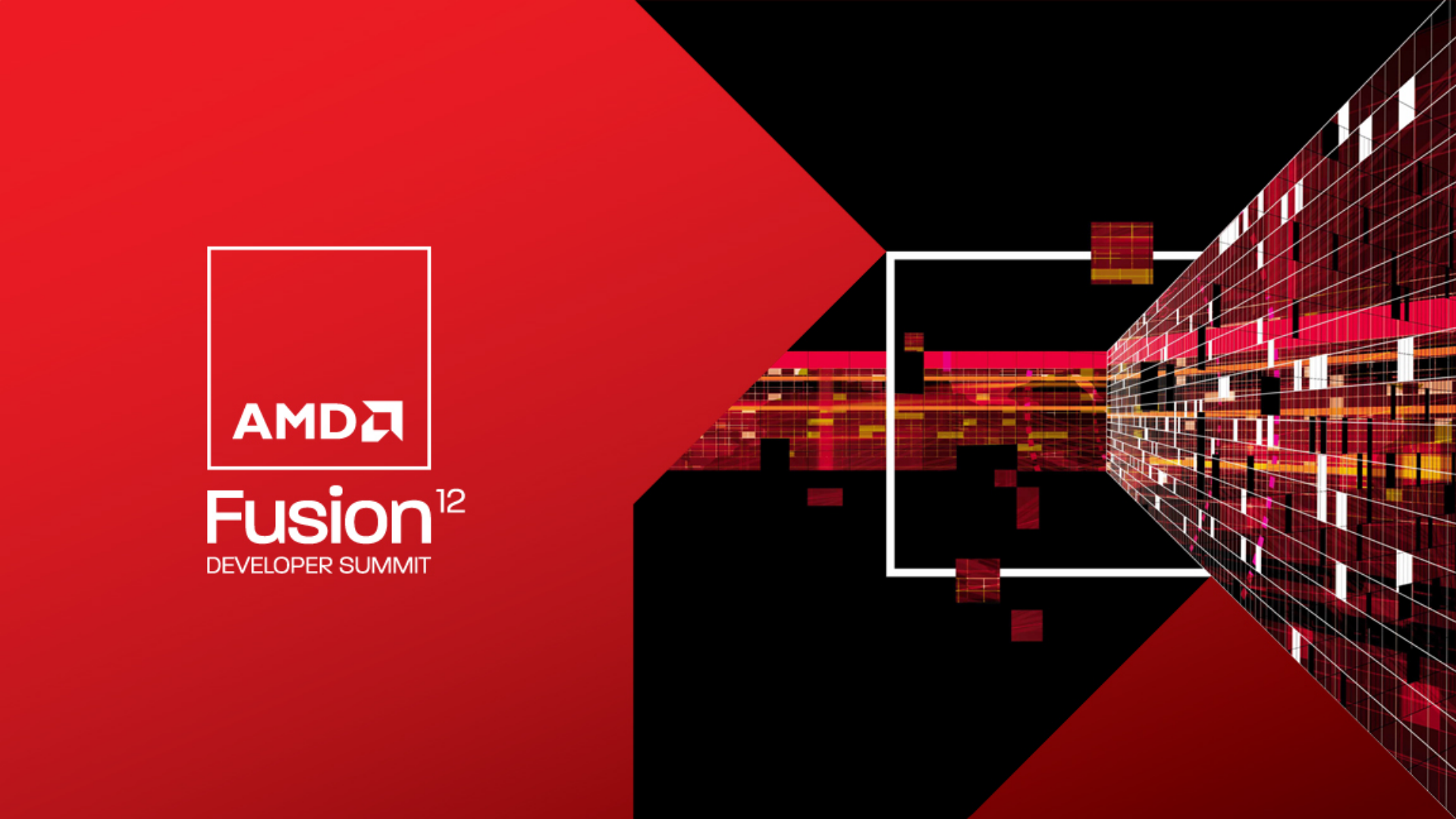

AMD
Fusion 12
DEVELOPER SUMMIT

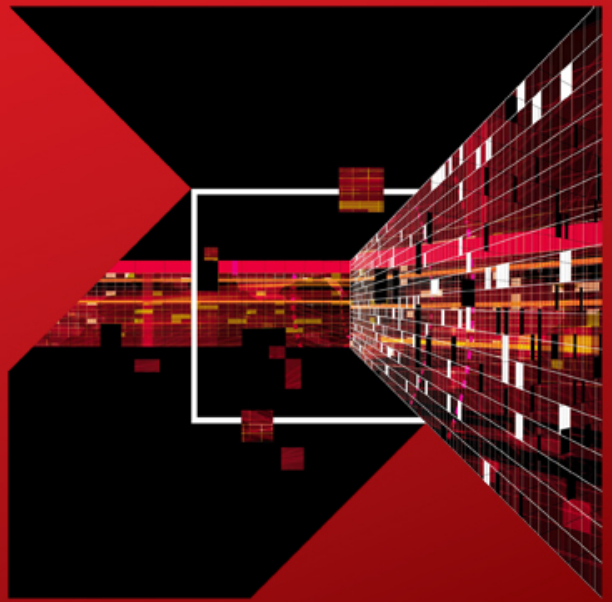
GPU ACCELERATION OF INTERACTIVE LARGE SCALE DATA ANALYTICS UTILIZING THE APARAPI FRAMEWORK

Ryan LaMothe
Pacific Northwest National Laboratory
Research Scientist
PNNL-SA-88181

