APPLIED RESEARCH LABORATORY FOR
INTELLIGENCE AND SECURITY

OPERATIONALIZING EMERGING HARDWARE FOR AI APPLICATIONS: A SURVEY OF TRANSITION OPPORTUNITIES AND DATASETS

So what for datasets:
- KG.
- PB Scale.
- Need Diversity because of Data-centrism of performance.
- Needs to cover:
  - Transaction (Financial) data.
  - Person-to-person comms
  - Social Media.
  - DEIJSE relationships
  - Dense Labeling of entities.

OGP Data on Wildfire?

Version: 2022-08-31

Lead Author: William C Regli, UMD/ARLIS
regli@umd.edu

Contributors: Peter Loats, UMD/ARLIS
Laurel Miller-Sims, UMD/ARLIS
Ben Johnson, Jataware
Jacob Bunker, UMD/ARLIS
Jared Ott, UMD/ARLIS
Christine Herlihy, UMD/ARLIS
And insights from colleagues at STR and elsewhere.

So What: for Benchmarks
- GRAPHs U common, ; are usually KG's.
- Streaming data more relevant than static for most.
- Time constraint processing is a common occurance.
- Real data isn't available, but we can produce Synthetic Scale replicas of most.

So what for KG Ops:
- Add / Remove. (STREAMING)
- Search ⇒ BFS Traversal (Node traversal)
- Path Finding ⇒ OSPF (Node traversal)
- Pattern Matching.
- Time Constraint?
- Co-travelers ... Pattern Matching?

Background

Recent and emerging investments in microelectronics has produced a wide variety of new technologies that offer to change the processing pipelines associated with Artificial Intelligence applications. Just as onboard floating point came to the early personal computers in the 1980s to the transformation created by the mathematical processing capabilities of graphics processing units (GPUs) since the 2000s, new chipsets and hardware paradigms are enabling the creation of new approaches to the design of intelligent systems for signal processing, large data analytics, and resource-constrained computation among other areas. This resurgence in the capability and the diversity of microelectronics is creating new opportunities to re-engineer our approaches to the design and implementation of intelligent systems. This requires a new process for hardware-software-application co-design that rethinks the relationship between software and the functional primitives enabled by the underlying hardware. Additionally, it requires those that run computationally intensive applications (AI/ML, data science, graph analytics, etc) to rethink their underlying computational infrastructure as compelling gains in performance or size weight and power (SWaP) may be possible. The rapid adoption of emergent hardware requires re-engineering operational systems to account for the new technologies and new processes by which to transition these technologies into use.

Document Purpose and Approach

This document captures the inputs and results of an ongoing survey of operational users from across the Department of Defense and Intelligence Community regarding their current uses of computational hardware to solve large-scale artificial intelligence problems. This survey was conducted while providing transition support for the DARPA Software Defined Hardware (SDH) and Hierarchical Identify Verify Exploit (HIVE) Programs conducted under the Autonomy Application and Engineering Exploratorium (A2E2) project at the Applied Research Laboratory for Intelligence and Security (ARLIS). The focus of this survey has been to identify specific use-cases from current missions that push the limits of what can be done with the current generations of computational infrastructure for AI. The survey was conducted over many months in a series of informal and formally structured meetings and interviews. In some of the cases, government partners provided access to specific (sanitized) datasets for our experimental inspection. In other cases, both broad and deep descriptions were provided of specific datasets and operational needs; these descriptions were used to develop abstractions of the problem for use in our analysis.

AT END OF DOC > See if these are answered. end of Doc comments in RED

In each case, the team attempted to ascertain:

- What is the nature of the dataset(s)? Can we determine the makeup of the datasets at the core of the operational need and what datatypes do they consist of? For example, are datasets large text corpora or visual media? Are the fundamental structures large graphs or video files? How are they organized? How often do they change?

  > Heterogeneous, But LOTS of Graphs. Almost all graphs are KGs

- What are the use cases of these datasets? The use cases help us assess what the canonical queries or analytic activities are on the datasets. For example, are we looking for patterns of concern (e.g., fraud) in a set of banking transactions? Are we trying to perform facial identification over a dataset of images or videos?

  > ITS

Determining the key use cases can enable a user-centered re-evaluation of the processing workflow in the light of new hardware or software technologies.

- Are there any specific constraints on these use cases? Is real-time processing needed (e.g., flight controls for an aircraft)? Are there bounds on time allowed? Are there a human operator in-the-loop or are these use cases more abstract?

→ Time constrained.

- Are there quality and accuracy constraints on these use cases? What's the acceptable amount of error or allowed failure rate, if any?

→ Not really aware, but in general for I+S → Very Accurate.

- To process the queries for these use cases, what computational tasks are executed on these datasets? For example, image segmentation and matching may involve different mathematical operations and computational kernels from natural language translation, which may also be different from finding topological embeddings on graphs. What is the computational bottleneck in the processing pipeline and what algorithmic activities are the key to exploiting these datasets? What are the main mathematical operations being conducted (e.g., dense or sparse linear algebra? Graph traversal? String matching or fast Fourier transforms? Convolutions and transformers or combinatorial optimizations?)?

Graphs:
- Update
- Search
- Shortest path
- Comm. det
- Sim.

- What is the current processing architecture being used? Is this a customized compute resource or a general-purpose cloud resource? Or are processing pipelines special-purpose (e.g., a sensor processing pipeline for a radar system)? What hardware technologies are currently used (i.e., CPU, GPU, FPGA, etc)? Where does computation occur (i.e., "on the edge" vs in a data center)?

→ Not Really Concerned.

