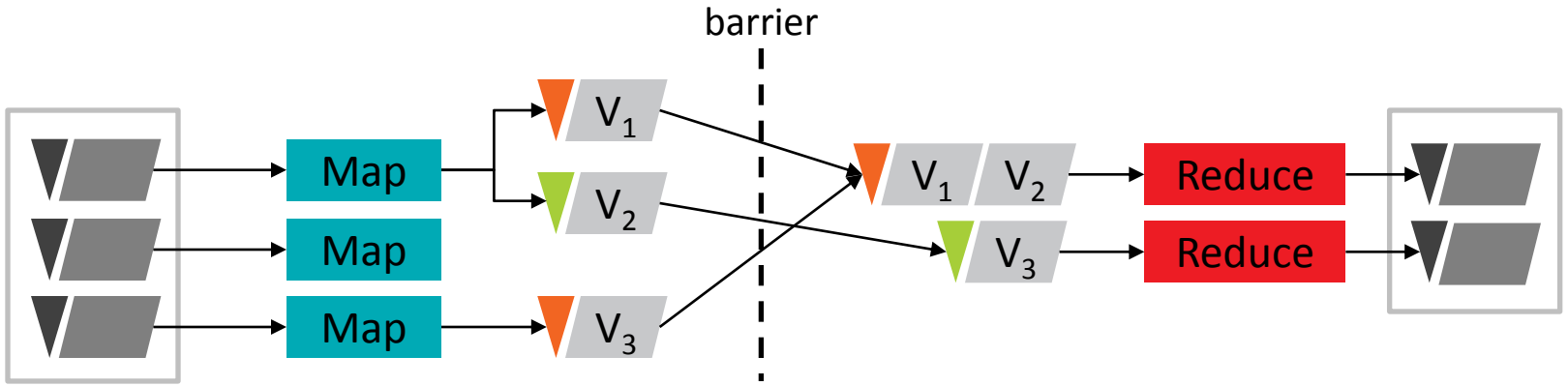
Map: random point inside? Issue k=1, v=1 else k=0,v=1

Reduce: count 0 keys and count 1 keys

Programmer: writes { map, reduce } methods, system does rest

HADOOP

- Open source implementation of MapReduce programming model

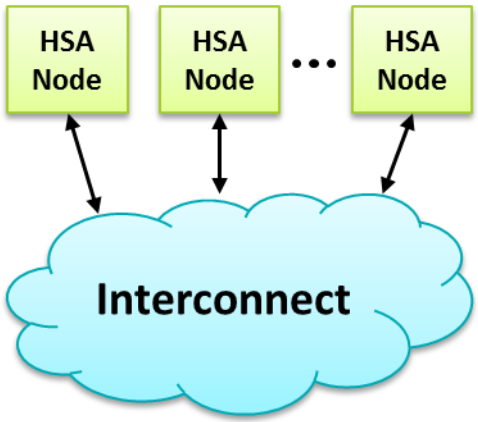


- Runs on distributed network in several Java VMs

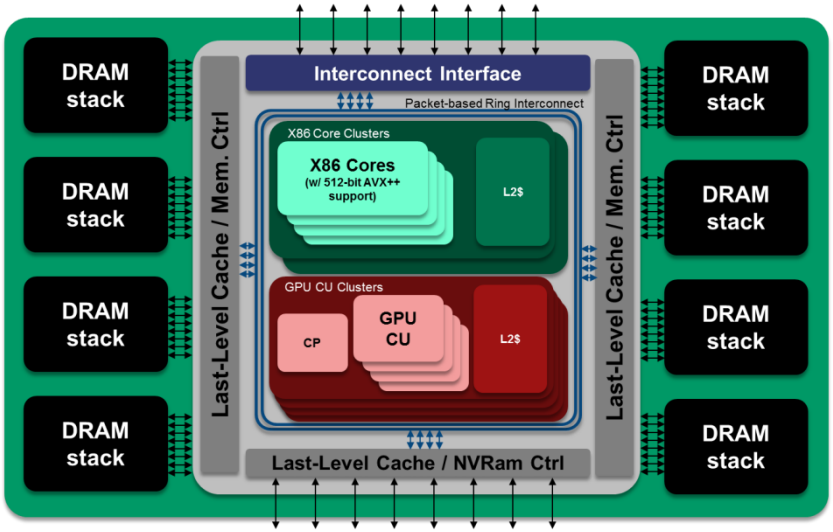
- Distributed file system, reliability guarantees, speculative execution, Java programming language and libraries, implicit parallelism

TARGET ARCHITECTURE

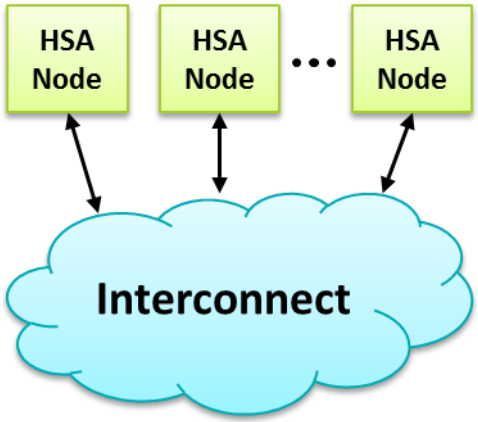
- **Target: CLUSTER of APUs**



- **Two-Level Parallelism:**
  - **Across nodes in cluster**
  - **Within Node (APU)**
    - **Multicore (CPU)**
    - **Data parallel(GPU)**

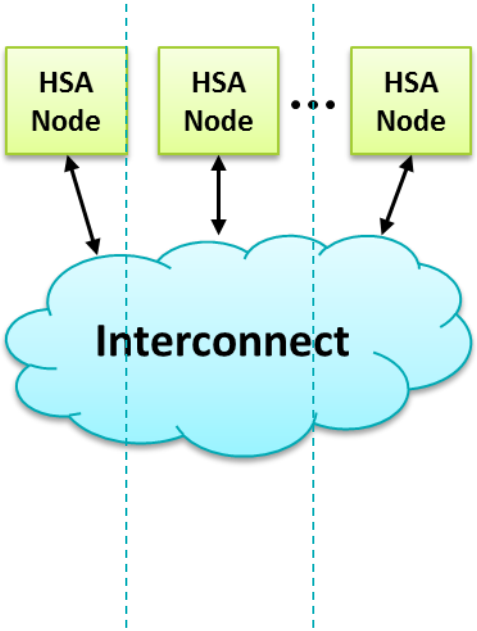


NESTED PROCESSING APPROACH



PARTITION → **WORK LOCALLY** → **COMBINE RESULTS**

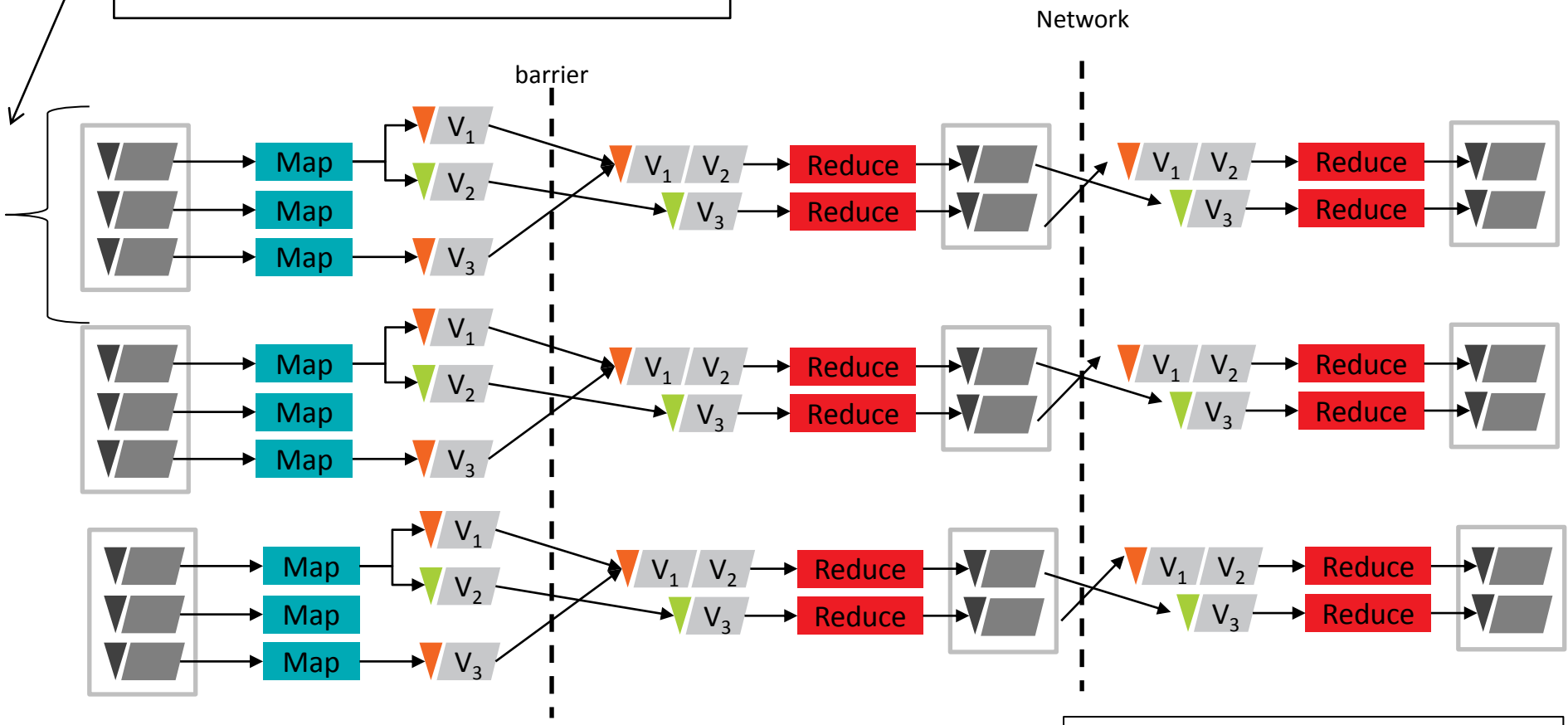
nest: CPU+GPU



WHY APUS – MAP REDUCE-REDUCE

AMD

SPLIT: enough work for one GPU
NO MEMORY LIMIT
CPU+GPU execution



CPU+GPU execution on each node

Final reduction in cluster
Aggregate node's results

HADOOPCL

Implemented with AMD's APARAPI:

Java methods -> GPU

Collaboration with Rice University:

HadoopCL

MapReduce on Distributed Heterogeneous Platforms Through Seamless Integration of Hadoop and OpenCL

Max Grossman[1], Mauricio Breternitz[2], Vivek Sarkar[1]
[1]Rice University, [2]AMD Research
2013 International Workshop on High Performance Data Intensive Computing.
May 2013.

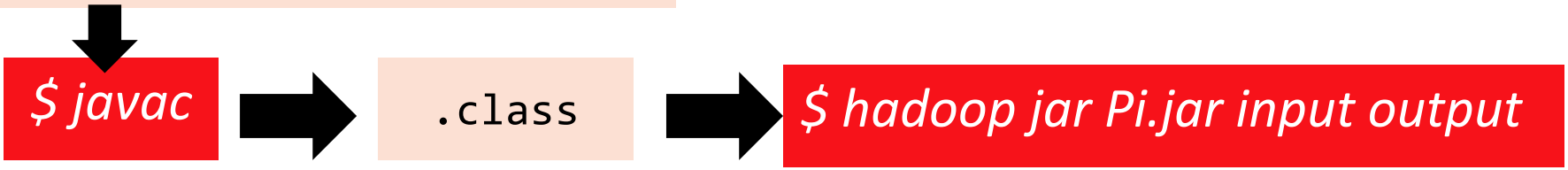
M. Grossman, M. Breternitz, V. Sarkar. "HadoopCL2: Motivating the Design of a Distributed, Heterogeneous Programming System With Machine-Learning Applications."
IEEE Transactions on Parallel and Distributed Systems, 2014

RUNNING HADOOPCL

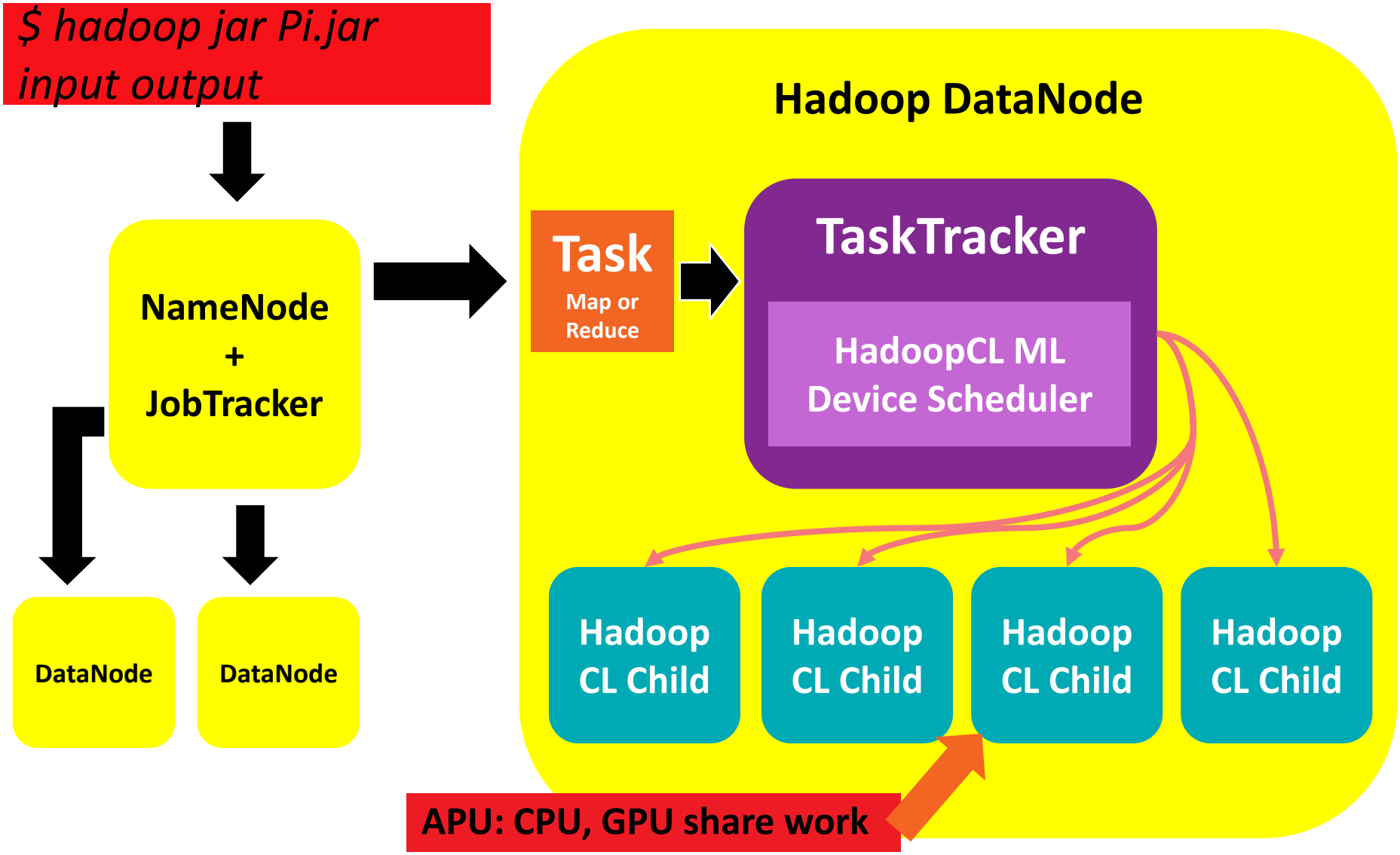
AMD

```
class PiMapper extends

DoubleDoubleBoolIntHadoopCLMap
per {

  public void map(double x,
      double y) {
    if(x * x + y * y > 0.25) {
      write(false, 1);
    } else {
      write(true, 1);
    }
  }
}

job.waitForCompletion(true);
```