Stuart Rose, Scott Dillard

GPU ACCELERATION OF INTERACTIVE LARGE SCALE DATA ANALYTICS UTILIZING THE APARAPI FRAMEWORK

CC-4257



AMD Fusion 12 DEVELOPER SUMMIT

CONTENTS

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

WHO IS PACIFIC NORTHWEST NATIONAL LABORATORY

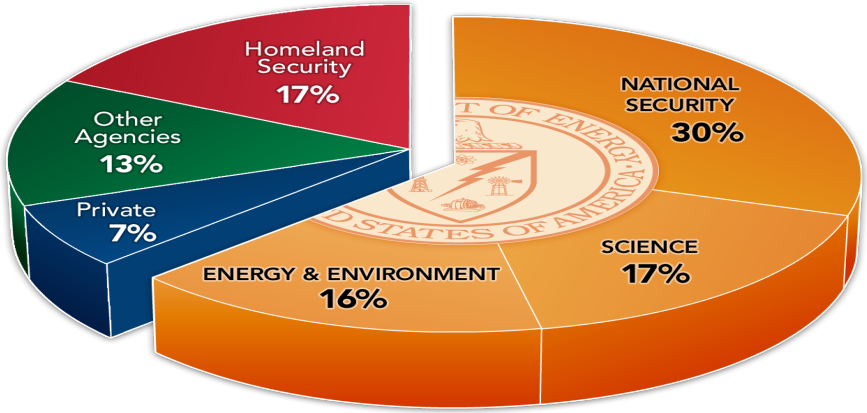
Our vision

PNNL will be recognized worldwide and valued nationally and regionally for our leadership in science and for rapidly translating discoveries into solutions for challenges in energy, the environment, and national security.

- Department of Energy (DOE) Office of Science Laboratory
- Battelle - the world's largest independent scientific research and technology development organization - has operated PNNL for DOE and its predecessors since 1965
- Outstanding science, impactful solutions

PACIFIC NORTHWEST NATIONAL LABORATORY AT A GLANCE

- Reports to DOE's Office of Science
- $1.1 billion* in R&D expenditures in FY09
- 4,700 staff
- 63% of funding is from DOE offices



THROUGH OUR MISSIONS, WE ARE ADVANCING SCIENCE AND TECHNOLOGY TO MAKE THE WORLD A BETTER PLACE

- Strengthen U.S. scientific foundations for innovation

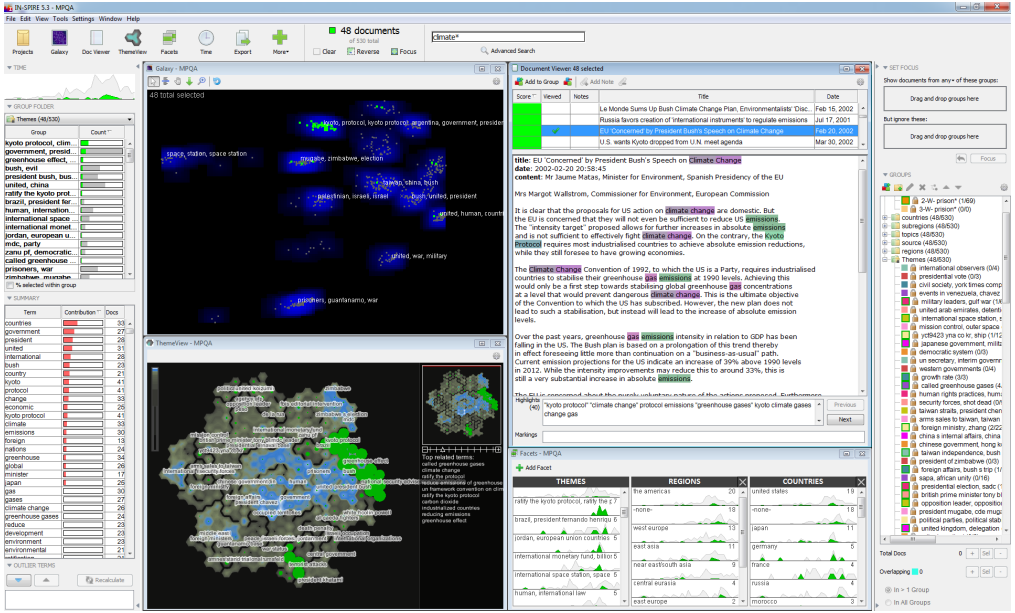
- Increase U.S. energy capacity and reduce dependence on imported oil

- Prevent and counter terrorism and proliferation of weapons of mass destruction

- Reduce environmental effects of human activity and create sustainable systems

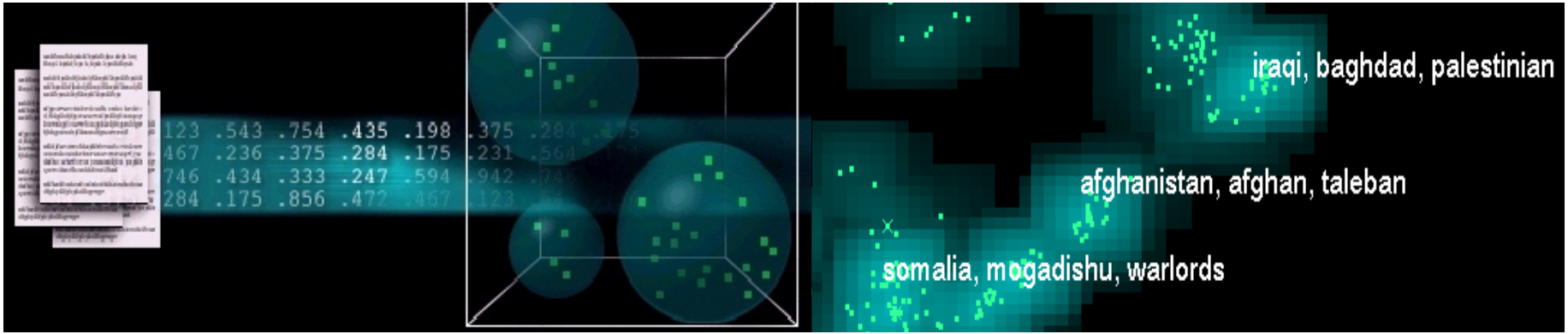
WHAT IS IN-SPIRE

- IN-SPIRE is an information visualization software system designed for the analysis of unstructured text data sources
- Key benefits and features
  - Text documents are automatically organized into topic clusters and visualized
  - Requires little or no a priori knowledge of the data sources
  - No up front training or key-wording
  - Language independent
  - Operates on commodity hardware/software
- A Thinking Aid
  - The user directs exploration and applies their interpretation
  - Helps the user see the expected and discover the unexpected