- What relevance and availability is there for proxy or synthetic data for these problems? Datasets such as Graph500, ImageNet, etc have provided contexts for vast amounts of academic and applied research---are there any such datasets for the problems at hand? Could such datasets be created? What are the limits on the use of synthetic data?

→ Nothing @ Scale,

Some stat similarities obvious.

While these questions do not necessarily create a complete picture---nor do all questions apply in every case---they provide a common rubric with which to consider operational needs and possible transition opportunities. The product of this survey enabled the team to create a framework with which we can describe the key commonalities and technical issues across these applications. In doing so, we hope to identify the main technical challenges in obtaining transition of emerging technologies into operational use and recommend actions and measures the community might undertake to hasten the insertion of new hardware into wickedly hard AI pipelines.

Summary of Findings

We believe that this survey was the first of its kind conducted. Most of the organizations spoken with were deeply operational---working with live systems that conduct warfighter support or law enforcement functions. While technologically literate and astute, their situation and operational tempo provided them with little opportunity to reflect on how changes in their computational infrastructures could produce improved performance or disruptively new capabilities. Key findings include:

- **Operational data is hard to obtain and share:** A key goal of A2E2 was to obtain datasets from operational users and employ these datasets in test and evaluation activities to aid in the transition of the emerging hardware. We found it was largely not practical to isolate operationally relevant datasets for such experimental use. This for a variety of reasons:
    - *Size:* Some of the organizations' datasets are very large scale (> 1 petabyte), which makes sharing a copy of the dataset logistically difficult.
    - *Velocity:* Data velocity is a key part of the operational use case. By velocity we mean the rate of change in the data and that the data is constantly changing. Hence a static snapshot may not be relevant for evaluation as operationally relevant workloads operate on data *streams* rather than static datasets.
    - *Security:* Classification levels of the datasets create numerous challenges. Procedurally, data co-use agreements take months-to-years to put in place, making it difficult for any facility (even an appropriately accredited one) to get data to a shared location in a timely manner. In a couple instances, rules associated with data created other unique problems. In one case there was a time constraint where operational data could only be used within 24 hours of collection; in another, any computer hardware that "touched" the data would have had to be destroyed after use.
- **Organizations have adopted (or are adopting) cloud or cloud-like approaches to their compute fabric.** Organizations we spoke with maintain operational systems and their computing architectures are "locked in" due to a combination of system design decisions (some of which are years old) and acquisition strategies. Such organizations do not have much flexibility to experiment with emerging hardware and/or novel accelerators.
- **The operational systems perform many costly computational functions, not all of which are comprised of SDH/HIVE-relevant workloads.** The operators we spoke with maintain systems that support a wide variety of use cases. In some cases only a subset of these use cases were possible candidates for acceleration by SDH/HIVE hardware. This means that - though a portion of their workload could be accelerated by SDH/HIVE

hardware - overall systems "cost" (time, energy, etc) would remain bound by other parts of the pipeline.

- **The specific nature of the query makes a big difference.** While this is to be expected, the team identified this in nearly every transition use case. The operational system may have many uses and the central problem in transition was to identify the specific set of queries to the dataset that could benefit from hardware-based acceleration. The conclusion was that not all queries would equally benefit from the hardware acceleration technologies. Same idea as Quantum.

- **The specific nature of data objects makes a big difference.** In other cases the processing pipeline may include mathematical functions that are candidates for hardware acceleration, but the nature of the dataset itself might not lend itself to benefit from the acceleration. For example, one of the hardware platforms under development on SDH specifically targets operations on matrices that are a mix of both sparse and dense. If the data is not of that character, the acceleration will not necessarily yield much benefit.

Need: Diverse Datasets.

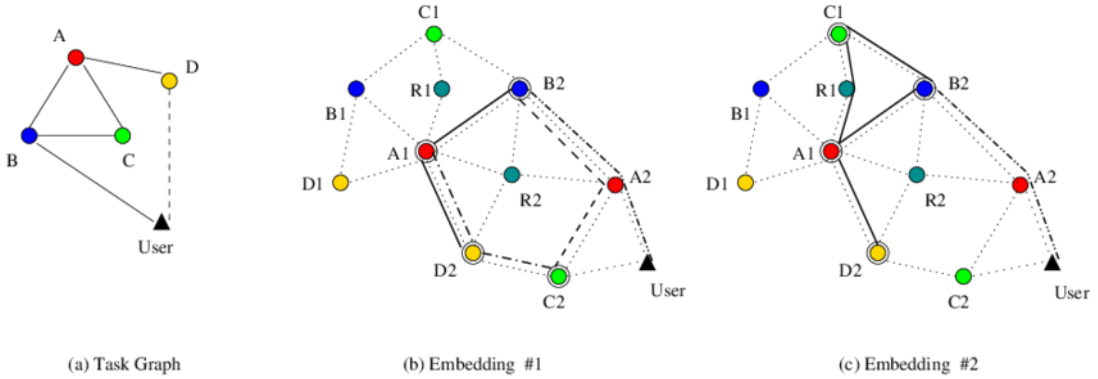
- **Different users are bottlenecked by different aspects of the AI/ML lifecycle:** Some of the people we spoke to cared most about improving the speed of their AI/ML training process, while others are focused on increasing the inference throughput of an already-trained model. The question of whether training vs inference performance is "more important" varies *widely* by organization and individual.

- **Many graph processing pipelines were target real-time applications.** The need to query a dynamic, rapidly changing graph in real time was encountered in several instances.