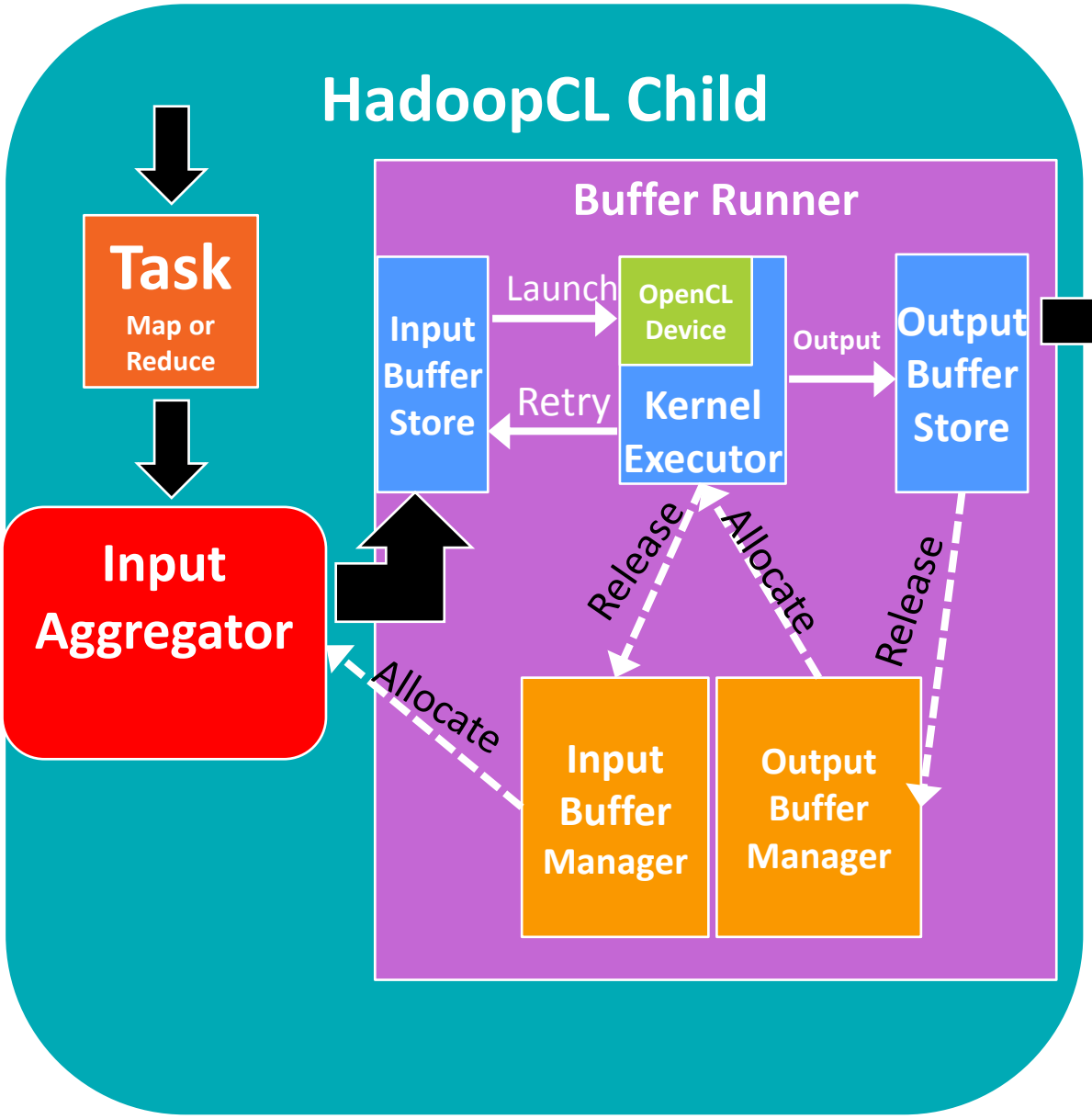
- HadoopCL supports
  - Java syntax & MapReduce abstractions
  - Dynamic memory allocation
  - A variety of data types (primitives, sparse vectors, tuples, etc.)
- HadoopCL does not support
  - Arbitrary inputs, outputs
  - Object references



HADOOPCL CLUSTER ARCHITECTURE



HADOOPCL NODE ARCHITECTURE



Each Child JVM encloses a data-driven pipeline of communication and computation tasks

Kernel Executor handles:

- Auto-generation and optimization of OpenCL kernels from JVM bytecode
- Transfer of inputs, outputs to device
- Asynch launch of OpenCL kernels

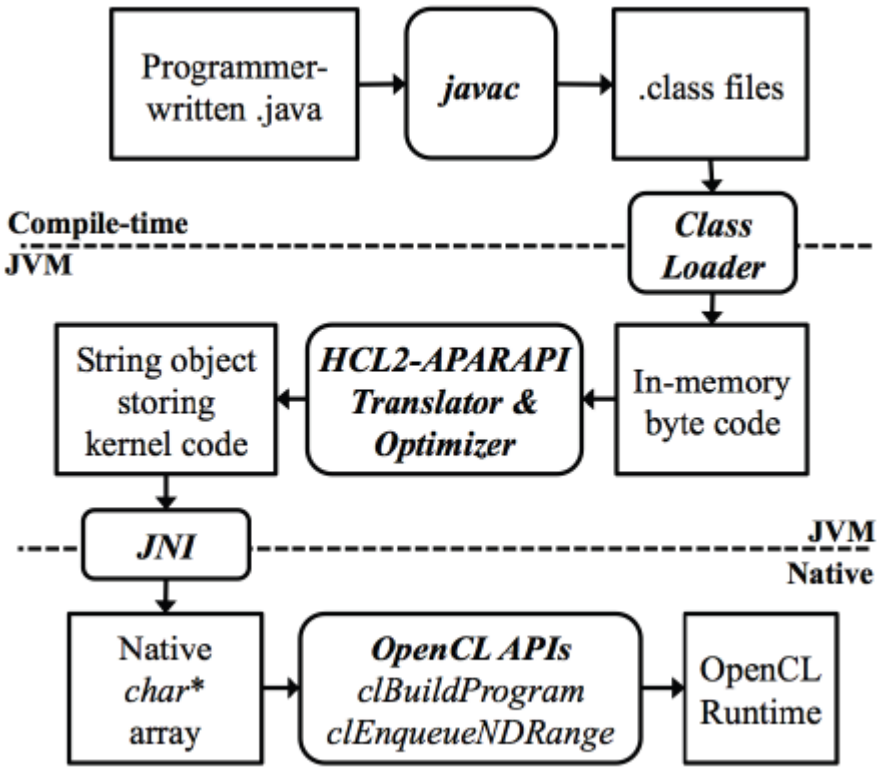
HCL2 EXECUTION FLOW

AMD

compile

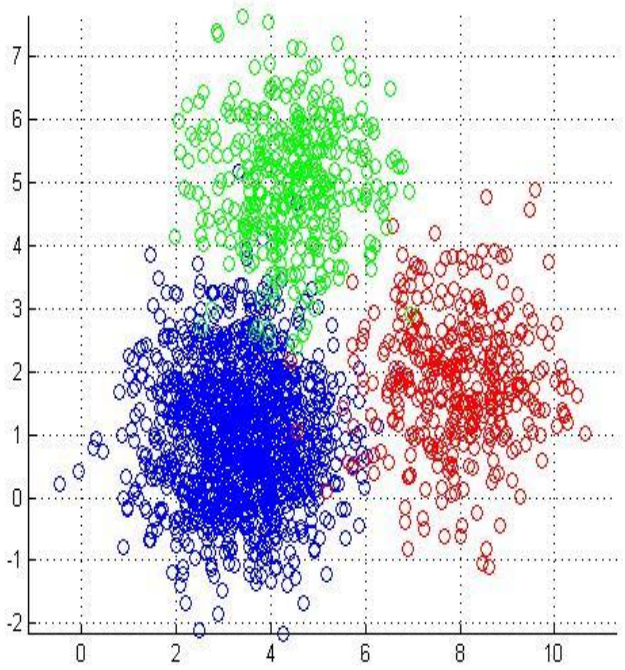
JVM

runtime



EVALUATION

- Mahout Clustering
  - Mahout provides Hadoop MapReduce implementations of a variety of ML algorithms
  - KMeans iteratively searches for K clusters
- Evaluated on 1 NameNode and 3 DataNodes in an AMD APU cluster
- Dataset built from the ASF e-mail archives
  - ~1.4GB
  - 1 iteration of searching for 64 clusters
  - Recorded
    - overall execution time,
    - time spent on compute,
    - time spent on I/O in each mapper and reducer



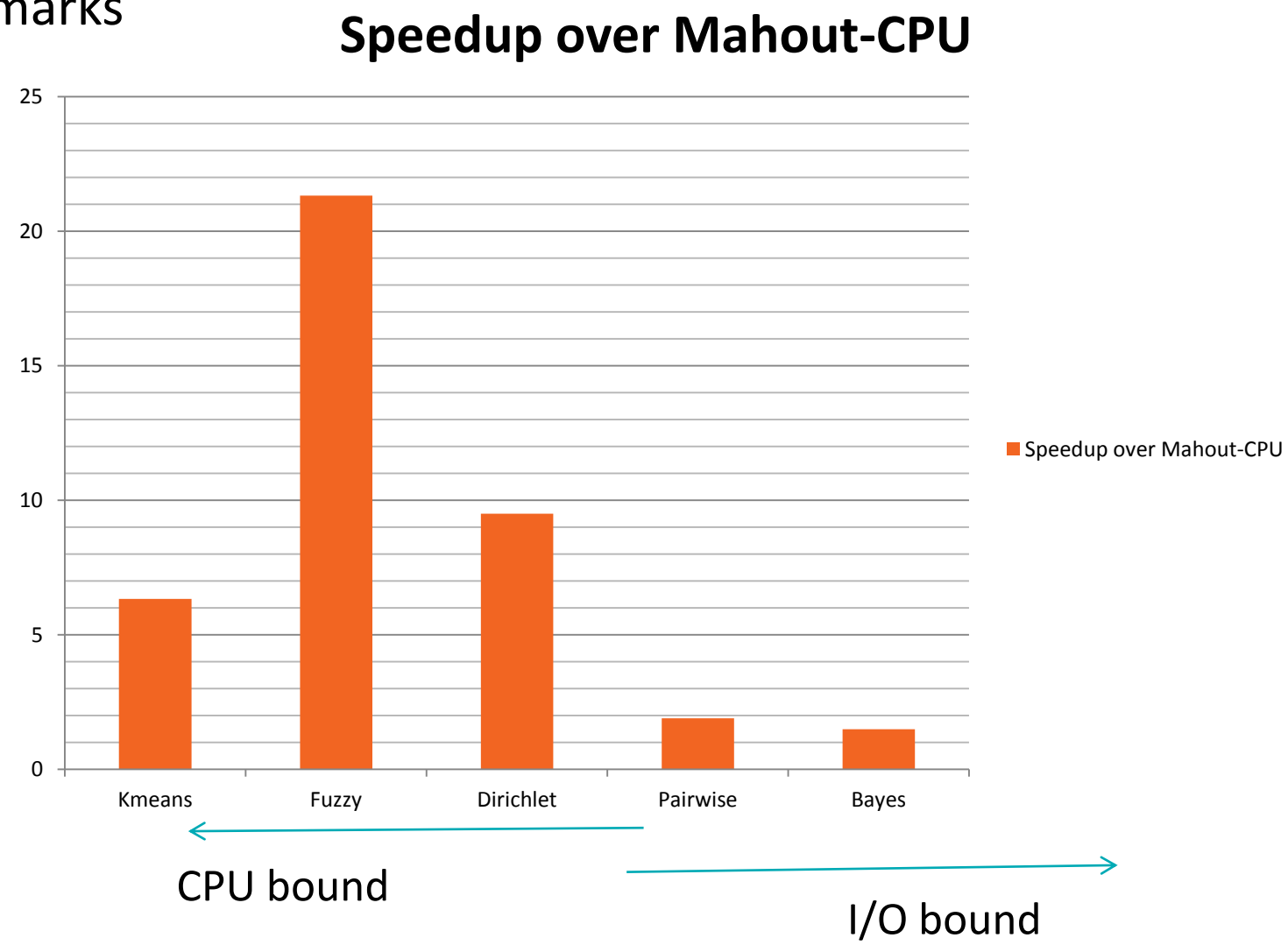
MAHOUT EXAMPLES

- KMEANS
  - Finds clusters
- FUZZY KMEANS
  - Probabilistic “soft clusters”
- PAIRWISE SIMILARITY
  - Recommender pipeline
- NAÏVE BAYES
  - Probabilistic classifier
- DIRICHLET
  - Finds document topics, cluster via probability distribution over ‘topics’