http://in-spire.pnnl.gov

TOPIC CLUSTERING



Extract Text from Documents

- *Create a mathematical signal (vector) for each document*

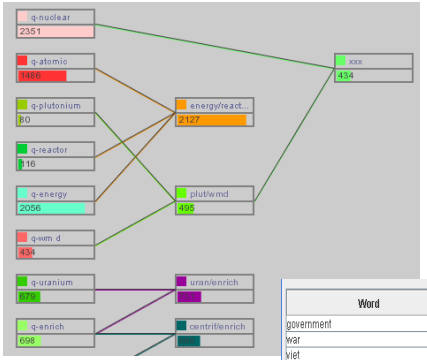
Organize According to Key Topics

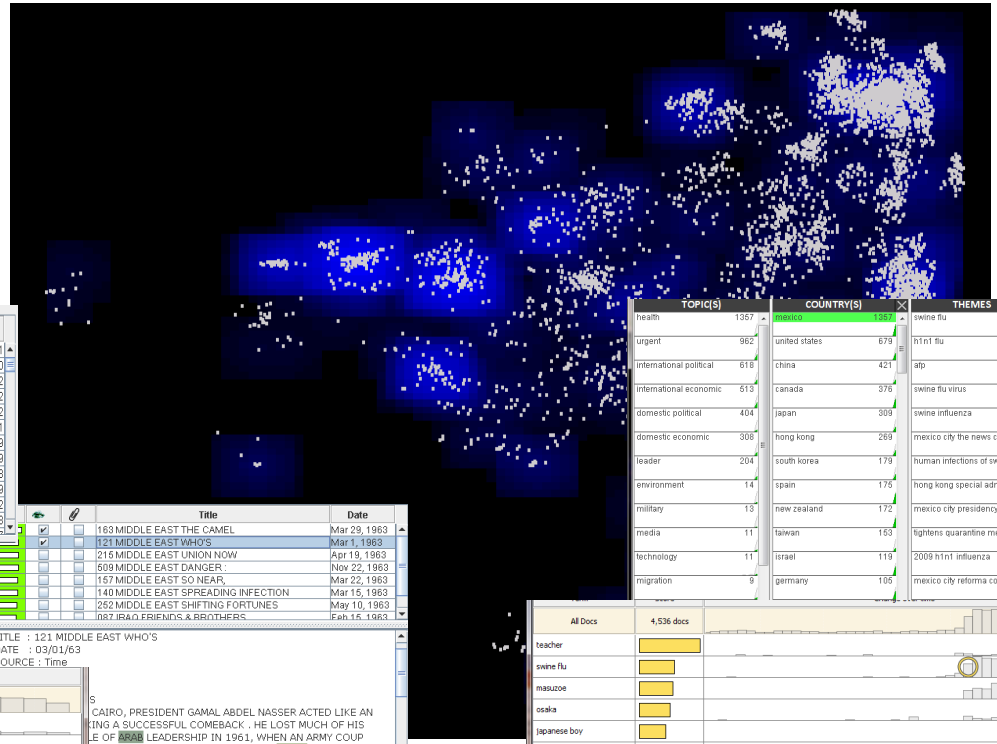
- *Cluster the document vectors in n-space*

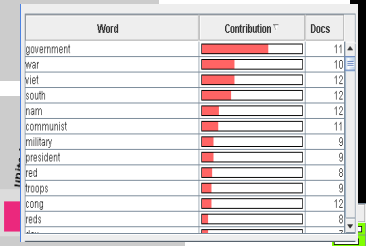
Present each document as a “docustar” where proximity suggests similar themes

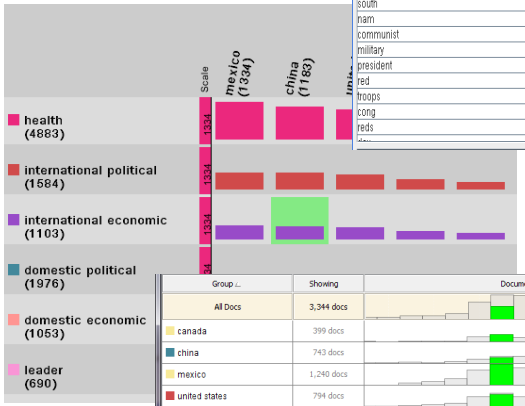
- *Project the n-space clusters into a two dimensional visualization*