  - **Three principal types of queries that appear frequently in graph analytics applications that query large graph datasets.** The first of these is "is A connected to B?", which amounts to finding the shortest path, or all paths, between two nodes in a graphical database. The second query is of the nature "Show me what is going on with A?", which amounts to a breadth-first search around the neighborhood of node A in an effort to assess the family of associations with node A. The third of which is a pattern-matching query, usually associated with activity recognition, that amounts to subgraph matching or subgraph embedding in a larger graph. All are well-known algorithmic problems that, with the exception of shortest path, are worst-case exponential in nature depending on the properties of the graph and its search space).

Query types

PRIMITIVES



(a) Task Graph (b) Embedding #1 (c) Embedding #2

Figure: From https://www.researchgate.net/figure/Example-of-Task-Graph-Embedding_fig1_260384855, illustrating a canonical example of graph embedding.

- **Information systems and processing architectures would need to be redesigned to fully leverage emerging hardware technologies.** The operational problems and case studies did not present obvious "drop in" approaches by which the emerging hardware acceleration capabilities can be readily applied. Some degree of refactoring the systems approaches to these problems appeared to be required in nearly every case. Additionally, we speculate that if hardware could disruptively accelerate key functional bottlenecks (i.e., orders-of-magnitude improvement in graph traversals or matrix computations) one might make significant changes to the overall algorithmic approach rather than just improve a local operation. For example, one might leverage new hardware to add new algorithmic functionality where previously it was too computationally costly.

These principal findings indicate that there remains a "valley of death" across which new hardware or hardware/software co-design solutions need to cross in order to be placed into operational use. Unlike situations in fast-moving industry markets---situations in which new technologies are commercial discriminators and rapidly deployed to discriminate products---the DoD and IC systems that could most benefit from the adoption of new hardware paradigms are often enterprise systems with significant investment in their architectural choices. Further, these systems cannot have their operations paused for experimental purposes. Crossing this valley of death and seamlessly inserting new hardware technologies will take new kinds of process and software innovation.

Summary of Case Studies

This section provides a summary of the engagements with the organizations surveyed during this project. In each case, the team worked to identify one or more operationally-relevant problems that were candidates for the hardware emerging from the SDH/HIVE efforts. For each problem, we ascertained the nature and diversity of the datasets in question, the nature of the queries and interrogations that were occurring over these datasets, and the computational processes that were consuming the most time and energy in conducting these queries. We present these as a series of case studies and in each we try to identify:

- The nature of the dataset(s) of relevance to the problem;
- The critical queries or algorithmic processing on these datasets;
- The relevant computational workflows behind those queries;
- The nature of the computational costs in executing the queries;
- The applicability of proxy or synthetic data for experimentation; and,
- Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings.

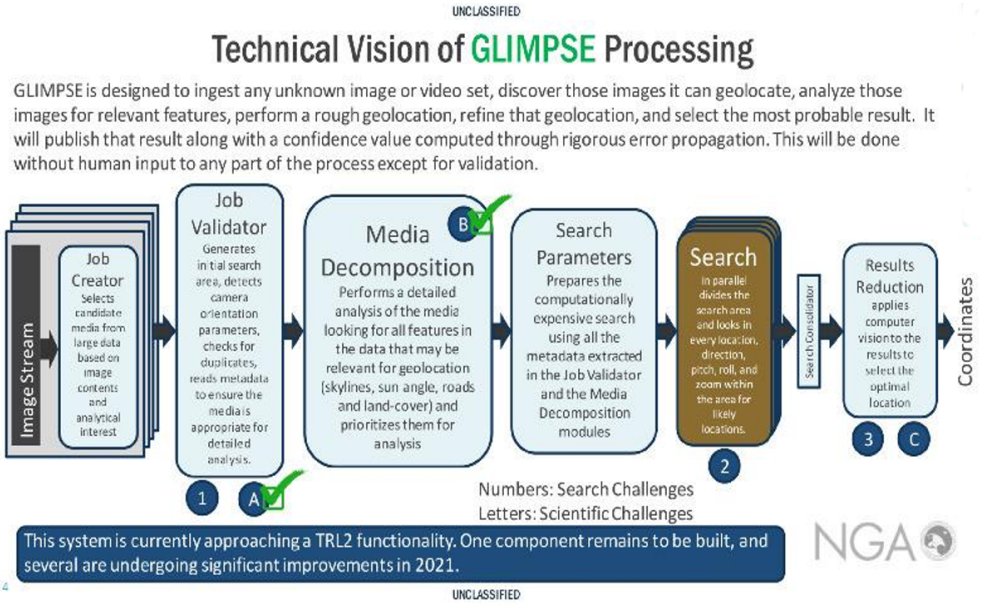


Figure: A slide from NGA outlining the overall GLIMPSE framework.

Case Study: GLIMPSE – Horizon Matching

The National Geospatial-Intelligence Agency (NGA)'s GLIMPSE is a spatial analytics program that serves a number of different operational scenarios. The principal scenario considered in this case study was the problem of horizon matching: given an image or a set of images, geolocate these images based on an analysis of the skyline and horizon.

- *The nature of the dataset(s) of relevance to the problem*: Datasets are collections of geolocated visual images and extraction metadata (such as a histogram describing the skyline). Datasets are global in nature, with locations on the planet being associated with multiple skyline images.

- *The critical queries or algorithmic processing on these datasets*: The key query on this dataset is to match a representation extracted from the horizon line in the query image to candidate images in a geotagged database. This is a form of nearest neighbor query based on a distance computation against the organized set of reference objects in the database.

- *The relevant computational workflows behind those queries*: On of the main algorithmic methods used in matching is a form of histogram matching. For example, one way of computing this would be as an earth-mover's distance (EMD) over the histogram extracted from the skyline.