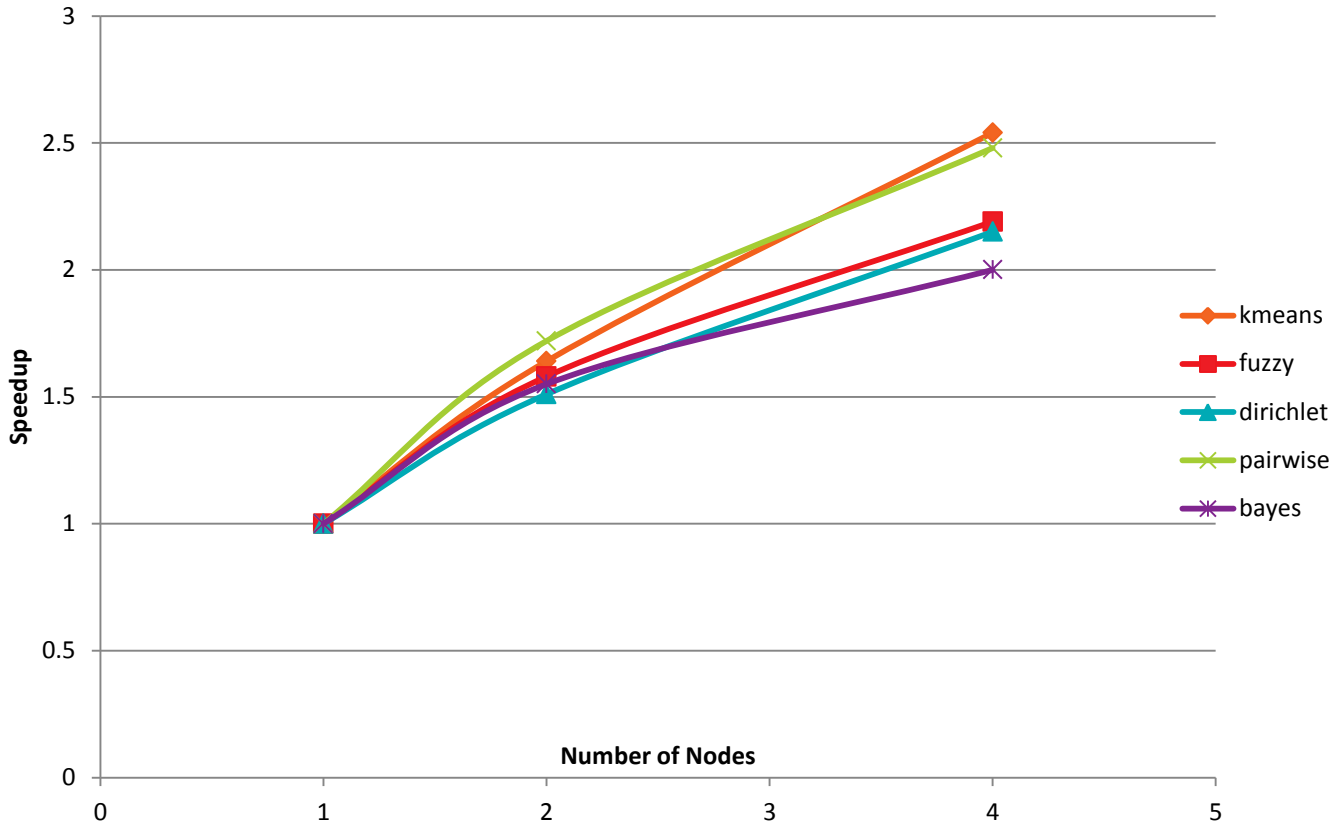
EVALUATION

AMD

Speedup on AMD A10-7300 95w APU for 5 Mahout Benchmarks



SCALABILITY



FUTURE/ONGOING WORK

AMD

- Evaluation on more Mahout applications, more data sets, more platforms
  - Xiangyu Li, Prof David Kaeli /Northeastern University :Mahout Recommenders
- Evaluate potential power savings
- In-depth analysis of effectiveness of machine learning on performance
- Target HSA instead of OpenCL, via Sumatra/APARAPI
- Various performance improvements

APACHE SPARK

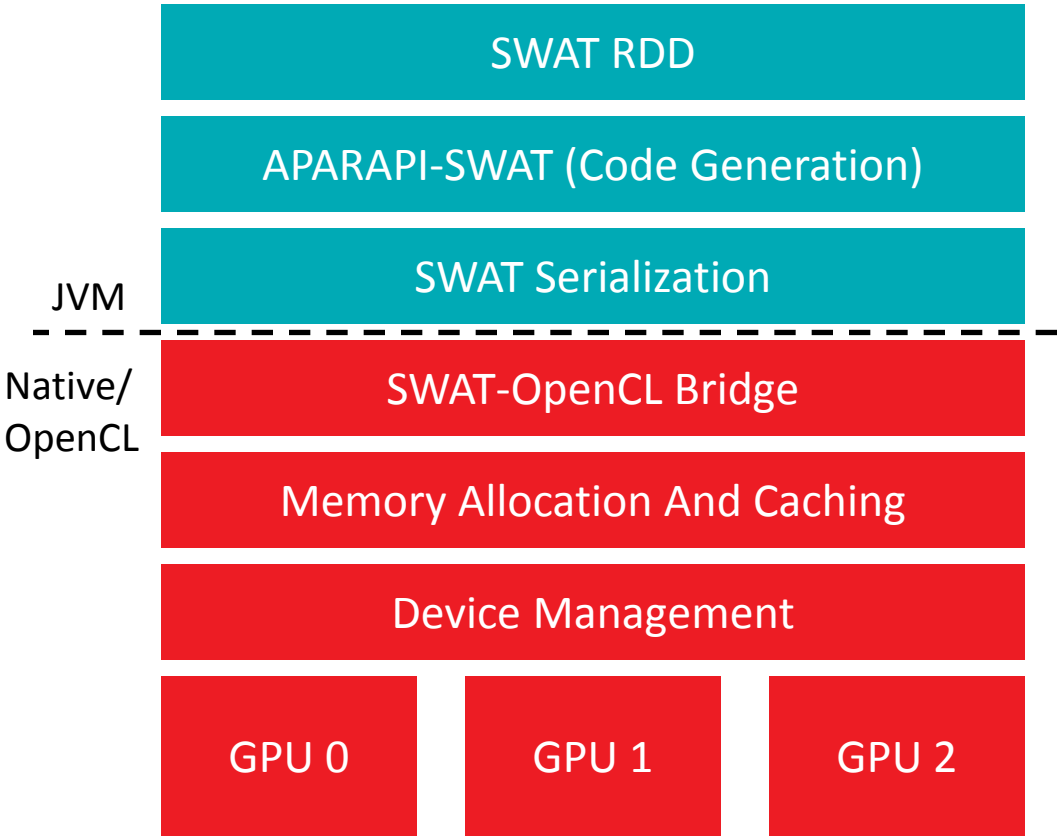
AMD

- Fast, MapReduce-like engine
  - In-memory storage abstraction for iterative/interactive queries
  - General execution graphs
  - Up to 100x faster than Hadoop MR
- Compatible with Hadoop's storage APIs
  - Can access HDFS, HBase, S3, SequenceFiles, etc.
- Great example of ML/Systems/DB collaboration
- http://spark.apache.org

SWAT

```
val rdd = CLWrapper.cl(sc.objectFile(inputPath))
val nextRdd = rdd.map(...)
...
```

SWAT


SWAT RDD
APARAPI-SWAT (Code Generation)
SWAT Serialization
JVM
Native/
OpenCL
SWAT-OpenCL Bridge
Memory Allocation And Caching
Device Management
GPU 0
GPU 1
GPU 2

SWAT

Big wins over HadoopCL:

- Built on Spark. And Scala.
- No HadoopCL-specific data structures required for representing complex data types
  - User-defined classes w/ restrictions, MLlib DenseVector, Mllib SparseVector, Scala Tuple2 for (key, value) pairs, Primitives
- Simpler semantics for some Spark parallel operations
  - e.g. Spark map() forces one output per input, MapReduce allows arbitrary # of outputs (though Spark has flatMap())
- Better locality, caching of on-device data based on broadcast, RDD IDs
- Simplified dynamic memory allocator on GPU
- More stable, nearing production-ready implementation.
- Max Grossman / Rice University

SWAT

Current benchmarks:

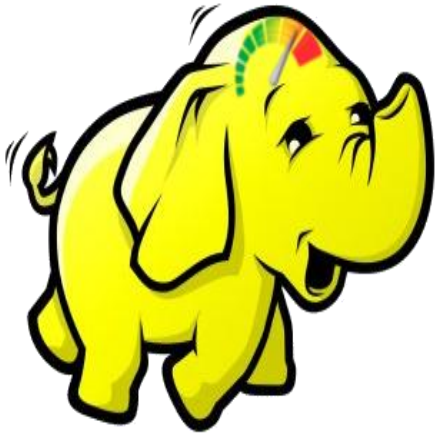
- Fuzzy CMeans, KMeans, Neural Net, Pagerank, Connected Components

Major Challenges:

- Architected as a third-party JAR, internal Spark state is hidden (unlike HadoopCL)
- Garbage collection… Allocation patterns of a SWAT program are very different from those of an equivalent Spark version
- Can we do better auto-scheduling than HadoopCL? Offline? Experiment with other classification algorithms based on IBM work?

CONCLUSION

- HadoopCL offers the flexibility, reliability, and programmability of Hadoop accelerated by native, heterogeneous OpenCL threads
- Using HadoopCL is a tradeoff: lose parts of the Java language but gain improved performance
- Evaluation of KMeans with real-world data sets shows that HadoopCL is flexible and efficient enough to improve performance of real-world applications



Thanks: Max Grossman, max.grossman@rice.edu

RESOURCES

AMD

APARAPI

https://code.google.com/p/aparapi/

http://developer.amd.com/tools-and-sdks/opencl-zone/aparapi/

HADOOPCL paper

https://wiki.rice.edu/confluence/download/attachments/4425835/hpdic.pdf?version=1&modificationDate=1366561784922&api=v2

HADOOP on GPU

http://www.slideshare.net/vladimirstarostenkov/star-hadoop-gpu

HADOOPCL presentation

https://www.youtube.com/watch?v=KMpjFsOO4nw

DEEP NEURAL NETWORK(DNN) ACCELERATION ON AMD ACCELERATORS

JUNLI GU, AMD RESEARCH

MACHINE LEARNING + BIG DATA INDUSTRY TREND

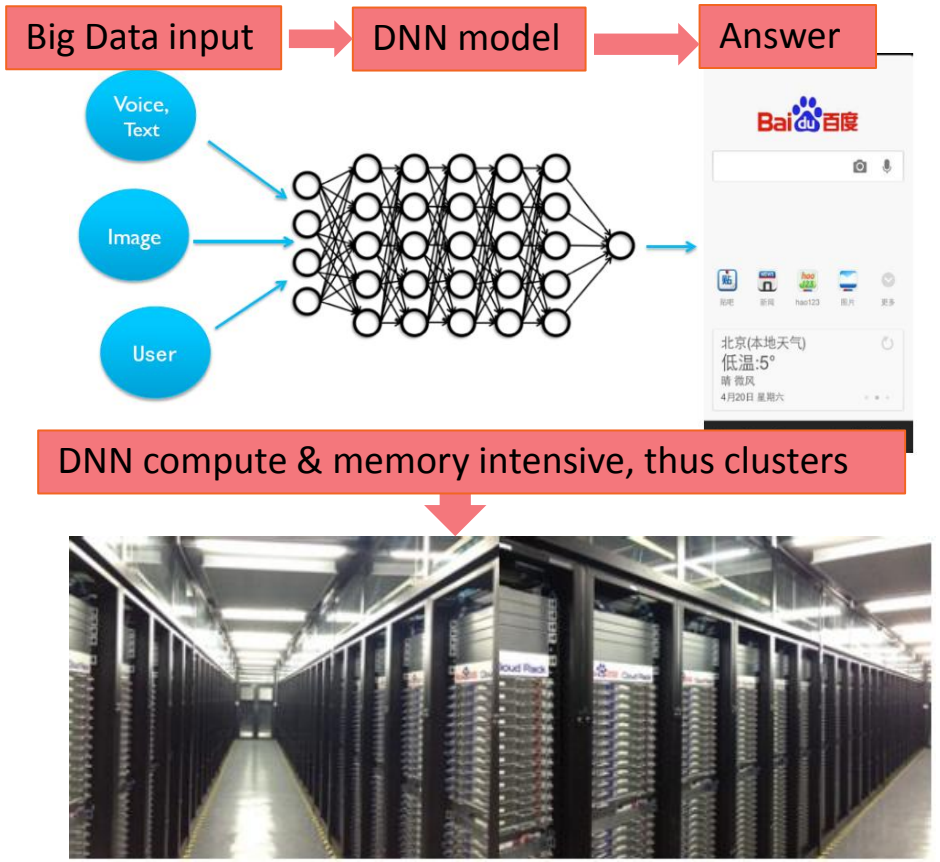
- Why machine learning for Big Data?
  - Original human defined algorithms don't work well for Big Data
  - Competing in machine learning to understand Big Data
- DNN (deep neural networks) is breaking through & leading direction
  - Large scale of image classification/recognition/search
  - Face recognition, Online recommendation, Ads
  - Documentation retrieval, Optical Character Recognition (OCR)
- Long list of companies looking for DNN solutions

DNN + Big Data is believed to be the evolutionary trend for apps & HPC systems.

DEEP LEARNING BRINGS CHALLENGES TO SYSTEM DESIGN

AMD

- **Typical scale of data set**
  - Image search: 1M
  - OCR: 100M
  - Speech: 10B, CTR: 100B
- **Projected data to grow 10X per year**
- **DNN model training time**
  - Weeks to months on GPU clusters
  - Trained DNNs then deployed on cloud
- **System is the final enabler**
  - Current platform runs into bottleneck
    - CPU clusters ➔ CPU + GPU clusters
  - Looking at dGPUs, APUs, FPGAs, ASIC, etc.