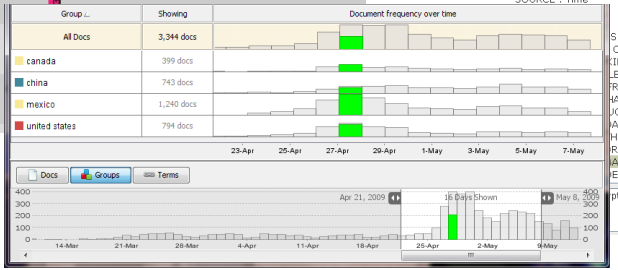
IN-SPIRE VISUALIZATIONS

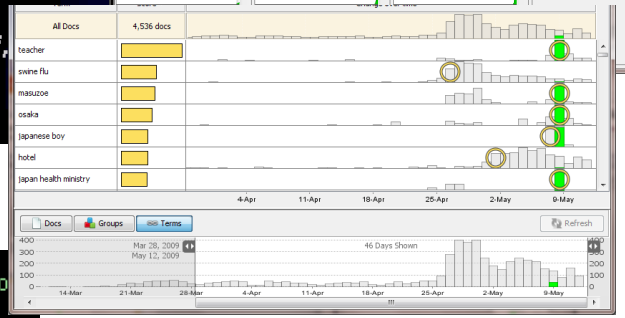












WHAT IS APARAPI

- Aparapi (A PARallel API) is a high-performance heterogeneous compute and data parallelization framework for Java that automatically converts Java bytecode to OpenCL at runtime for execution on multi-core CPU, GPU and APU devices
  - Open-sourced by AMD Inc. in 2011
  - http://code.google.com/p/aparapi/
  - Pacific Northwest National Laboratory is a contributor
- For most general-purpose use cases, developers can use Aparapi to convert algorithms that already describe parallel computations using for-loops into parallel computations that can be executed via OpenCL
  - Similar in concept to creating “parallel-for” loops
    - For-loops have to be converted to Kernels
    - Kernels use well-known Java thread semantics
  - Provides automatic failover to native Java Thread Pool when OpenCL is unavailable
  - GPUs require “share nothing” data parallelism and compute
    - “Advanced” users can leverage GPU “local” memory for special types of shared memory needs

WHAT PROBLEM ARE WE TRYING TO SOLVE WITH GPUS

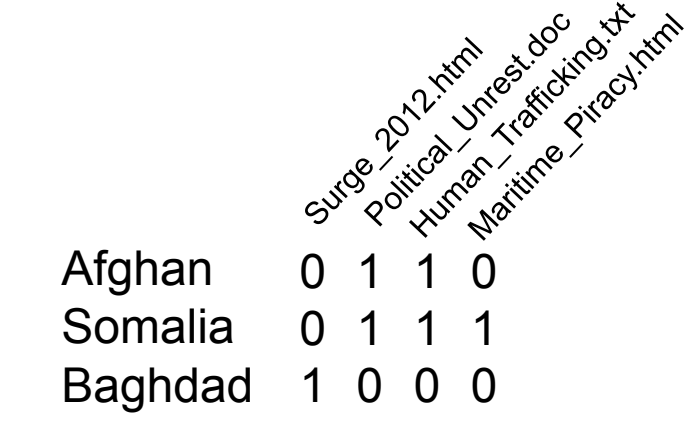
- We're trying to solve the problem of accelerating computation of large-scale term associations on arbitrary sets of documents
  - Dramatically decrease "time-to-solution" for analytics and visualization problems
  - Enable knowledge discovery through text analysis methods
  - Provide users with contextual insight into millions of documents in near-real time
  - Address the unique requirement that computations be performed locally on individual user workstations
    - Near real-time not possible with local CPU-only solutions
  - Support future server-side computation scaling requirements

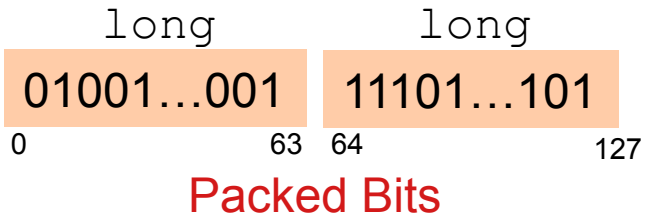
WHAT PROBLEM ARE WE TRYING TO SOLVE WITH GPUS

- Scaling of a fundamental computation used by many types of text analysis and set computation methods
  - Term association computations
    - Correlation/co-occurrence analysis
  - Drives the following features in IN-SPIRE
    - Vector space rankings
    - Principle component analysis
      - Latent semantic indexing
    - Text galaxy visualizations
    - Correlation tool
      - Document and terms
      - Document groups
    - Theme generation
    - Faceted browsing
    - Content searching and discovery
      - Finding important term associations

HOW ARE WE SOLVING THIS PROBLEM

- Term-document matrices
  - What is a term-document matrix
    - Rows = Terms
    - Columns = Documents
  - Matrix entry could be an existence bit, frequency count, etc.
    - Existence: Does term *t* appear in document *d*?
    - Frequency: How often does term *t* appear in document *d*?
  - For efficiency, bits are packed into longs
    - Efficiently stores 64 bits per matrix entry
  - One row of the matrix is effectively a bit vector
    - One Term existence bit set for all documents





HOW ARE WE SOLVING THIS PROBLEM

- Measurement of term associations is based on their co-occurrence within documents
  - Two terms are associated if they occur together frequently
- Intersection of bit vectors gives co-occurrence counts
  - Set intersection = Dot product of binary vectors
  - Extended to packed binary vectors using bitwise AND (`&`) and "population count" or popCount operation

Set Intersection = Dot Product

Afghan	0	1	1	0
Somalia	0	1	1	1
+	0	1	1	0 = 2

Dot Product of Packed Vectors

	01001…001	10100…011
	&	&
	11101…101	00000…001
popCount(	01001…001	00000…001)

HOW ARE WE SOLVING THIS PROBLEM

- Matrix multiplication naturally follows from dot products
  - Goal: Correlation Matrix C = A x B^T
    - Computes the matrix product of all-pairs intersections
    - Requires computing intersections of $2N^2$ elements
      - Computational complexity for A_{mxp} x B_{pxn} is O(mnp)
- In regular Matrix Multiplication, we must execute the following steps:
  - Multiple matrices A_{txd} x B_{txd}
    - Constraint: The number of documents (columns) in A must match the number of terms (rows) in B
  - In special cases, if Matrix B has incorrect dimensions, but will work if transposed, Matrix B is transposed
- Once we have Matrix A and Matrix B configured correctly, we perform a Dot Product on all Rows and Columns