- *The nature of the computational costs in executing the queries*: The primary computational cost is in execution of the database query over database entries.

- *The applicability of proxy or synthetic data for experimentation*: Synthetic data is not applicable in this situation because the computational costs are specific to searching across the global database of geotagged images. The scale and diversity of this global database is central to the computational problem and this dataset and its properties are not easily approximated. Basic experimentation could be done using a database for histogram matching and EMD with synthetic data.

- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: There are several aspects of the computational pipeline for GLIMPSE that could be hardware accelerated, specifically the use of histogram matching via techniques like earth-mover's distance.

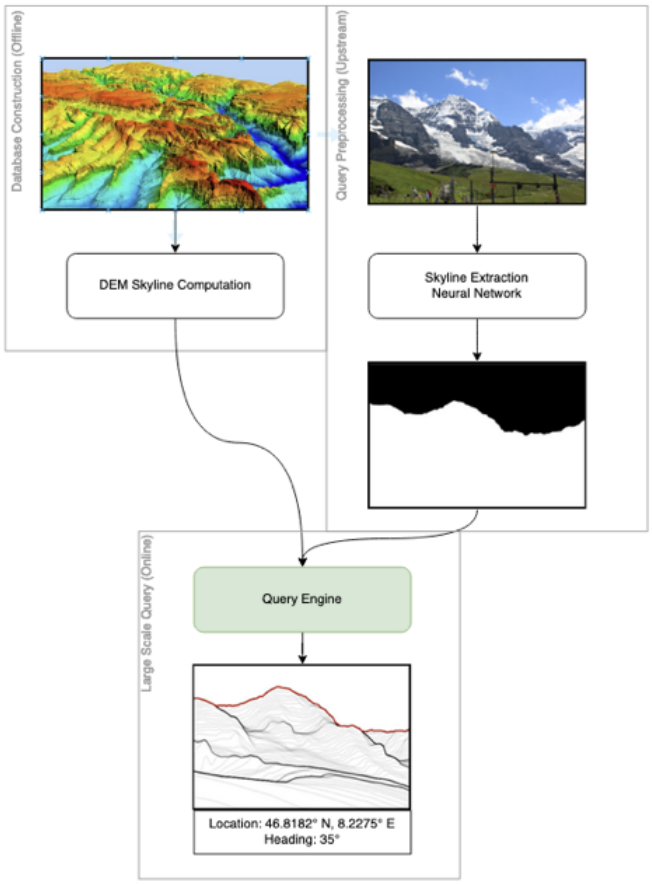


Figure: An example processing workflow associated with horizon matching.

Case Study: GLIMPSE - Skyline Extraction

A significant sub-component of the GLIMPSE horizon matching pipeline is extracting skylines from an image.

- *The nature of the dataset(s) of relevance to the problem*: Datasets are a collection of images with unknown geolocation.
- *The critical queries or algorithmic processing on these datasets*: Each image must be processed to produce a vector representing the angular height of the skyline samples at regular intervals across the camera's horizontal field of view.
- *The relevant computational workflows behind those queries*: This workflow uses a Convolutional Neural Network to classify pixels as "skyline" or "not-skyline". A secondary processing step (such as morphological erosion) is used to reduce the result to a line.

- *The nature of the computational costs in executing the queries*: The primary computational cost is classifying pixels as skyline or not-skyline via the neural network model.

- *The applicability of proxy or synthetic data for experimentation*: Images including skylines are widely available. Adequately testing the robustness of the technique requires the use of a variety of terrains and weather conditions, such as cloudy or clear skies, and foreground objects (plants, buildings, people).

- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: Anything that accelerates Convolutional Neural Networks should accelerate skyline extraction. Hardware acceleration of Convolutional Neural Networks is already a high-profile target in private industry.

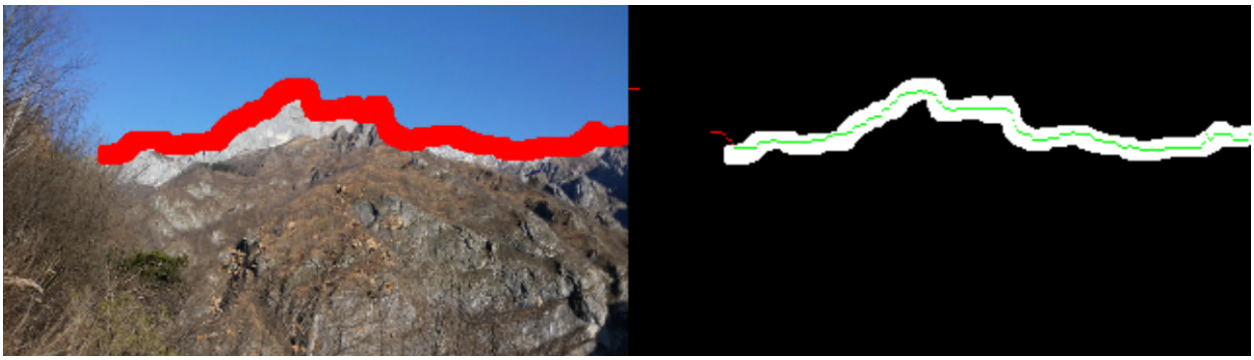


Figure: An example of generating a vector (green line) from an image containing a skyline.