WHAT HAVE WE DONE
BASED ON TODAY'S INDUSTRY

AMD

- Focused on three major industry DNN algorithms
- CNN: Convolutional Neural Network (image/video classification)
  - Reference Source: Univ. of Toronto CUDA version (http://github.com/bvlc/caffe)
  - AMD implementation open sourced at https://github.com/amd/OpenCL-caffe
- Multi-layer Perceptron (Voice Recognition)
  - Source: publications, interaction with industry experts and ISVs
  - AMD implementation in C++, OpenCL
- Auto-encoder + L-BFGS training image and document retrieval)
  - Reference Source: Stanford Univ. Matlab code (http://ai.stanford.edu/~quocle/nips2011challenge/)
  - AMD implementation in C++, OpenCL
  - CPU and GPU computing interact frequently

EVALUATION METHODOLOGY AND PLATFORMS

AMD

Implementations based on commercial BLAS libraries

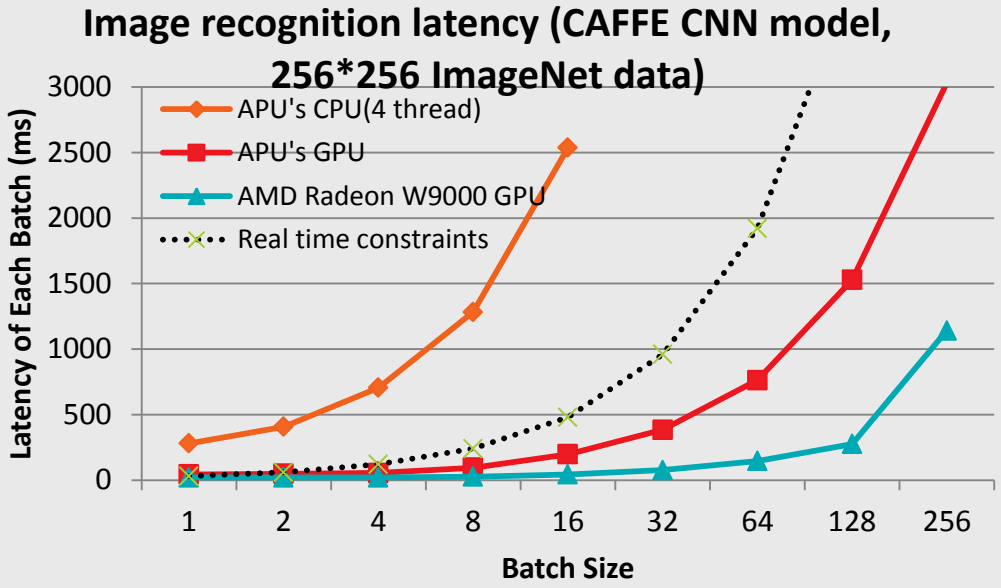
- Mainstream X86 CPUs: C++ & math library
- AMD APUs & GPUs: OpenCL & CLAMDBLAS
- Mainstream GPU: CUDA C & CUBLAS (for competitive purposes)

Platforms

Device Category	Device Name	Throughput (GFLOPS)	Price (USD)	TDP (Watt)	CPU version	AMD OCL version	CUDA version	Note
CPU	Mainstream x86	848	360	**84**	√	√		Realtime power traces
APU series	AMD APU A10-7850k	856	204	**95**		√		Realtime power traces
	AMD APU A10-8700p		150	**35**		√		NA
	Mainstream x86 SOC	848	360	**84**		√		Realtime power traces
GPU	AMD Radeon HD7970	3788.8	322	**250**		√		TDP used
	Mainstream GPU	3977	612	**250**		√	√	TDP used
	W9100	4096	6000	**250**		√		TDP used

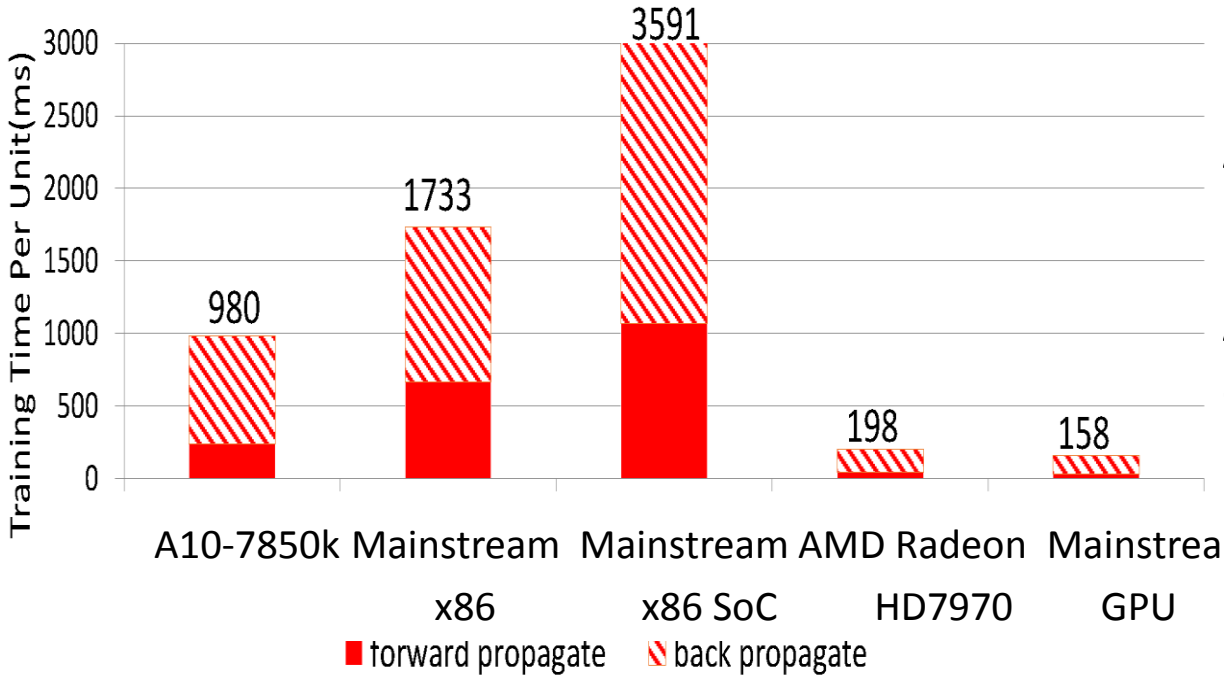
IMAGE RECOGNITION SPEED

- Usually real time image recognition speed constraint is 30 frames per second (fps), 30 ms per image
- The CPU (4-thread) can't meet real time constrains, best 10 fps
- APU can compute up to 84 fps (8x faster than the CPU)
- Discrete GPU vs APU, 2x-3x faster for small (1-8) and big (256) batch size, 4x-5x faster for medium (6-128) batch size



Large scale object recognition

MLP MODEL (VOICE RECOGNITION)



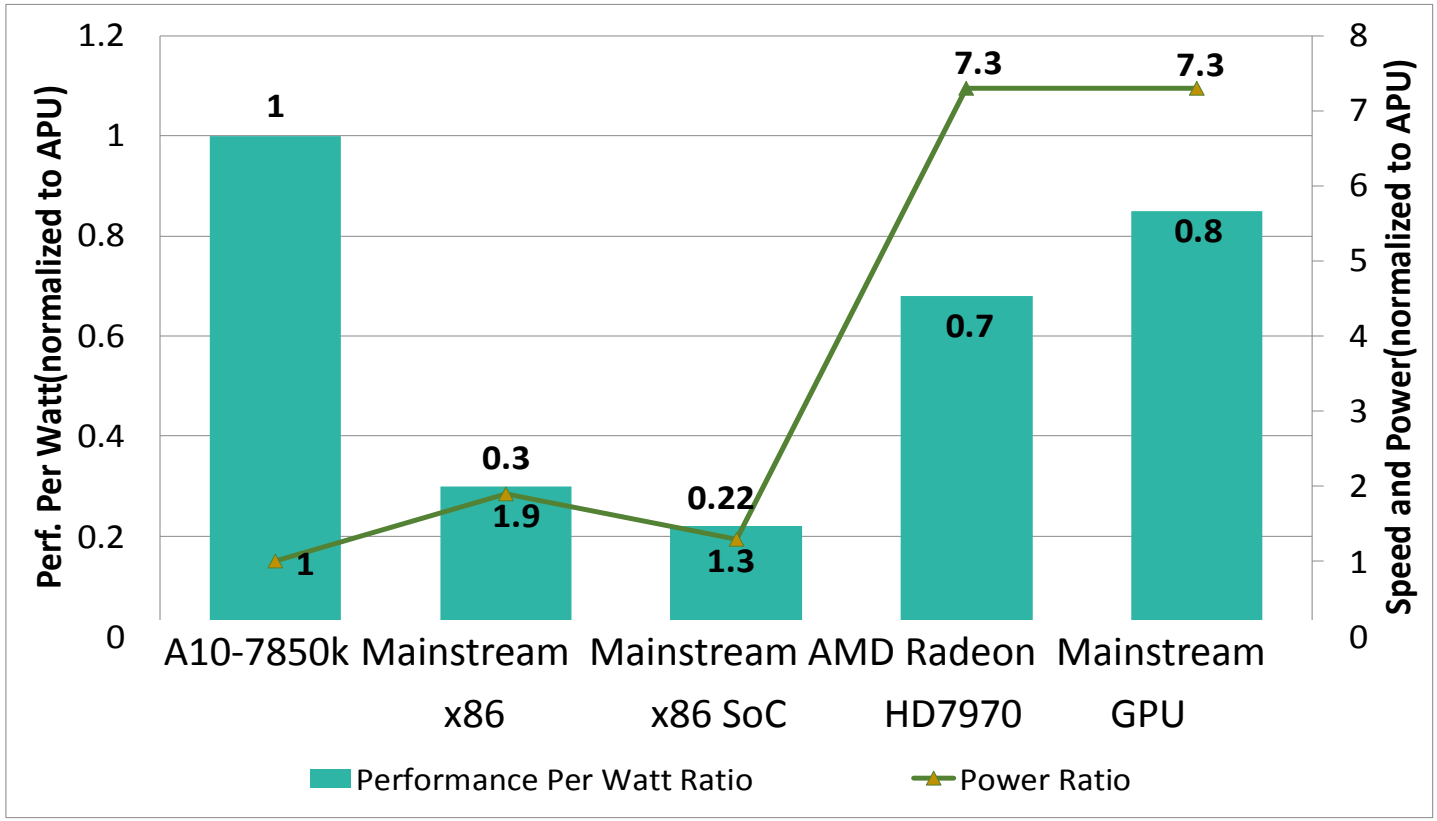
APU vs. Mainstream x86
1.8x speedup

APU vs. Mainstream x86 SOC's
3.7x speedup

Mini-batch size: 1024

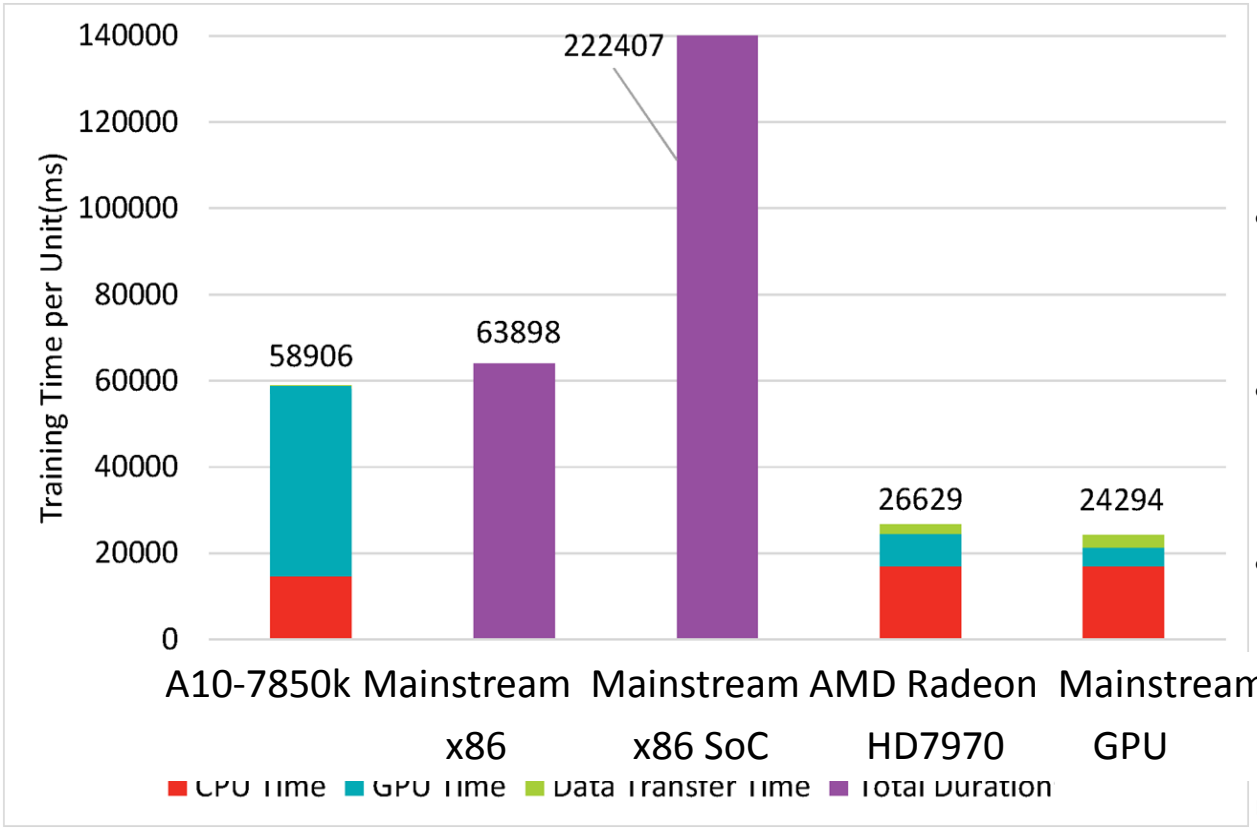
CPU prepares data, GPU computes

POWER/PERF_PER_WATT



- **APU achieves the highest Performance/Watt E.g. 1.2x compared to GPU**
- **GPU achieves 5x performance with 7x power**
- **CPU gets 60% performance with 1.9x power**

AUTOENCODER (IMAGE AND DOCUMENT RETRIEVAL)

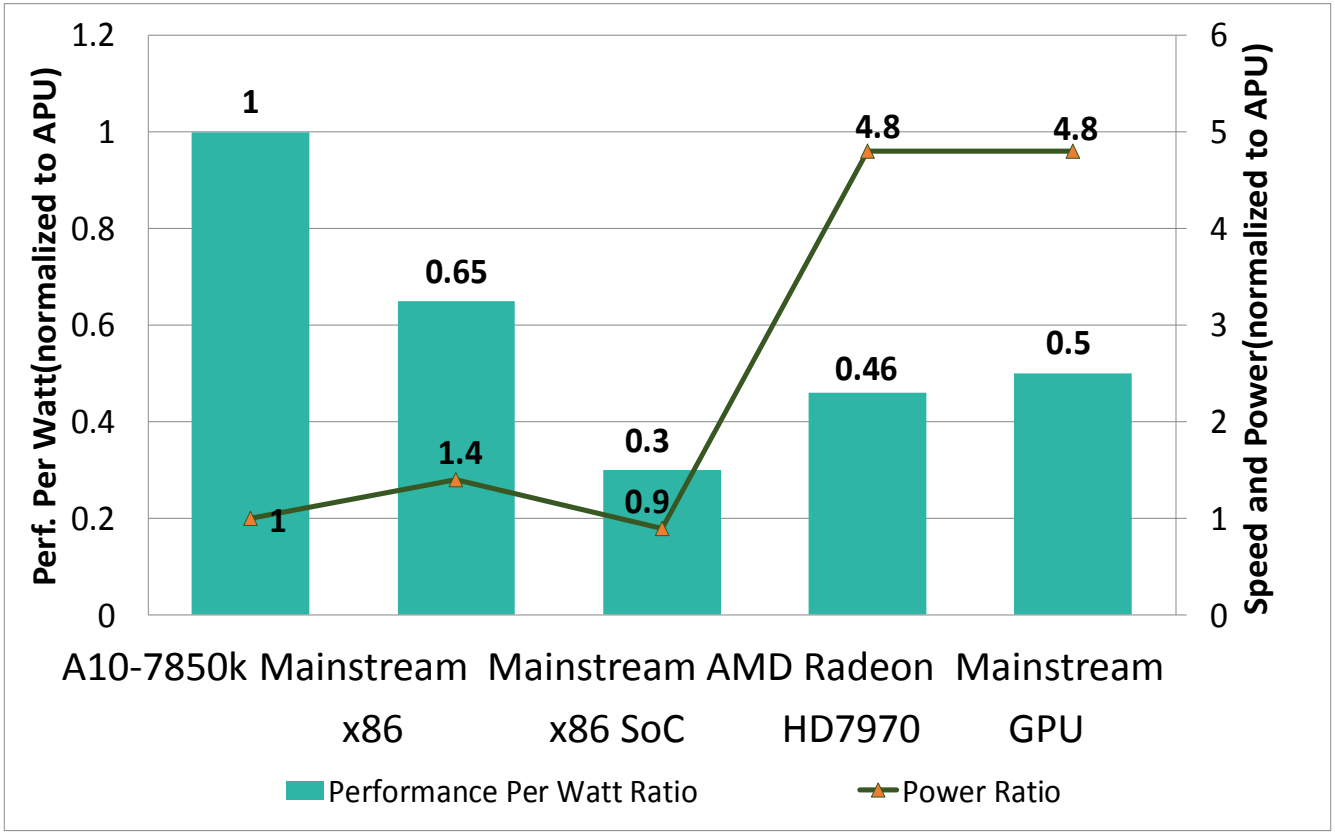


- **Algorithm is mix of CPU+GPU compute**
- **APU vs. Mainstream x86 8% slow down**
- **APU vs. Mainstream x86 SOC's 3.8x speedup**
- **The larger the batch size is, the bigger advantage APU presents.**