HOW ARE WE SOLVING THIS PROBLEM

- How is our Matrix Multiplication different?
  - Packed elements
  - Modified matrix multiplication
  - Modified dot product
- We have the fortunate constraint that the number of documents has to be the same for both matrices
  - Required for algorithms using the result matrix
  - A_{txd} x B_{txd} where both matrices contain the same number of documents
- Therefore we can always make two assumptions:
  - One of the matrices will always need to be transposed
  - Transposition will always result in two matrices that abide by the rules of matrix multiplication
    - We will always perform A x B^T

HOW ARE WE SOLVING THIS PROBLEM

- Once we have A x B^T we proceed to the next step of matrix multiplication which is the transposition of Matrix B
- But wait, isn't A x $(B^T)^T$ just A x B?
  - Yes!
  - We do not need to perform the transpositions, avoiding very costly computations
- Next is the modified dot product
  - Normal dot product
    - $(A_{t1d1} \times B_{t1d1}) + (A_{t1d2} \times B_{t2d1})\ldots$
  - Modified dot product
    - We introduce two changes to this formula
    - $(A_{t1d1}$ **&** $B_{t1d1}) + (A_{t1d2}$ **&** $B_{\mathbf{t1d2}})\ldots$

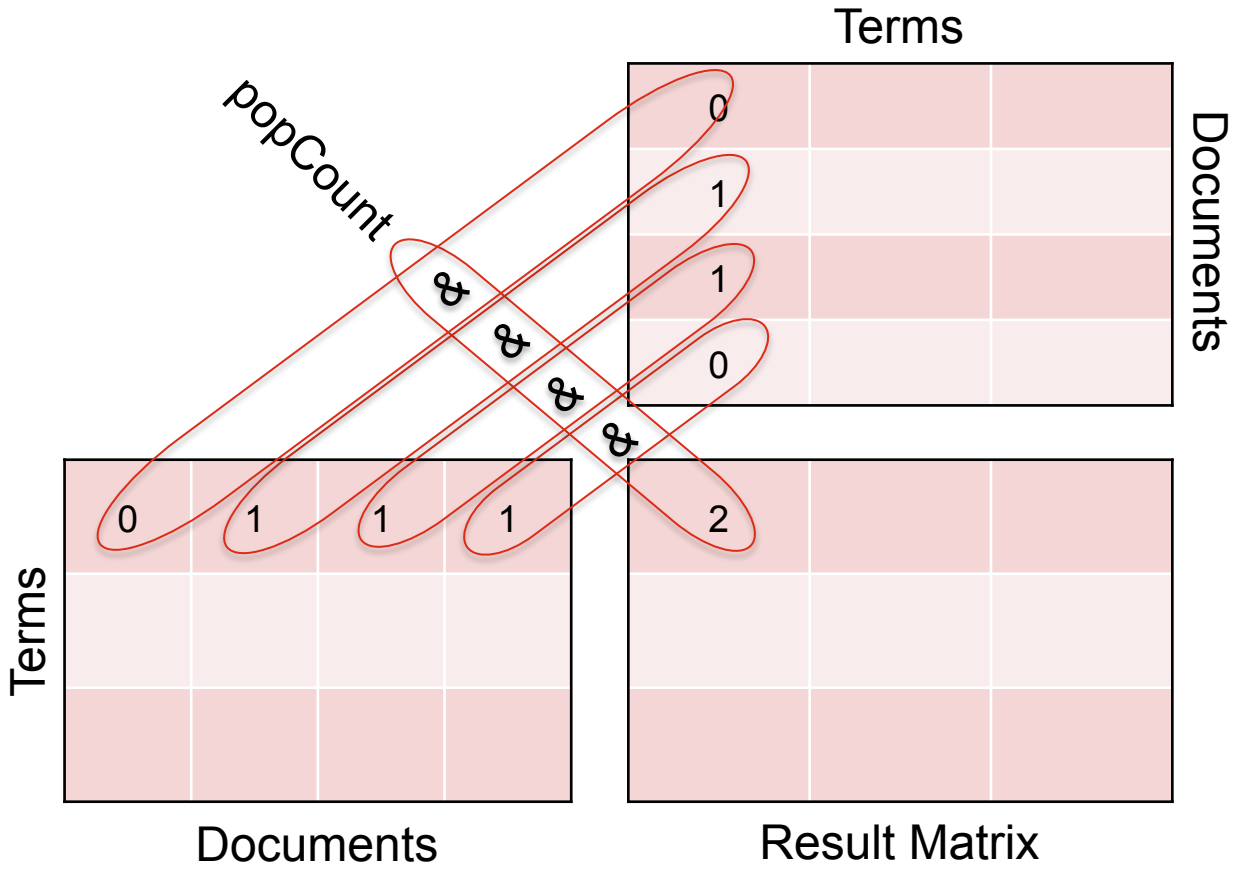
HOW ARE WE SOLVING THIS PROBLEM

- The two changes are the following:
  - We perform operations on only row elements of $A_{t1d(1...n)}$ and row elements of $B_{t1d(1...n)}$
  - Instead of multiplication we perform a bitwise AND (&) operator
    - All packed longs are unpacked before AND operator is applied
- What these changes allow:
  - We can perform a “population count” or “popCount” of bit intersections for each row across both matrices
  - We can treat all operations as “row major” ordering across both matrices
    - This allows us to perform another important optimization later
- What we end up with is a Term x Term dimension Correlation matrix of intersection counts

$$C_{ij} = \sum_{dk=1}^{n} A_{ti,dk} \;\&\; B_{tj,dk}$$

Where *t* and *d* refer to the term and document vectors, respectively, and & is a bit level operator