Case Study: Graph Analytics Test Resource

The Graph Analytics Test Resource is an experimental program being developed by the US Air Force looking at large scale graph datasets (PB in scale) and their use for analytic products to support a variety of mission users and operators across the DoD and IC. These datasets are principally transactional information (i.e., financial transactions) captured as a set of entities (nodes) and relationships (edges), all of which would be labeled with textual and semantic information about the nature of these entities and relationships.

- *The nature of the dataset(s) of relevance to the problem*: Large-scale graph data that captures transactions among entities and their relationship. Information in the graph is tagged to include the named entities, geo-location information, the nature of their relationships, temporal information, etc. Graph scale is PB in size, with billions of entities and trillions of edges, and it updated in real-time as new information comes into the network.

- *The critical queries or algorithmic processing on these datasets*: There are several candidate queries in this scenario, including template graph matching (setting a database

trigger to look for candidate activities) or graph search (find me entities in the graph with certain properties or relationships).

- *The relevant computational workflows behind those queries*: Computational workloads are those associated with graph operations in the queries, including subgraph matching or graph search.

- *The nature of the computational costs in executing the queries*: Each query, while it could be improved in the average case and with heuristics, is potentially an operation that is of exponential cost in the size of the query graph.

- *The applicability of proxy or synthetic data for experimentation:* While the scale of the datasets is hard to reproduce, the statistical properties of the dataset can be used to create synthetic data with similar characteristics.

Is there a point where the scale is not?

- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: Intel's PIUMA architecture enables hardware accelerated graph traversals in a way that could be applied to pattern matching within larger graphs. The graph embedding and subgraph isomorphism problems incur considerable computational costs when conducting traversals of node adjacencies. Hence, new hardware could enhance one of the core bottlenecks for finding patterns of interest in large graphs.

How to generate graph embeddings?

Case Study: RAVEN

The Department of Homeland Security's (DHS) Homeland Security Investigations (HSI) Innovation Laboratory operates a number of advanced projects, including RAVEN. HSI is the lead development organization for RAVEN ("Repository for Analytics in a Virtualized Environment"), an analytic platform for DHS. RAVEN is a platform for data integration and analysis across a number of specific data feeds of relevant to DHS missions such as port security, fiscal transactions, movement of people and goods, etc. At the core of RAVEN is a graph analytics engine that performs vital functions to provide active support to law enforcement and active investigations.

- *The nature of the dataset(s) of relevance to the problem*: RAVEN includes a large (PB) scale graph database that is used to integrate data from a wide variety of sources. The graph structure is used as the main framework for linking these diverse datasets in order to make unified queries.

- *The critical queries or algorithmic processing on these datasets*: RAVEN provided detailed information about the nature of their queries, which take two main forms. First, to support an investigative inquiry, given a subject of investigation, find the network of associated objects with the subject. This amounts to a breadth-first-search-like traversal

of the graph started at the node(s) representing the subject of investigation. The second form of query is to determine if two entities in the network are related and, if so, how. This amounts to a path finding problem.

- *The relevant computational workflows behind those queries*: The first query, BFS, may be exponential in the size of the graph; the second is a shortest path algorithm.
- *The nature of the computational costs in executing the queries*: Computational costs are incurred in the execution of these algorithms for each query. In large graphs, the database lookups required to traverse the graph structure could require substantial memory access times.
- *The applicability of proxy or synthetic data for experimentation*; It is very difficult to develop an accurate statistical model for this use case due to the extensive and important nature of the tags and attributes on the graph. In this application, while the graph is central, it is really the integration framework that brings all of the semantic data together from the other sources.
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: Hardware assessments could be done about the generic queries, such as BFS in a large, attributed graph. Intel's PIUMA platform has demonstrated strong performance on standard large-scale benchmarks (e.g. Graph500).

Figure: An illustration of image segmentation, object recognition and tracking (left); and the process of mathematical convolution, a computation that is central to modern deep learning systems.

Case Study: Video Analytics and Object Detection in Video

Among the use cases for the technologies emerging from Project MAVEN are those aimed to enhance the processing of video streams.

- *The nature of the dataset(s) of relevance to the problem*: Large volumes of full-motion video (FMV) as well as wide area EO/IR and SAR datasets are generated from many airborne platforms requiring analysis – ideally in near-real time.
- *The critical queries or algorithmic processing on these datasets:* Algorithmic processing in this case study is for identifying and classifying objects of interest, for single and multiple targets in a frame as well as object tracking over time. Storing detections in relational and graph databases allows for both tracking as well as time and geo-based queries.
- *The relevant computational workflows behind those queries:* Convolutional Neural Networks, Transformers and other deep learning techniques for image processing. Neural networks and/or multi-hypothesis trackers for target tracking.
- *The nature of the computational costs in executing the queries:* Costs are those associated with deep learning processing of large volumes of data. Query costs are associated with traversing graph databases and relational datasets.
- *The applicability of proxy or synthetic data for experimentation*; Operational data is not accessible for MAVEN, but there are numerous open corpora for video and object detection that could be suitable.
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: Accelerating neural network processing would seem to have high potential for improvements given both the volume and usefulness of the data. Assessing capability to improve edge performance would be desired given to improve delivery times of results and decrease performance cost at the network's edge (e.g., on an aerial platform).

Case Study: Supply Chain Analytics

The team has been in discussion with a number of organizations that wish to create integrated datasets to support supply chain analysis and risk assessment. These organizations include the Office of the Director of National Intelligence's Supply Chain and Cyber Directorate (SCD), OUSD(R&E), and OUSD(I&S). Over the course of this project, the team has been party to several studies conducted under other sponsorship that have informed this case.

- *The nature of the dataset(s) of relevance to the problem*: Datasets relevant to supply chain analytics are highly heterogeneous but usually center on financial and corporate

data related to organizations and activities in a supply web. The scale of these datasets was not notable.