Data: CIFAR10, Mini-batch size: 2048

CPU: L-BFGS; GPU: Autoencoder forward and backward propagation

POWER/PERF_PER_WATT

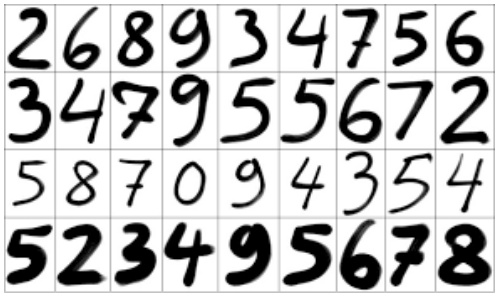


- **APU achieves the highest Performance/Watt E.g. 2x compared to dGPU**
- **GPU achieves 2x performance with 5x power**
- **CPU gets 90% performance with 1.4x power**

REAL CASE TRAINING

AMD

- MNIST Training through MLP Model
  - Handwritten digits , 60000 images
  - Mini-batch size 1024, 200 epochs



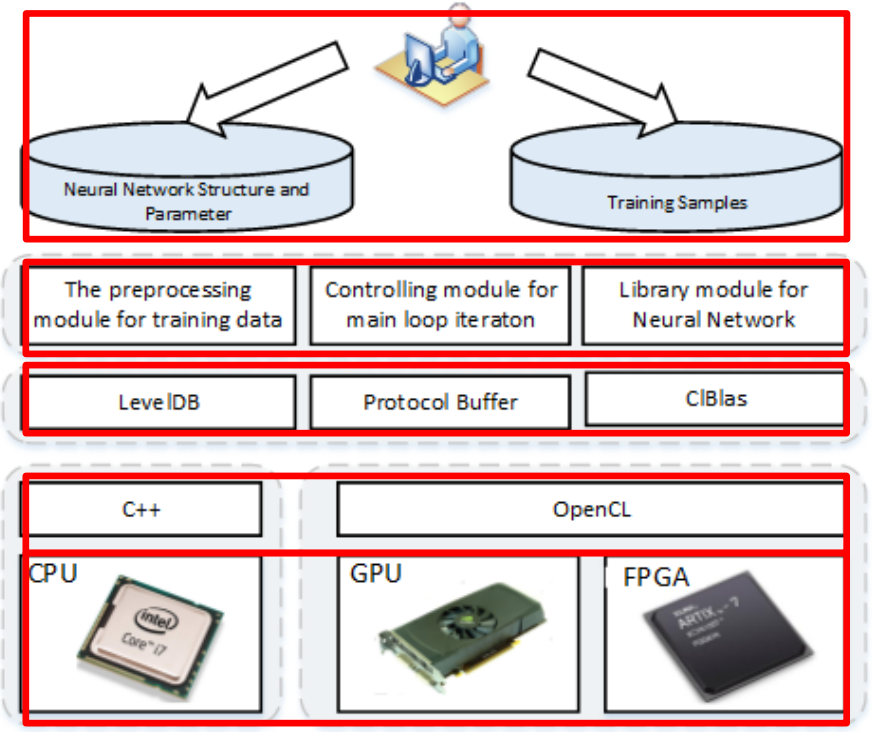
		APU A10-7850	GPU HD7970	GPU vs. APU
Training Process	Time	362 second	192 second	1.9x speedup
	Average Power	47 Watt	250 Watt	5.3x power
	Energy	17k Joule	40k Joule	2.4x energy
Predicting Process	Time	8.1 second	3.5 second	2.3x speedup
	Average Power	37 Watt	250 Watt	6.8x power
	Energy	300 Joule	875 Joule	2.9x energy

PUBLIC RELEASE OF OPENCL-CAFFE

HTTPS://GITHUB.COM/AMD/OPENCL-CAFFE

AMD

- We open sourced OpenCL-caffe framework
  - Welcome everyone to join us in improving on APU or other research
- Users only need to provide CNN framework, parameters and training samples.
  - Framework will train automatically



- Contain three modules:
  - Preprocess the image data by LevelDB and Protocol Buffer. Convert convolutional computations into matrix operation and use CLBLAS to accelerate
  - CPU is responsible for data processing and main loop iteration
  - GPU device is responsible for major kernel computation

CONCLUSIONS

AMD

- For DNN domain:
- APUs achieve up to 2x higher performance per watt efficiency, compared to mainstream GPUs
  - For auto-encoder, GPU consumes 5.3x more power to achieve 2.4x speedup
- APUs offer the most compelling performance/ watt and performance / $$
- Architectural advantages:
  - APUs have very large unified address space
  - Remove GPU's device memory limitation and data transfer bottleneck, which suits better for Big Data inputs

A PRACTICAL IMPLEMENTATION OF BREADTH-FIRST SEARCH ON HETEROGENEOUS PROCESSORS

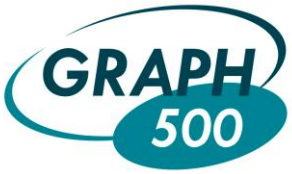
MAYANK DAGA
AMD RESEARCH

BREADTH-FIRST SEARCH (BFS)

AMD

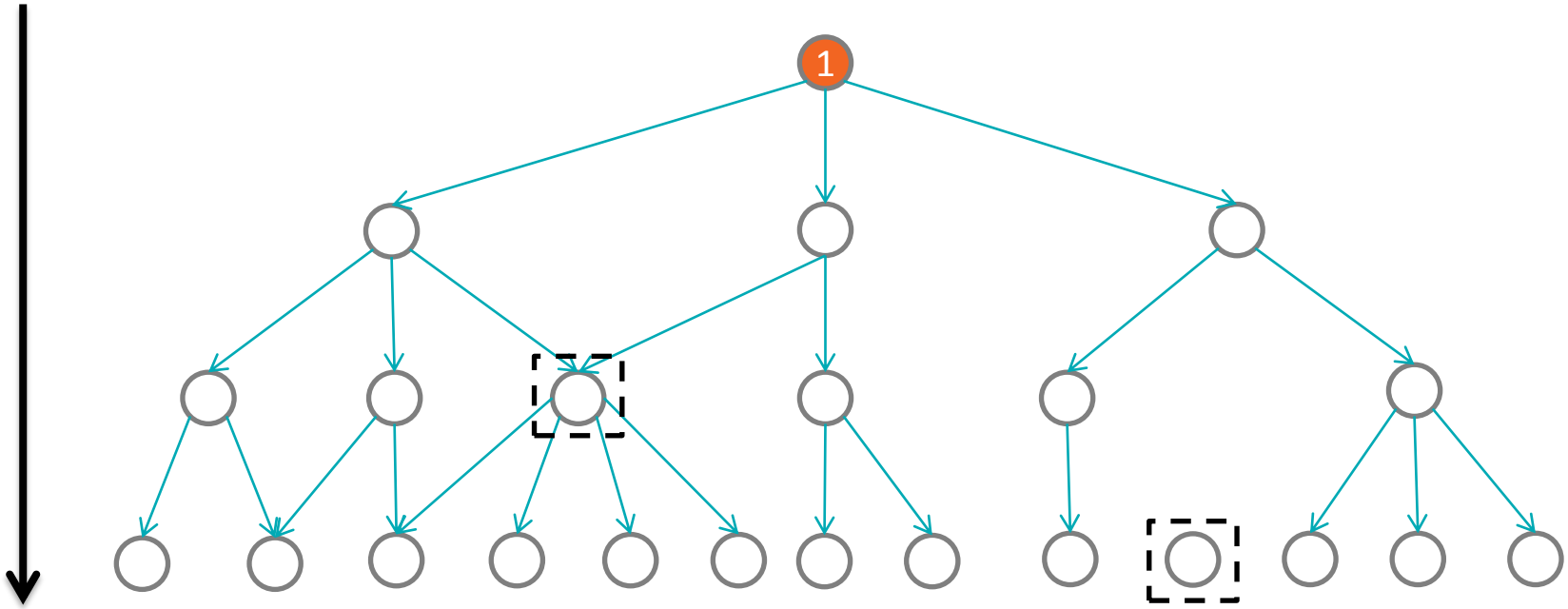
- BFS is a fundamental primitive used several in graph applications
  - Social network analysis
  - Recommendation systems

  - Graph500 benchmark
    (www.graph500.org)
    - Cybersecurity, Medical Informatics,
    - Data Enrichment, Social Networks, and Symbolic Networks



- Accelerating BFS can provide widespread advantages given its pervasive deployment

BREADTH-FIRST SEARCH - 101



Top-Down BFS

visited nodes find their unvisited children

1 visited node

unvisited node

BFS AND GPUS

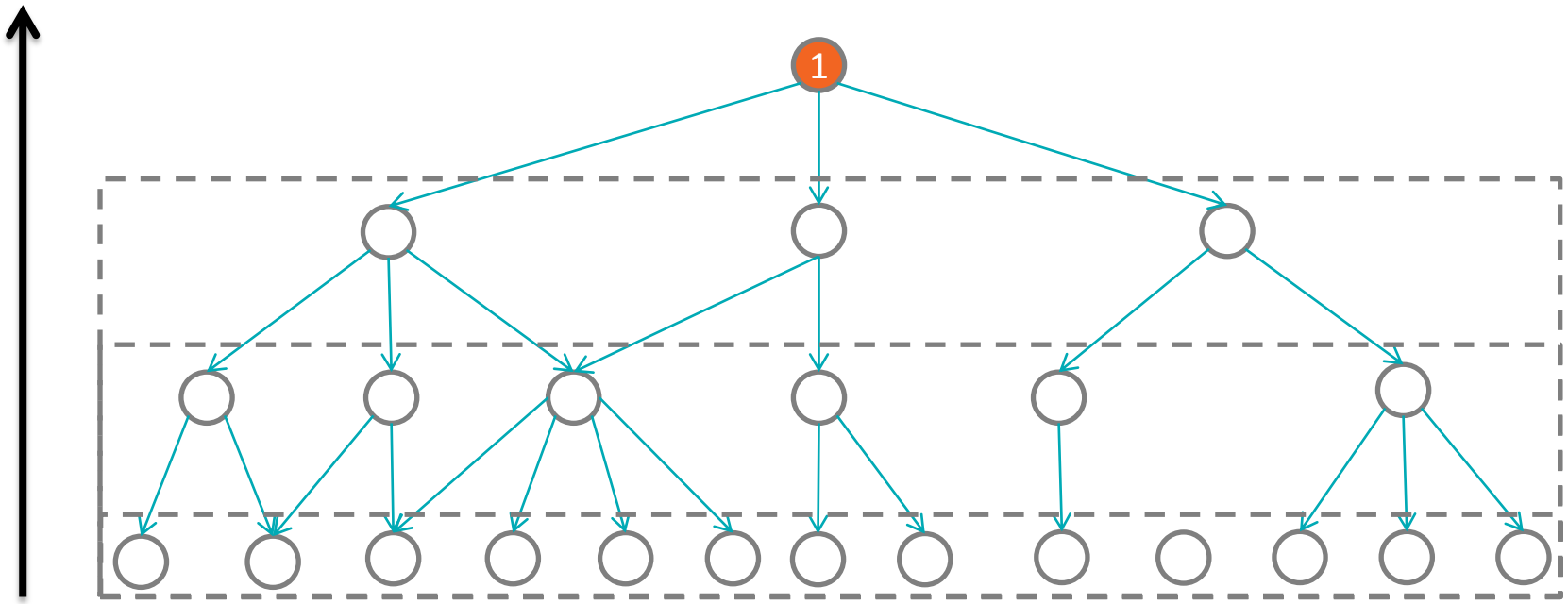
- Breadth-first search is *not* a GPU-friendly application
  - Load imbalance
  - Irregular memory access patterns

- Recent work has demonstrated a credible BFS implementation[2]
  - Tracks level or depth of nodes

- Does *not* create a BFS-tree as required by Graph500
  - Does not have widespread use

[1] D. Merrill, M. Garland, and A. Grimshaw. "*Scalable gpu graph traversal*". In Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '12, New York, NY, USA, 2012.