HOW ARE WE SOLVING THIS PROBLEM

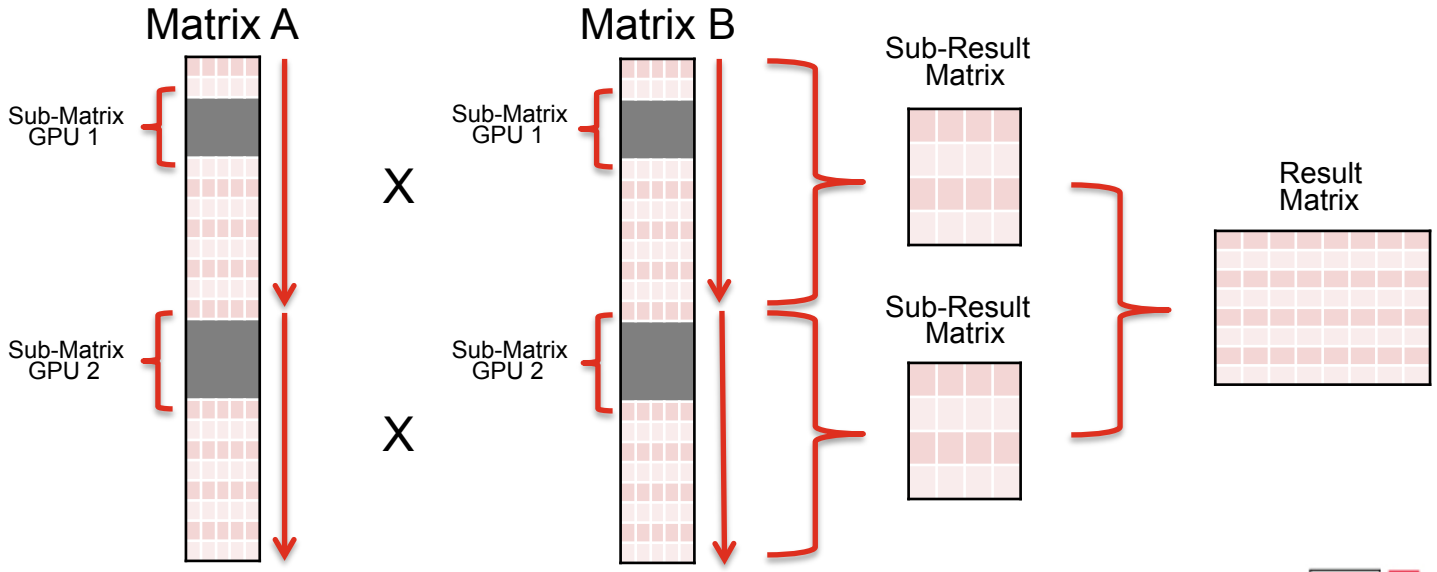


HOW ARE WE SCALING THE COMPUTATION

- Naïve implementation of matrix multiplication works great until you need to scale to large data sizes
  - Limited by memory available on GPU card
  - Minimum memory requirement is 2 x computation matrices + 1 x result matrix
- How do you scale to unlimited size term-document matrices?
  - At the most basic level we are performing vector-vector dot products on individual rows for entire matrices
  - This allows us to take a relatively simple approach to scaling
    - Perform dot products on multiple subsets of both matrices until entirety of both matrices have been covered
    - Only limitations are the maximum size of each sub-matrix and the sub-result matrix that can fit in the GPU card's memory

HOW ARE WE SCALING THE COMPUTATION

- There are numerous benefits to this approach
  - We can tailor the sub-matrix sizes to each GPU being used
  - We can distribute computation across multiple GPUs or CPU + GPU combinations
  - Final result matrix can be assembled on the CPU



HOW WAS APARAPI USED

- We initially tried two different approaches
  - Implement existing Lucene OpenBitSet “intersection count” algorithms using Aparapi
    - Executing Lucene OpenBitSet.intersectionCount in a double for-loop was used as the naïve base case
  - Implement traditional OpenCL-specific Matrix Multiplication algorithms modified for BitSets using Aparapi
- It turns out that implementing Lucene OpenBitSet algorithms in Aparapi was very easy
  - Kernel consisted of “copy and paste” with minor modifications to existing algorithms to support custom array access
    - Converted CPU-specific code to be GPU-optimized
  - Majority of research and development centered around optimizing the traversal of matrices in host code
  - When GPUs are unavailable, JTP fallback is still optimized for CPU
- The OpenCL-specific Matrix Multiplication code, modified to meet BitSet-specific requirements was difficult
  - Surprisingly, performance was overall slightly slower than OpenBitSet Kernel implementation in GPU mode
  - Unfortunately due to localBarrier usage in Kernel, algorithms did not perform acceptably when executed in JTP fallback

HOW WAS APARAPI USED

- Lucene OpenBitSet IntersectionCount (naïve)

```
for (int i = 0; i < obsList.size(); i++) {
  final Pair<OpenBitSet, OpenBitSet> obsA = obsList.get(i);

  for (int j = 0; j < obsList.size(); j++) {
    final Pair<OpenBitSet, OpenBitSet> obsB = obsList.get(j);

    final int truePositive = (int) OpenBitSet.intersectionCount(obsA.getLeft(),
          obsB.getRight());
    obsResultMatrix[i][j] = truePositive;
  }
}
```

```
public static long pop_intersect(long A[], long B[], int wordOffset, int numWords) {…}
```

```
public static int pop(long x) {…}
```