- *The critical queries or algorithmic processing on these datasets*: In the case studies surveyed, these datasets are used to support investigative activities. Hence, queries are driven by the need to look for patterns of risk or activity in the supply network that could prose a vulnerably to the system.

graph matching.

- *The relevant computational workflows behind those queries*: Computational needs in these cases were confined to traditional database operations.
- *The nature of the computational costs in executing the queries*: N/A.
- *The applicability of proxy or synthetic data for experimentation:* It would be very difficult to reproduced real-world use cases using proxy data in this example.
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings:* N/A.

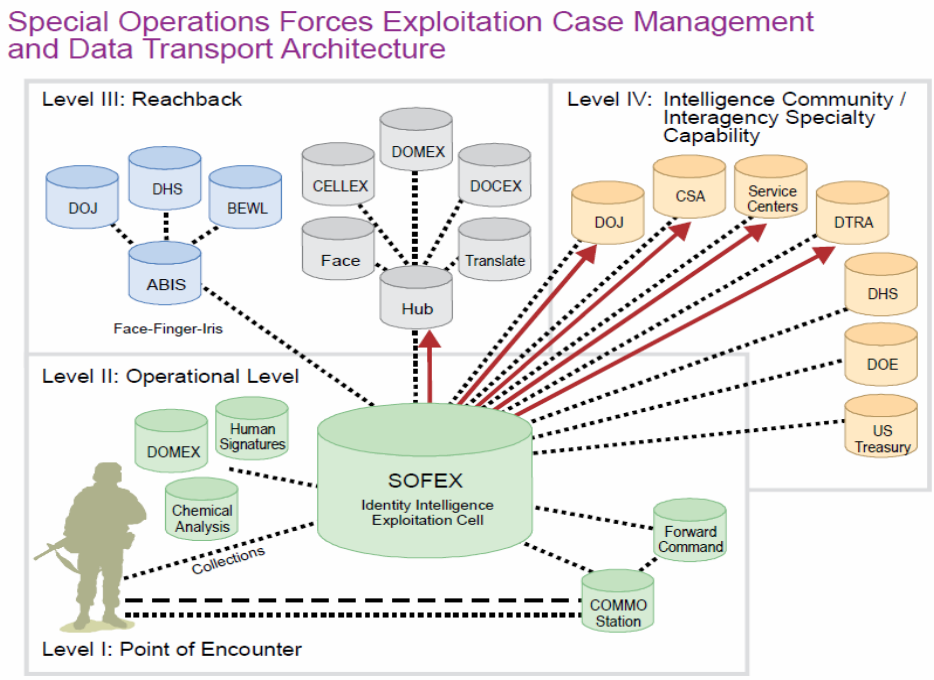


Figure: SOFEX architecture – illustrating reachback for Collected Exploitable Materials processing (CELLEX, DOMEX, DOCEX etc.).

Case Study: Collected Exploitable Materials (CEM)

- *The nature of the dataset(s) of relevance to the problem*; CEM includes a wide variety of physical and digital material, but the OUSDI&S effort will focus on Digital and Multi-media (D/MM) and Document and Media Exploitation (DOMEX) data specifically, to include cell phones, hard drives, GPS devices, flash drives, and other digital media along with a wide array of file types to include text, image, video, and audio files. D/MM and DOMEX data is complex, being in a variety of types, formats, and languages, and in substantial volume, challenging analysts' ability to characterize holdings and identify relevant information in a timely manner

- *The critical queries or algorithmic processing on these datasets*;

    - Bulk processing of CEM files will occur during initial ingest when CEM files are analyzed for specific information that will help tie the files and contents to a specific digital identity that is being built. Speech-to-text and machine translation may be applied before natural language processing algorithms extract named entities. Optical character recognition (OCR) will be applied to text documents and text occurring in images and video on the devices being processed. The extracted entities and text will be populated into a knowledge graph to assist in triage and retrieval

    - Critical queries are typically made of the knowledge graph to help identify a digital identity of interest as well as refine, merge and deduplicate entities within the graph. Associations between identities will traverse the graph based on common locations or specific attributes of the identities or shared interactions (such as cell phone conversations or text message)

- *The relevant computational workflows behind those queries*;

    - After extraction from a device, the following technologies are used to extract attributes of a digital identity:

        - Production of text from non-text sources

            - Speech-to-text and speech detection to identify and extract text content from voice data in audio and video files.

            - Optical Character Recognition (OCR) of both images of documents and images of text within video and pictures

            - Machine Translation from source language to English

        - Natural Language Processing of resultant text

        - NLP algorithms leaning heavily on Named Entity Recognition both in English and trained on source languages.

    - During triage and analysis Knowledge Graphs are analyzed to filter and extract digital identities of interest.

- *The nature of the computational costs in executing the queries*;

    - Costs of running NLP tasks for the production of and processing of text is associated with neural network algorithms like Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) - among other techniques - to extract and classify text from exploitable materials. The volume of materials and calculations suggests use of server-class computing hardware. However, customers have also expressed a desire for local compute capabilities in order to get immediate feedback from a specific collection.

    - Triage and analysis of results costs are focused on database queries from relational, graph and NoSQL databases.

- *The applicability of proxy or synthetic data for experimentation*; proxy data is not necessary for this application. While data sets are large, CEM data is considered UNCLASSIFIED until analyzed and thus is accessible.

- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: Heavy use of NLP techniques both server-side and in edge computing suggest that AI acceleration is something that should be considered as CEM programs are enhanced.