BREADTH-FIRST SEARCH - 201



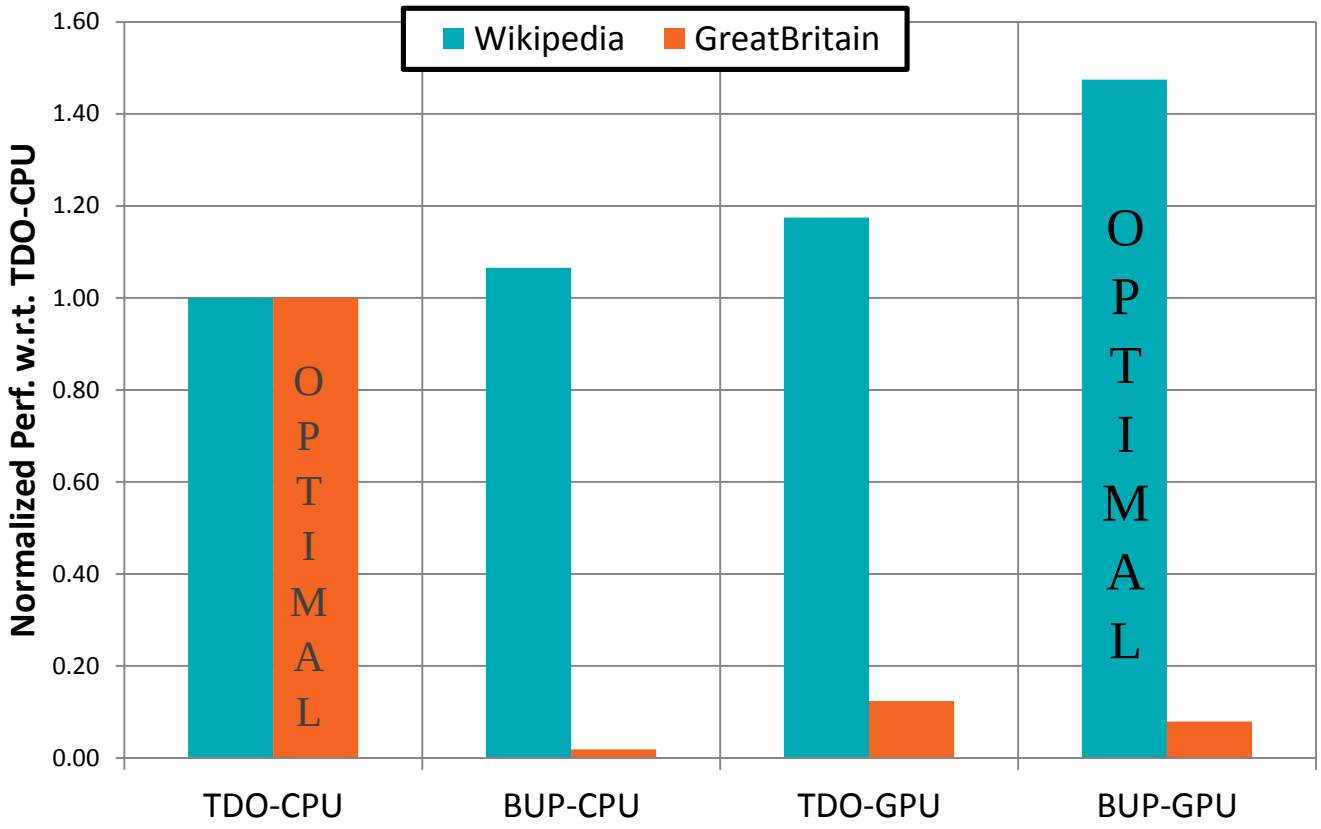
Bottom-Up BFS

<u>unvisited</u> nodes find their <u>visited</u> parent

1 visited node
unvisited node

[2] S. Beamer, K. Asanovic, and D. Patterson, *"Direction optimizing breadth-first search"*. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12.

NO ALGORITHM OR PLATFORM IS A PANACEA FOR BFS



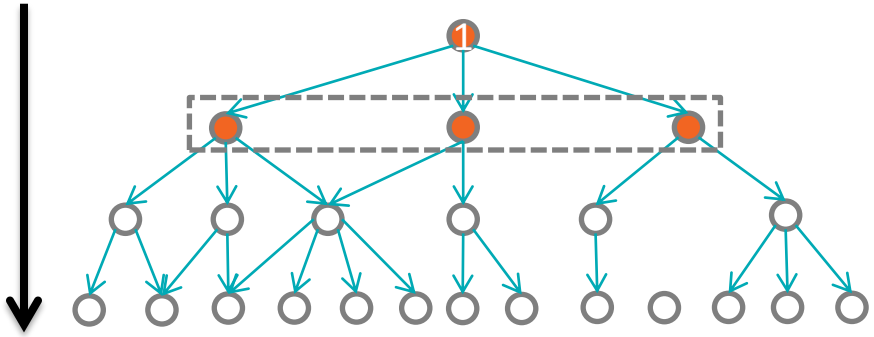
TDO – Top-Down BFS
BUP – Bottom-Up BFS

HYBRID++ BFS TRAVERSAL ALGORITHM

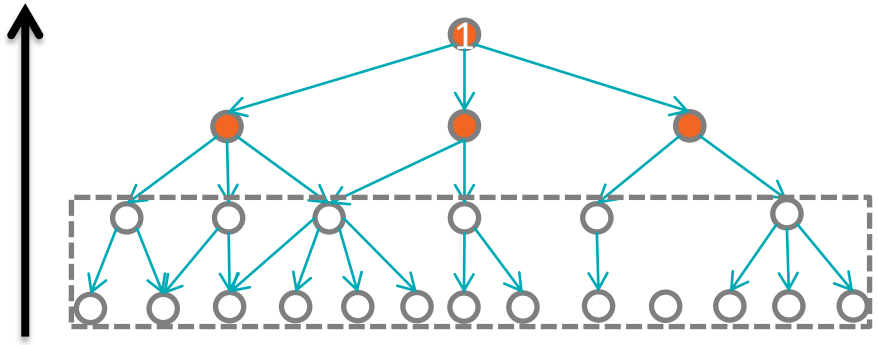
AMD

HYBRID GRAPH TRAVERSAL TECHNIQUE[1]

- Top-Down

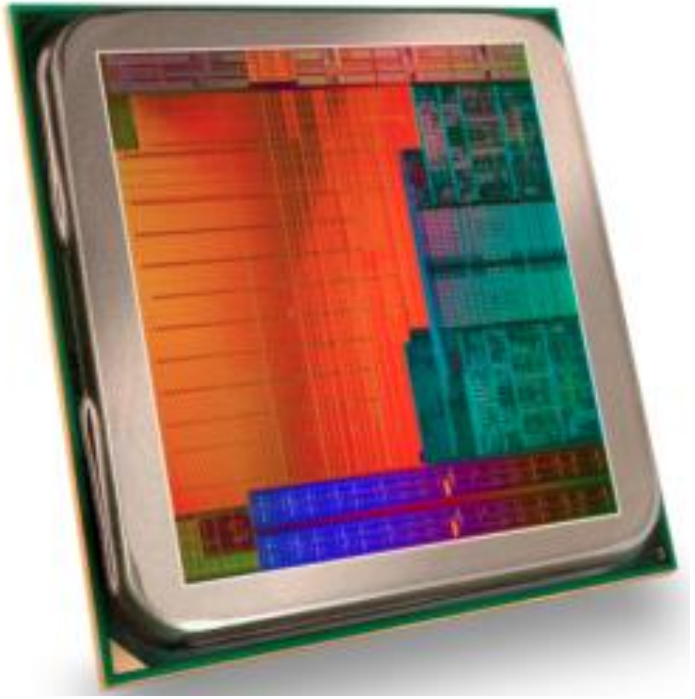


- Bottom-Up



Hybrid Platform of Execution

- AMD Accelerated Processing Unit (APU)



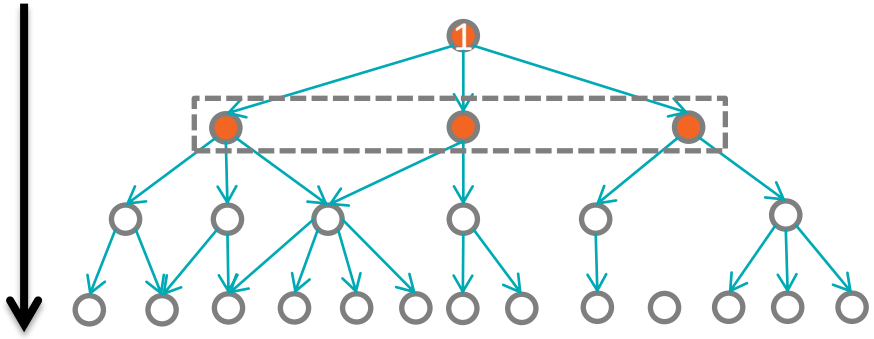
OUTLINE

- Introduction
- Optimal platform and optimal algorithm
- Why APU
- GPU accelerated Bottom-Up BFS
- Results

OPTIMAL PLATFORM FOR HYBRID ALGORITHM

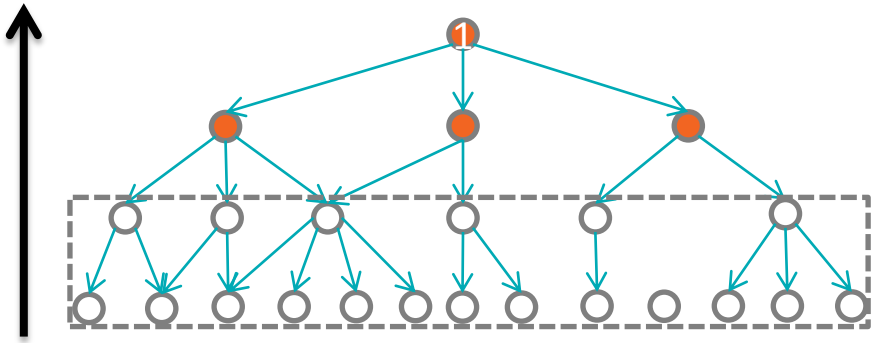
AMD

Top-Down Algorithm



- Amount of parallelism is limited
- Suitable for initial and final iterations
- Ideal for CPU

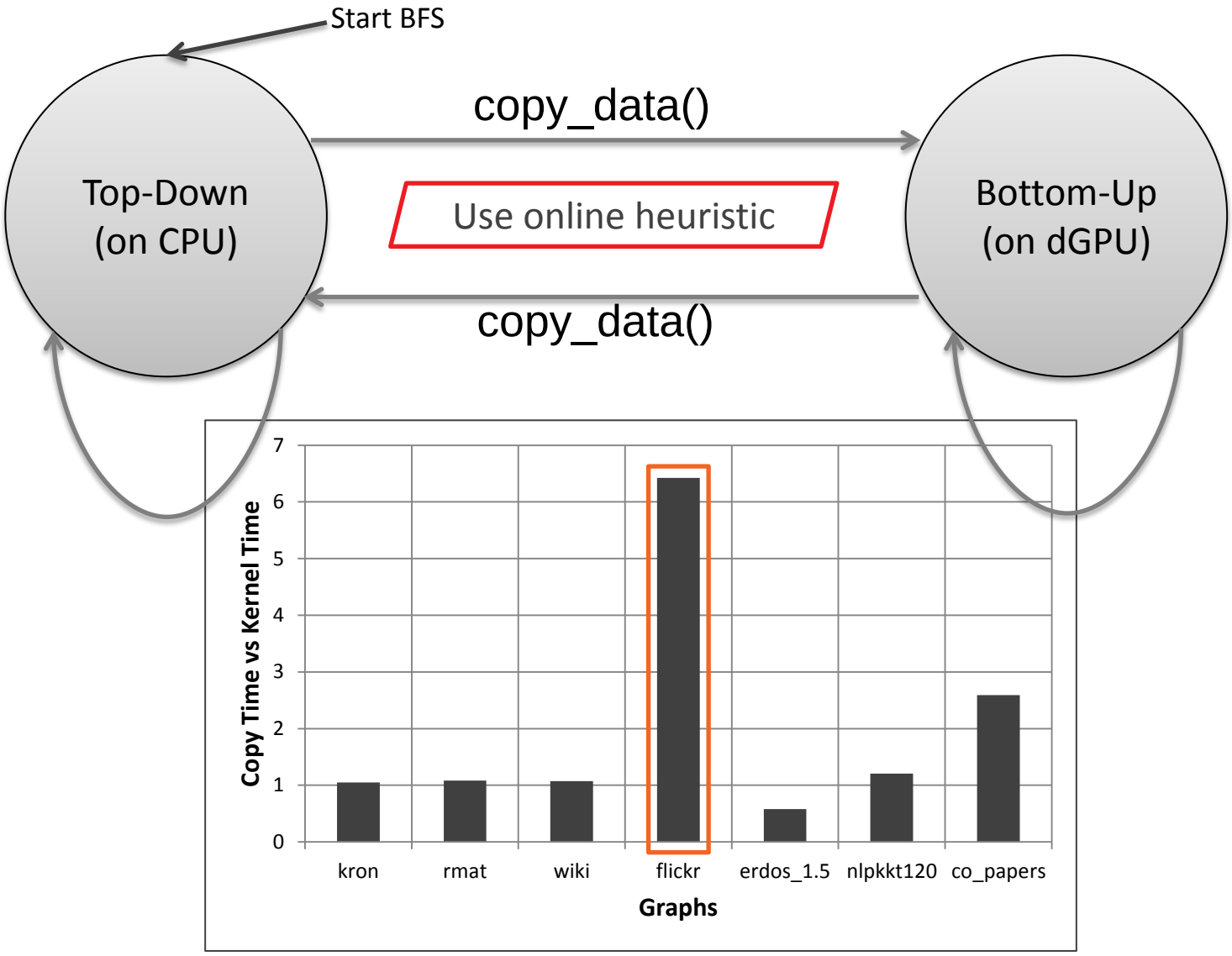
Bottom-Up Algorithm



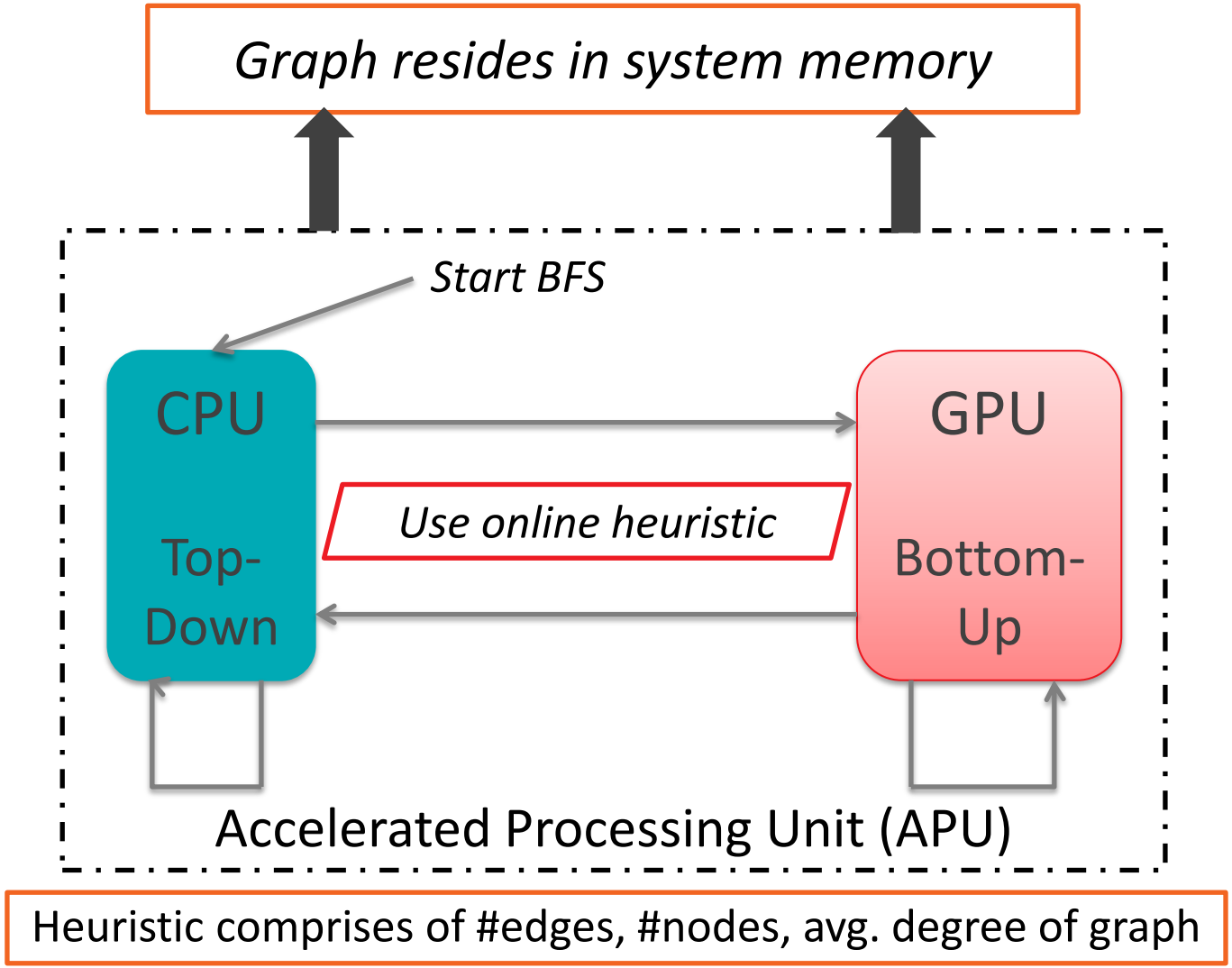
- Amount of parallelism is abundant
- Suitable for intermediate iterations
- Ideal for GPU

HYBRID++ ON DISCRETE GPU

WHY APU?