HOW WAS APARAPI USED

- Aparapi OpenBitSet IntersectionCount Kernel

```
@Override
public void run() {
  final int i = getGlobalId(0);

  if (i < matrixA_NumTerms) {
    final int j = getGlobalId(1);

    if (j < matrixB_NumTerms) {
      resultMatrix[i * matrixB_NumTerms + j] = pop_intersect(matrixA, i * numWords, matrixB,
          j * numWords, numWords);
    }
  }
}

private int pop_intersect(long matrixA[], int aStart, long matrixB[], int bStart, int
numWords) {…}

private int pop(long x) {…}
```

RESULTS

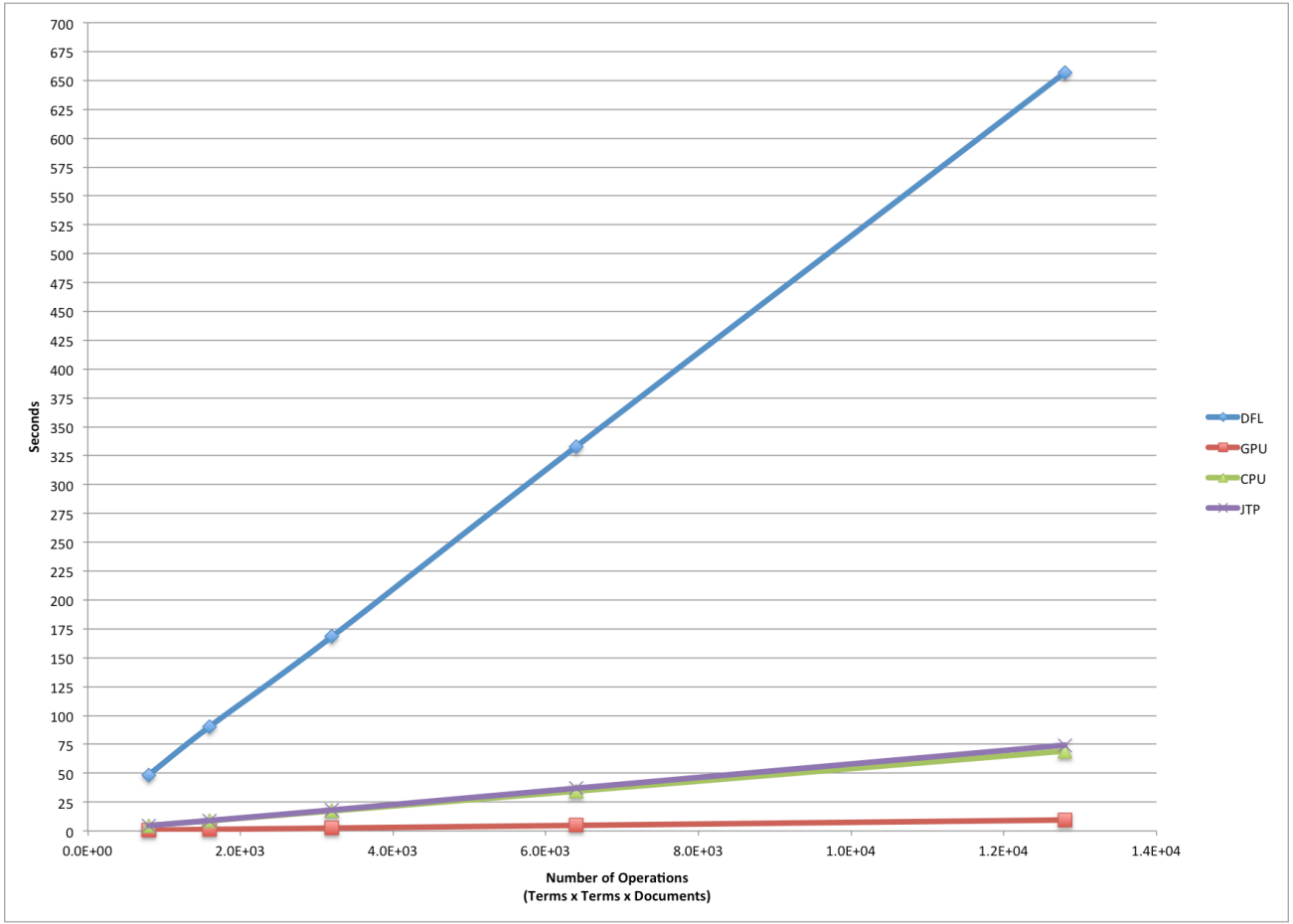
- A number of CPU vs. GPU comparisons were performed to gauge the time required to calculate various combinations of Term Document matrices
  - Fixed number of Terms, Varying number of Documents
  - Varying number of Terms, Fixed number of Documents
- Hardware Specifications

CPU	GPU
Dell 7500[1]	ATI Radeon HD 7970[2]
Windows 7 64-bit	Windows 7 64-bit
2 x 2.4Ghz Intel Xeon Quad Core (Gulftown/Westmere) 16 Effective Cores with Hyper-threading	32 compute units x 925Mhz (Tahiti) 2048 Stream Processors
PCI-E 2.0	PCI-E 3.0
12GB DDR3-1066	3GB GDDR5-1375

1. http://www.dell.com/downloads/global/products/precn/en/q2wk6_dell_precision_t7500_spec_sheet.pdf
2. http://www.amd.com/us/products/desktop/graphics/7000/7970/Pages/radeon-7970.aspx#3