Case Study: Personnel Datasets

The team held discussions with the Defense Counterintelligence and Security Agency (DCSA) regarding the possibility of applying AI/ML technologies to improve the process of personnel vetting and the granting of security clearances. This problem---posed as a study and not as an operational question---would aim to perform pattern and language analysis on personnel records such as found in SF86 and e-QIP information, as well as associated public and social media data. Given known problematic personnel, could AI/ML be used to identify any predictive patterns that could come from the personnel information?

- *The nature of the dataset(s) of relevance to the problem*: Data could include personnel records as well as scraped social media data.

GRAPH STRUCTURE

- *The critical queries or algorithmic processing on these datasets*: In this scenario, there are no specific queries as the computational costs would be related to ML training.

- *The relevant computational workflows behind those queries*: Computational workflows would be related to processing images, text and media associated with the personnel information (i.e., NLP, computer vision pipelines, etc) as well as those related to training classifier systems based on the feature information extracted.
- *The nature of the computational costs in executing the queries*: N/A
- *The applicability of proxy or synthetic data for experimentation*: N/A
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: N/

Shit loads
Also I have Many GB of CUI LEM.

Case Study: Multi-Target Tracking

The team studied a DARPA program focused on the persistent tracking of multiple targets. Most relevant to this case study is one of its central problems: the computationally challenging task of multi-hypothesis tracking of many objects in a scene or environment. Sensors could be still image-based, video, satellite images, or other forms of input and temporal scales could be real-time or over other time intervals.

- *The nature of the dataset(s) of relevance to the problem*: Fused multi-sensor data providing indicators on multiple objects in a cluttered, noise-filled, environment.
- *The critical queries or algorithmic processing on these datasets*: Queries are related to geo-location of objects in the environment at a current or future time.
- *The relevant computational workflows behind those queries*: The computational costs are those related to the filtering, tracking or statistical processing needed to manage the object tracking. For example, a Markov-Chain Monte Carlo simulation would potentially involve many simulations for each object, executed over many runs, executed for all objects.
- *The nature of the computational costs in executing the queries*: Queries are product of the hypothesis tracking system, hence the computational costs mostly occur elsewhere.
- *The applicability of proxy or synthetic data for experimentation*: Synthetic data would not be practical in this situation because the nature of the problem is so operationally specific. The computational costs in the MHT could be simulated in some generic sense.
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: Testing MHT and algorithms such as Markov-Chain Monte Carlo were areas of original interest for SDH technologies.

Case Study: Enterprise Classified Information Management

The team held discussions with the organizations within the Office of the Under Secretary of Defense for Intelligence and Security focused on the processing and release of previously classified materials. The research question posed was to examine AI tools for document processing and analysis to see if they can assist in this process.

- *The nature of the dataset(s) of relevance to the problem*: Datasets in question would be mostly textual information in reports and technical documents, although it is assumed that documents could include figures, illustrations and technical drawings as well.
- *The critical queries or algorithmic processing on these datasets*: Processing would not be queries, rather it would be text and image processing needed to make sense of the documents themselves and perform tasks such as "autotagging" and redaction.
- *The relevant computational workflows behind those queries*: Workloads in this scenario would be NLP-related activities.
- *The nature of the computational costs in executing the queries*: At the initial high-level that this problem was considered, costs would be those associated with running NLP-type analyses over many documents---with the scale and number of documents being the main source of costs.
- *The applicability of proxy or synthetic data for experimentation*: N/A
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of finding*: Data sources such as those discussed in this case would be similar to those found in repositories that are maintained by the Defense Technical Information Center (DTIC). There are numerous available datasets (DoD, commercial, and public), with which one could test NLP systems at scale.

Case Study: Logistics and Optimization for US TRANSCOM

United States Transportation Command (TRANSCOM) is the unified combatant command that focuses on the global logistics network supporting the Department of Defense. US TRANSCOM is tasked with coordinating the movement of people, machines, and sustainment-related resources around the world to support US defense activities during both peace and wartime. US TRANSCOM leverages both public and private sector transportation resources to complete its mission.

The domain-level input components of TRANSCOM supply chain problems reflect these priorities, and include: (1) physical locations (e.g. military bases, supply depots, airports, seaports, military targets); (2) transportation assets, represented at varying levels of granularity (e.g., individual cars, trucks, trains, tanks, planes, and/or boats, along with asset groups); (3) requirements to be transported (e.g., people, transportation assets, supplies, etc.); (4) transportation infrastructure; and (5) metadata/constraints associated with the aforementioned

components, which may include information about counts, cost, time, distance, energy, risk, sequence/dependency, availability/feasibility, and/or preference/rank. With respect to orders of magnitude, the number of physical locations in a given scenario is of order hundreds and the number of items to be transported is of order thousands.

TRANSCOM scenarios can be formally represented as constrained combinatorial optimization problems that, when solved, provide decision-makers with domain-level information about the feasibility, and/or optimality, of the resource allocation/supply chain configuration problem(s) under consideration.

- *The nature of the dataset(s) of relevance to the problem*: US TRANSCOM does not have "datasets" per-se, rather they have scenarios. The size and scale of these scenarios is noted above (hundreds of locations, thousands of entities).
- *The critical queries or algorithmic processing on these datasets*: The main computational activity on these datasets takes the form of combinatorial optimization, with the current solver framing this as a mixed integer programing problem.
- *The relevant computational workflows behind those queries*: Matrix computations related to mixed integer programming and related optimization techniques.
- *The nature of the computational costs in executing the queries*: N/A.
- *The applicability of proxy or synthetic data for experimentation*; The problem of logistics and scheduling has been subject of study for many years and synthetic data is widely accepted.
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: N/A.