ACCELERATING BFS USING HYBRID++ ALGORITHM



BOTTOM-UP BFS

AMD

- Consecutive GPU threads map to consecutive vertices

```
function bottom-up-step()
    for all the vertices in the graph
        has the vertex, 'v', been visited?
            if not visited, for all the neighbors of 'v'
                has the neighbor, 'n' been visited?
                    if yes, 'n' is the parent of 'v'
                        update data structures
```

Load imbalance

Scattered reads

Atomic update

unvisited nodes find their visited parent

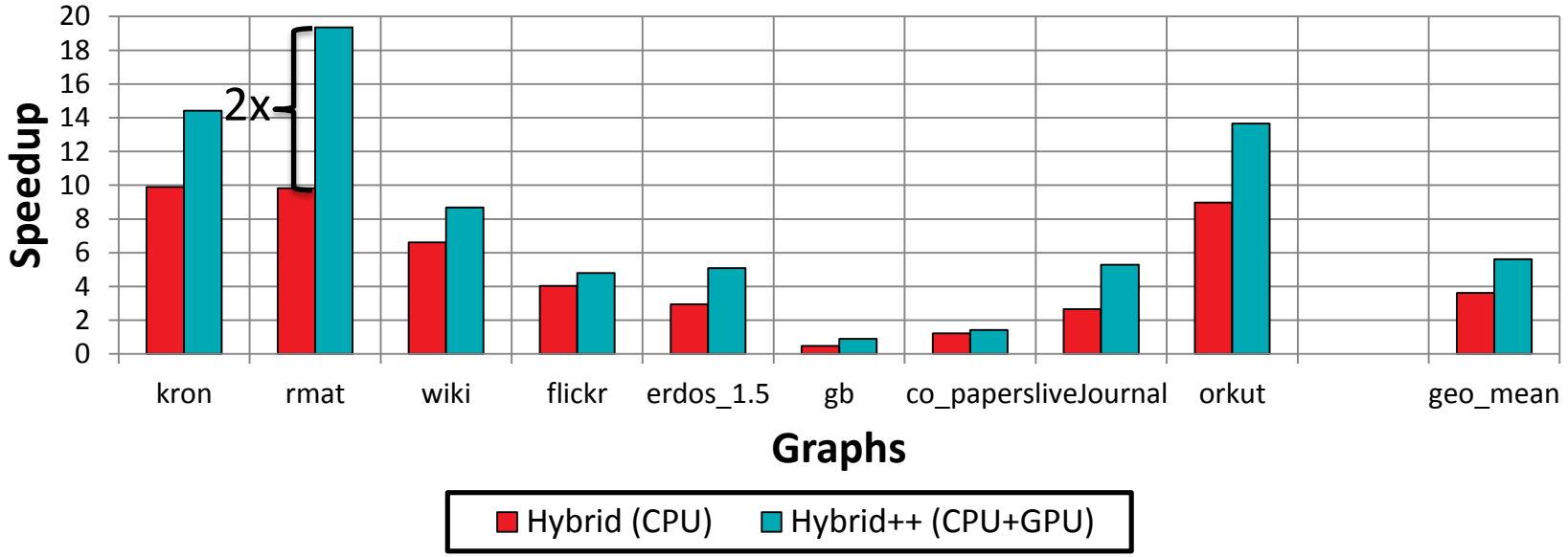
EXPERIMENTAL SETUP

AMD

- AMD A10-7850K “Kaveri” APU
  - Quad-core CPU + 8 GCN Compute Units with 95W TDP

- Variety of input graphs
  - 0.5 million to 14 million nodes
  - 8 million to 134 million edges

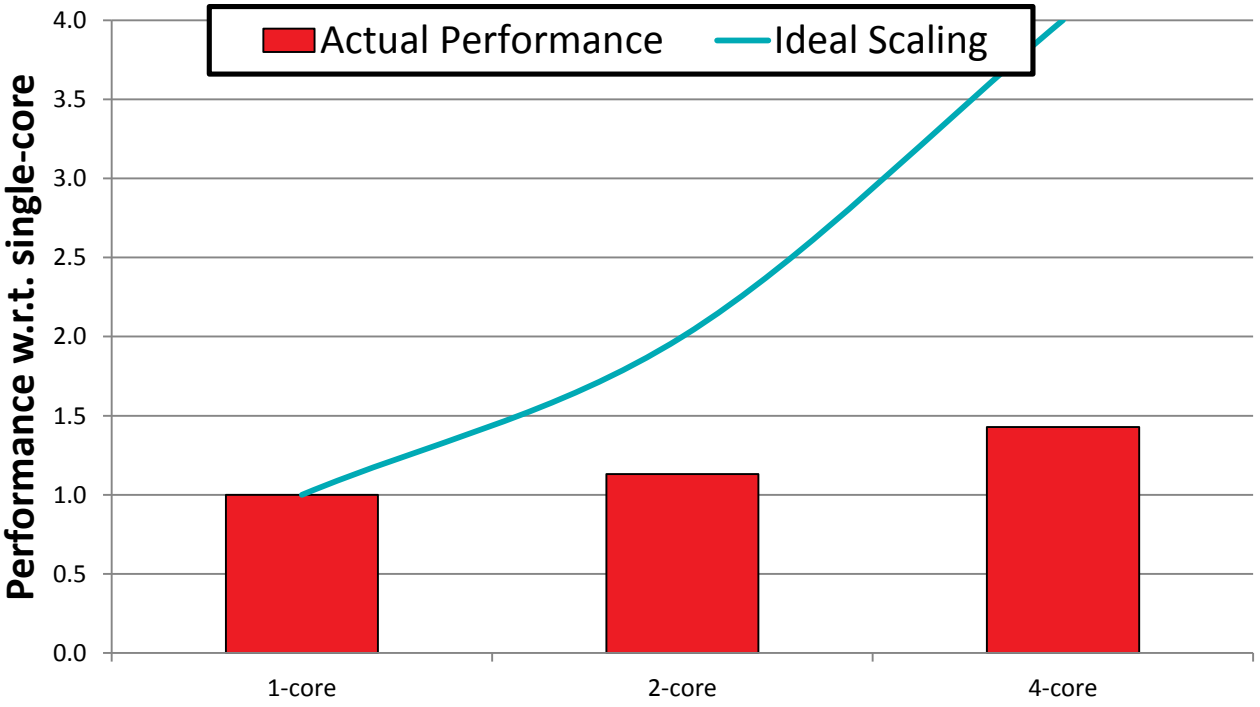
PERFORMANCE OF HYBRID++ ON AMD A10-7850K APU



Y-axis represents speedup over the 4-core OpenMP reference BFS algorithm in Graph500

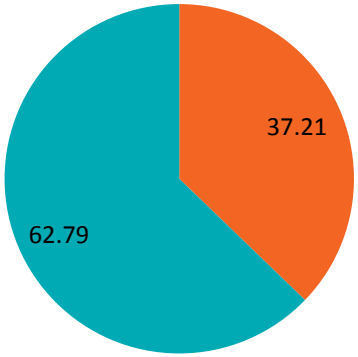
- Speedup by
  - algorithmic improvements: ~10x (best) and ~3.6x (avg.)
  - algorithmic + platform improvements: ~20x (best) and ~5.6x (avg.)

CPU SCALING

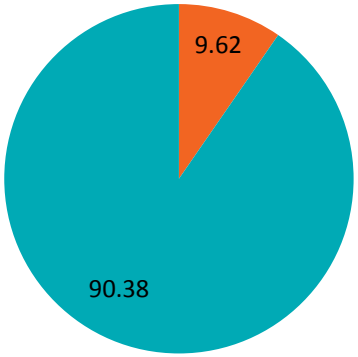


- Experiment was conducted using CPUs of AMD A10 7850K “Kaveri” APU
- Performance depicted is the geometric mean of various inputs
  - Performance does not scale linearly

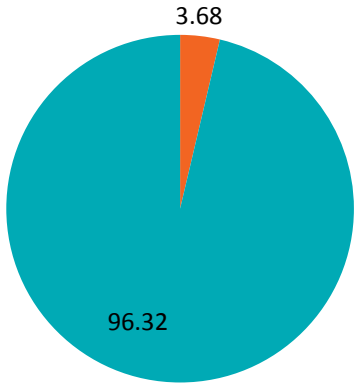
HYBRID++ TRULY UTILIZES BOTH CPU AND GPU



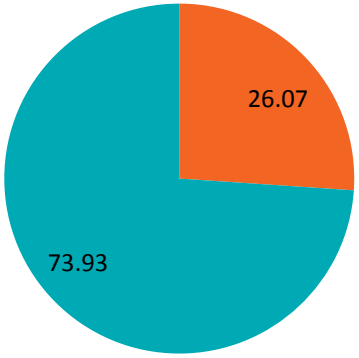
erdos_1.5



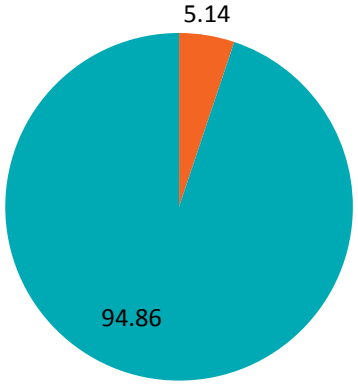
wiki



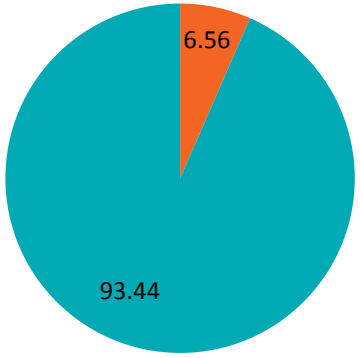
orkut



rmat



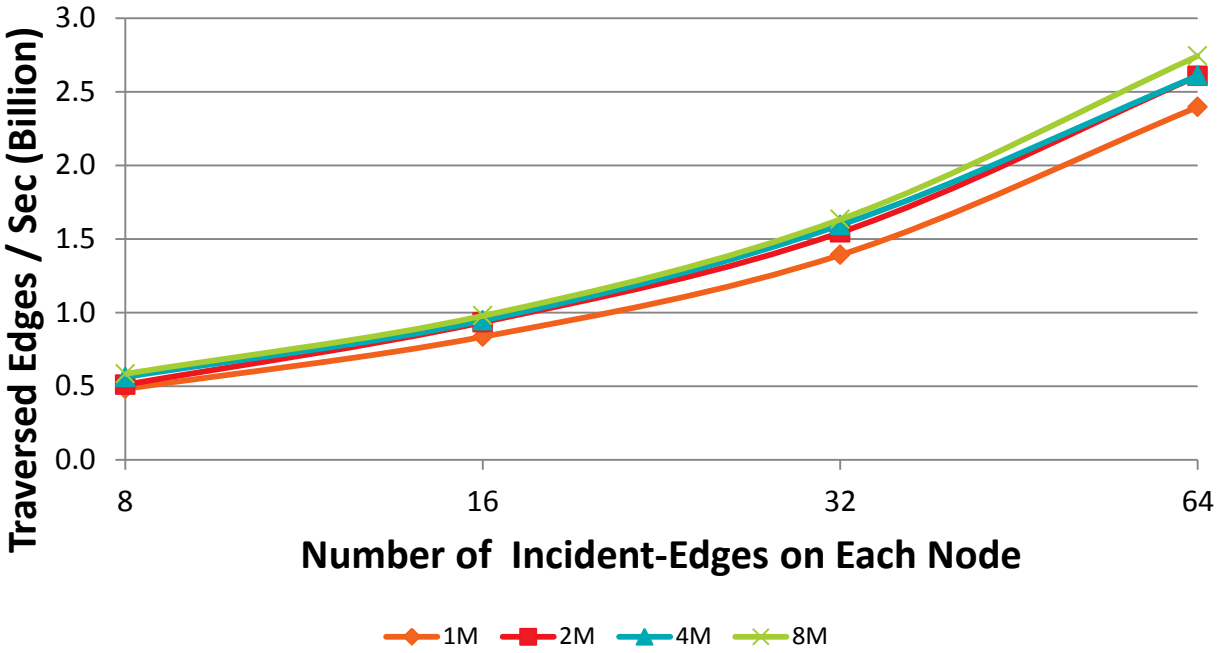
live_journal



kron

CPU GPU

SCALABILITY OF HYBRID++



The legend depicts the number of nodes in the graph in million

- Scales well with both #nodes and #edges
- Particularly beneficial to social-network type graphs (top-right corner)