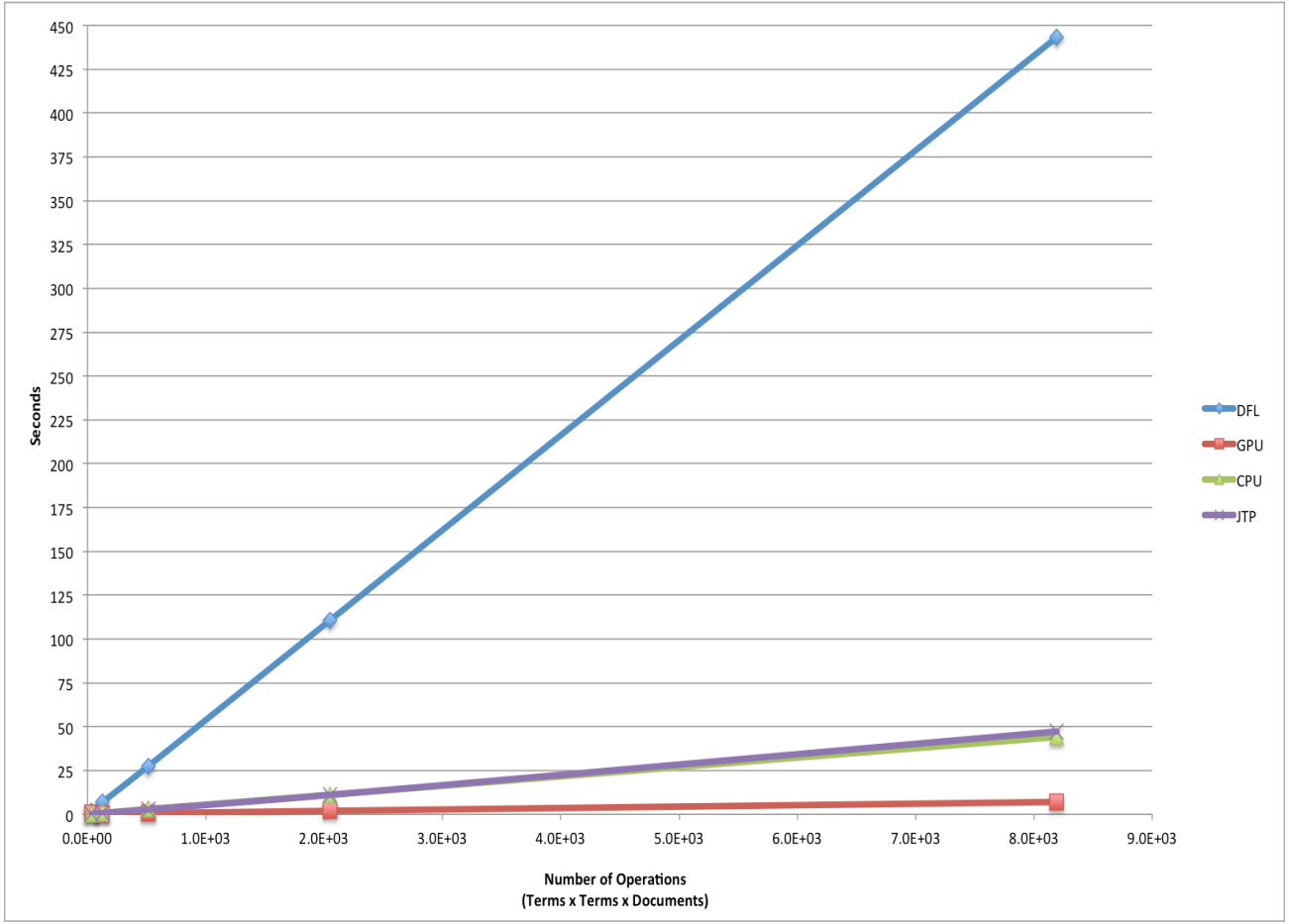
RESULTS

- Fixed Terms, Varying Documents
  - Results represent calculation time excluding host setup or OpenCL generation time
- Terms (Number of Rows)
  - 5000
- Documents (Number of Columns)
  - 32000…512000
- Number of Operations
  - 8.0E+11…1.3E+13
- Lucene OpenBitSet
  - Naïve implementation utilizing double for-loop (DFL)
- Aparapi
  - Java Thread Pool (JTP)
  - OpenCL CPU Mode (CPU)
  - OpenCL GPU Mode (GPU)
    - All implementations use OpenBitSet-based Kernel



RESULTS

- Varying Terms, Fixed Documents
  - Results represent calculation time excluding host setup or OpenCL generation time
- Terms (Number of Rows)
  - 500…8000
- Documents (Number of Columns)
  - 128000
- Number of Operations
  - 3.2E+10…8.2E+12
- Lucene OpenBitSet
  - Naïve implementation utilizing double for-loop (DFL)
- Aparapi
  - Java Thread Pool (JTP)
  - OpenCL CPU Mode (CPU)
  - OpenCL GPU Mode (GPU)
    - All implementations use OpenBitSet-based Kernel



SUMMARY

- Current IN-SPIRE Correlation Matrix CPU-specific computation was having difficulty meeting desired performance goals
  - Time-to-solution
  - Large matrix dimensions
- Conversion of existing CPU-specific code to GPU code can be extremely difficult
  - Research revealed that Correlation Matrix computation is GPU-friendly
  - Aparapi made GPU conversion of CPU code very easy
- At the smallest matrix dimensions, performance between JTP, CPU and GPU was very similar
- As matrix dimensions increased
  - JTP and CPU performance decreased at a similar rate, quickly becoming unacceptable
    - OpenCL CPU mode performance was nearly identical to native Java Thread Pool mode
  - GPU performance decreased at a significantly lower rate, remaining acceptable
- Clear path forward to allow nearly unlimited size matrix performance scaling
- Desired IN-SPIRE performance goals for both JTP and GPU modes will be met

QUESTIONS?

- Contact Information

Ryan LaMothe
Research Scientist
Computational and Statistical Analysis
Pacific Northwest National Laboratory
ryan.lamothe@pnnl.gov
www.pnnl.gov

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. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD's positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.