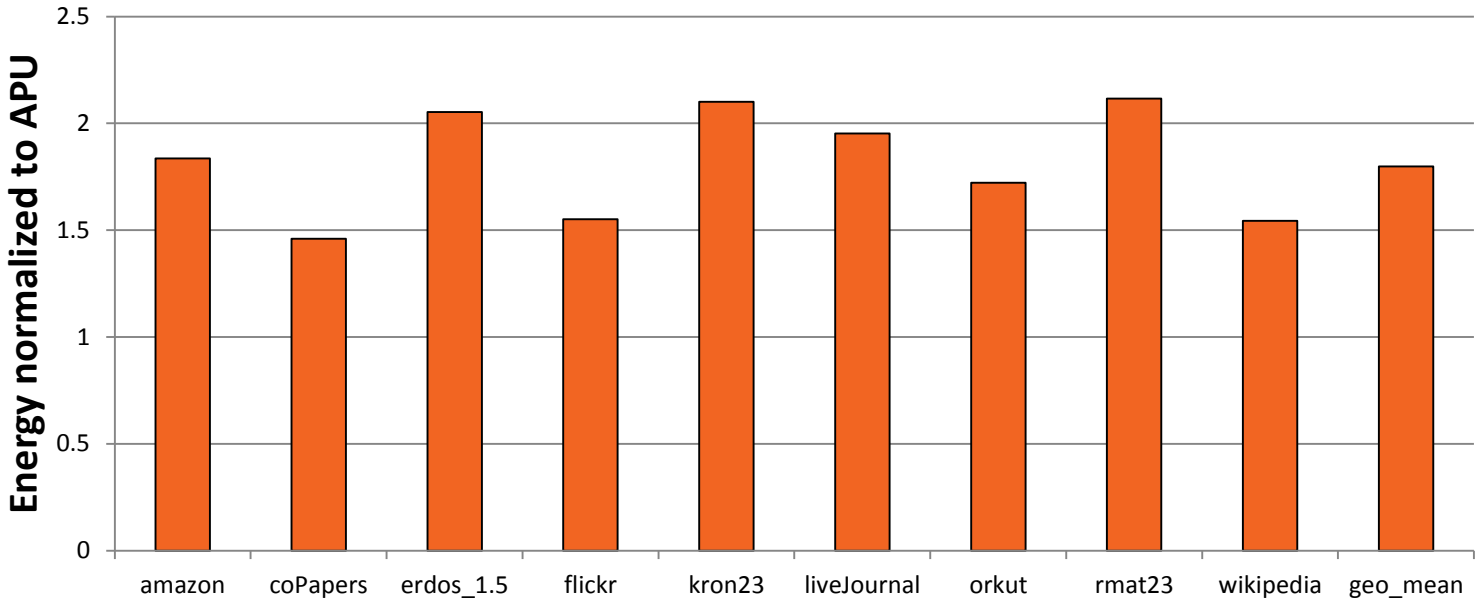
APU VS DISCRETE GPU

AGAIN WHY APU

System	Number of Vertices in Graph500	
	<= 8 million	> 8 million
Discrete GPU *(memory size = 4GB)*	✓	✗
APU	✓	✓
CPU	✓	✓

Limit on the size of input for discrete GPU

ENERGY CONSUMPTION ON APU VS DISCRETE GPU



Y-axis represents performance per watt over Graph500 implementation on the APU

On avg. the discrete GPU consumes 80% more energy

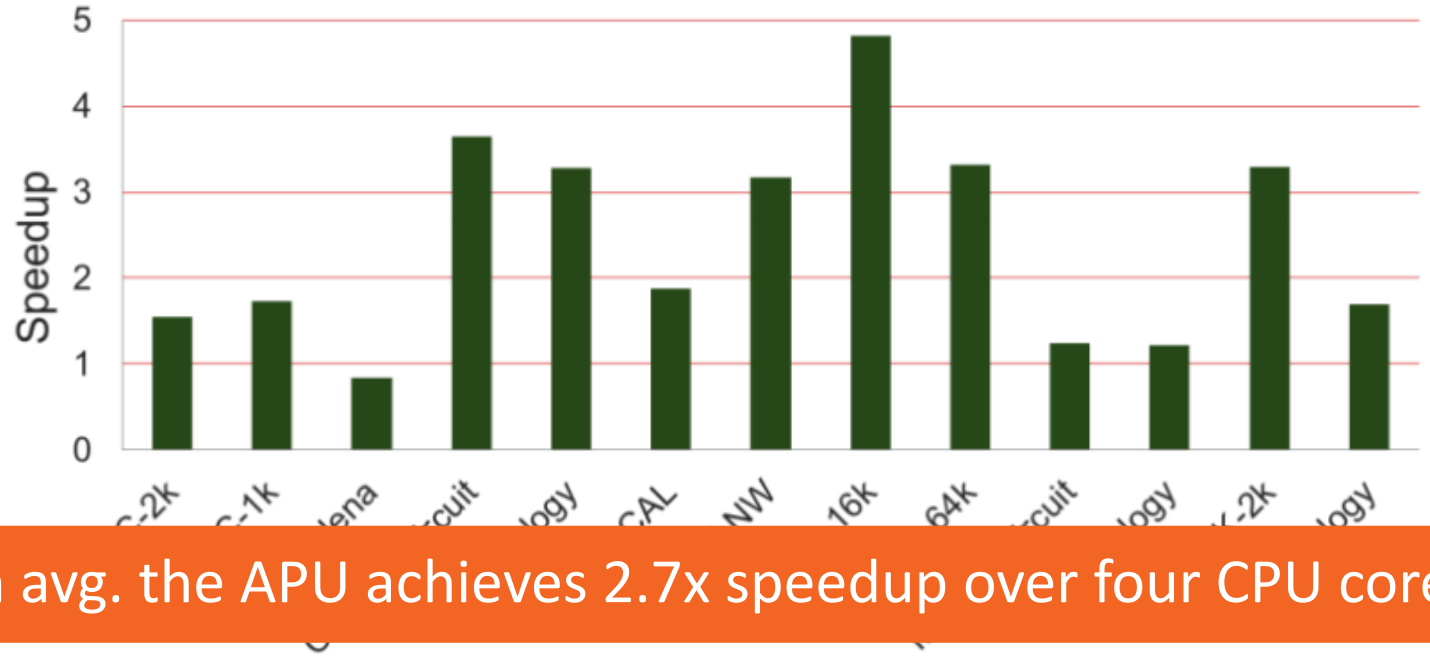
Graph Processing

PANNOTIA BENCHMARK SUITE

HTTPS://GITHUB.COM/PANNOTIA/PANNOTIA

AMD

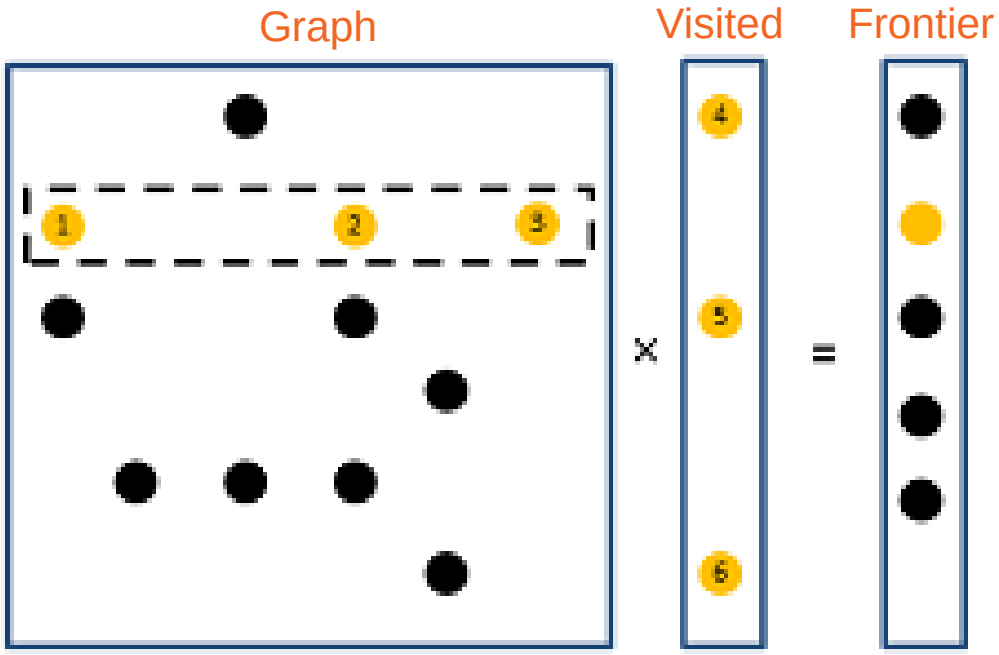
- A collection of OpenCL implementations of graph algorithms
- Sample algorithms include:
  - Betweenness Centrality
  - Single-Source Shortest Path (SSSP)
  - Maximal Independent Set
  - Graph Coloring
  - All-Pairs Shortest Path (Floyd-Warshall)
  - PageRank



On avg. the APU achieves 2.7x speedup over four CPU cores

BELRED

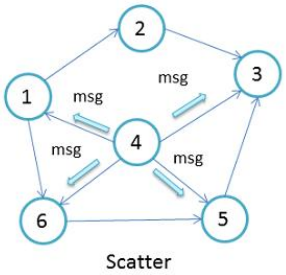
- Heterogeneous graph processing using sparse linear algebra building blocks
- Important routines include: *SpMV, min.+, SpGeMM, element-wise operations, segmented reduce, etc.*
  - Inspired by Kepner and Gilbert. *Graph Algorithms in the Language of Linear Algebra*



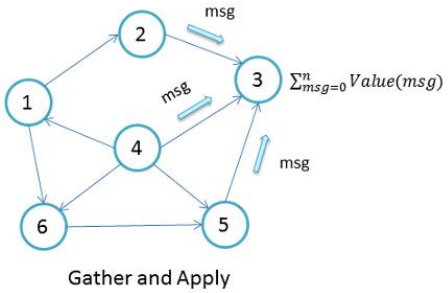
i[th] iteration of a BFS using SpMV

GASCL
LARGE-SCALE GRAPH SYSTEMS

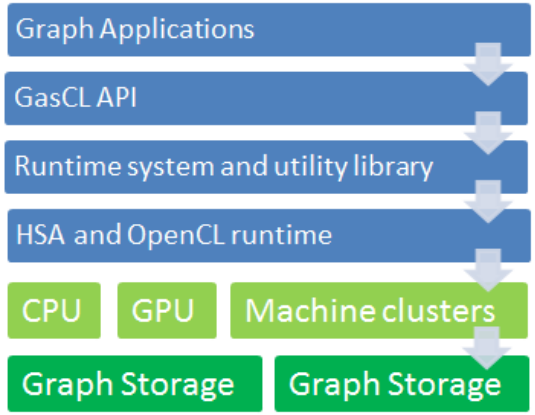
- Scale heterogeneous graph applications to multiple nodes
- System development using a model similar to GraphLab, Pregel, etc.
- Users develop graph applications in vertex-centric, *gather, apply* and *scatter* kernels

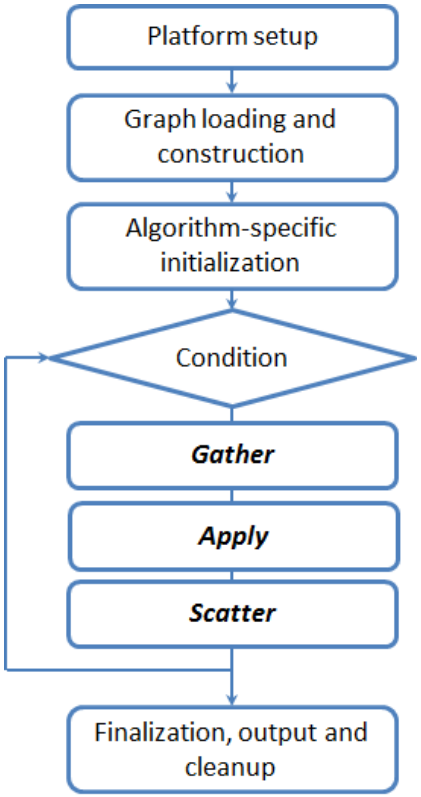


Scatter



Gather and Apply





CONCLUSION

AMD

- Large-scale data analytics and heterogeneous compute are the ~~future~~ **present**
- Heterogeneous computing can
  - Reduce power consumption in smart phones, tablets and the data center
  - Provide compelling new user experiences by improved performance
- AMD is pioneering the advancement of heterogeneous compute
  - Open standards
  - Easy to program
- Partner with us … Collaborate with us …
  - Machine learning
  - Graph processing
  - Deep neural networks
  - Open to other verticals

Mayank.Daga@amd.com **Mauricio.Breternitz@amd.com** **Junli.Gu@amd.com**

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

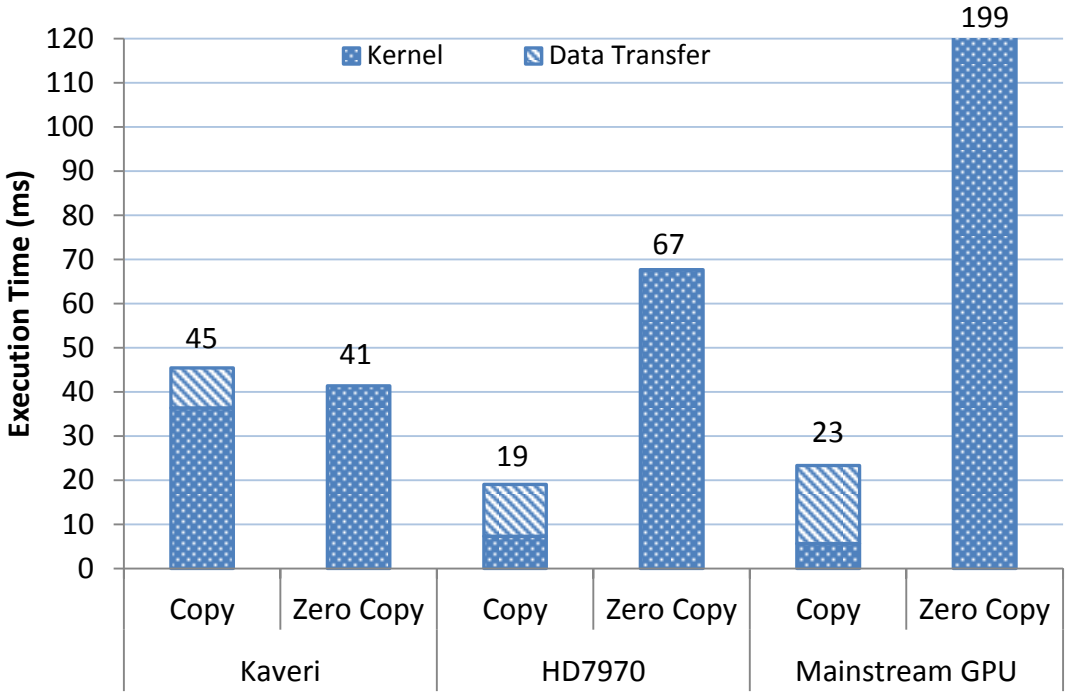
AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

WHY APUS - DATA TRANSFER OVERHEADS

- How to avoid data copy through the ***zero-copy*** technique on APUs?
  - APU: Zero-copy improves performance by 10%
  - GPUs: Zero-copy degrades performance by 3.5x for AMD GPU and 8.7x for Competitor GPU



Zero-copy technique:

GPUs: GPU accesses host memory through PCIe, slow

APUs: CPU and GPU share the same piece of memory, efficient

Experiment design:

CPU initializes 2k **x** 2k matrixes

(A, B), GPU performs C=A*B

Matrix multiplication performance comparison among copy and zero-copy