While the TRANSCOM problem are complex and computationally expensive, the algorithms were not obviously directly related to the hardware acceleration technologies of HIVE and SDH.

Case Study: Media Analytics for the AFRL Sensors Directorate

The USAF ARFL Sensors Directorate maintains repositories of sensor data for use in testing new AI algorithms to support sensor processing, target tracking, and video analytics. These datasets include, for example, media streams from UAVs consisting of hours of video that needs to be processed, condensed, labeled etc. For this data to be useful, human review of the data is required and software systems are needed to pre-process the data in order to optimize the allocation of the attention of analysts to the most salient elements of the most relevant scenes.

- *The nature of the dataset(s) of relevance to the problem*: Many thousands of hours of overhead sensor data taken from operational platforms.

- *The critical queries or algorithmic processing on these datasets*: One of the challenge problems would be that of autotagging the video streams using previously-trained machine vision systems to look for patterns of interest and flagging them.
- *The relevant computational workflows behind those queries*: Computationally, tagging video streams implies constant execution of multiple video analytics systems (multiple neural nets for example) that process frame-by-frame calculations over the video data.
- *The nature of the computational costs in executing the queries*: Costs come from the mathematical operations needed to process pixels at frame rates, perform the needed transformations, etc.
- *The applicability of proxy or synthetic data for experimentation*: There are many corpora of video that could be used as proxy data.
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: Several of the SDH's computational workloads could be useful in accelerating the execution of a video analytics pipeline.

Case Study: NSA Y Division

NSA Y Division presented us with a unique case study in which large amounts of sensor data are aggregated for the purpose of performing entity resolution and identifying patterns of activity that require investigation. In their framework, the data was to be assembled into a graph structure capturing entities and relationships. The unique aspect of this case was that the assembly of the dataset and the execution of the queries had to be done in a highly time-constrained manner as the data had an expiration timebound. This presents a unique "anytime" or "contract" algorithm scenario for computing over a graph.

TIME constraint IS weird.

SIMULATE:
① Time to conflict
② Data expiry time.
→ Create a graph that "deletes" data after a given time period.

- *The nature of the dataset(s) of relevance to the problem*: The dataset is a large (millions-to-billions of entities) graph holding information about entities and their relationships.
- *The critical queries or algorithmic processing on these datasets*: Entity resolution and pattern matching in graphs.
- *The relevant computational workflows behind those queries*: The entity resolution problem is known to be NP-hard, hence performing just that task across a large data graph is computationally intensive. The added challenge of performing a "best of effort" approach within a time bound is quite unique.
- *The nature of the computational costs in executing the queries*: See above.
- *The applicability of proxy or synthetic data for experimentation*: This is similar to many real-world problems in which it is hard to reproduce a graph database to mirror the real

world scenario---at best one can build synthetic data based on graph statistics and invariants.

Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings: The time-bounded nature of this problem could be a fertile area for investigation and would benefit from hardware acceleration.

Imp-driven Recommendation.

Case Study: Financial Pattern Analysis

The team considered one operational dataset that contained financial transactional data. The data was a time-varying entity-relationship graph. This dataset was very similar to the kinds that would be relevant to fraud detection as currently done by financial services companies

- *The nature of the dataset(s) of relevance to the problem*: Dataset captured transactional information (the entity, the financial organization, nature of the transaction, etc), represented graphically (nodes and edges).
- *The critical queries or algorithmic processing on these datasets*: Queries on this dataset look for patterns of activity and resemble subgraph embedding and subgraph isomorphism problems.
- *The relevant computational workflows behind those queries*: Subgraph matching and isomorphism is a well-known NP-hard computational problem with a number of existing heuristic techniques that make it addressable in certain cases.
- *The nature of the computational costs in executing the queries*: Computational costs depend on the nature of the algorithm for solving the subgraph matching problem. There are techniques that are combinatorial (based on discrete graph algorithms) as well as those that are based on matrix mathematics (i.e., spectral methods).
- *The applicability of proxy or synthetic data for experimentation*: While synthetic data could be produced in such a way that the statistical properties are based on the operational dataset it would be difficult to produce query dynamics that mirror what is operationally relevant and extremely difficult to create a dataset of the appropriate scale.
- *Any recommendations for analysis to assess the potential for hardware acceleration as well as any predictions of findings*: This case further indicates the need for a comprehensive approach to assessing graph matching across a variety of algorithmic techniques could have practical impact.

Other Cases Examined and Considered

In addition to the operational datasets, the team had several other scenarios that it considered that were not formally tied to discussions with transition organizations. These are briefly covered here.

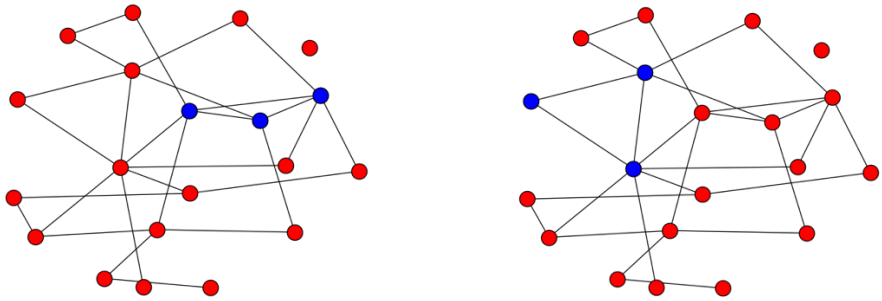


Figure: From https://louridas.github.io/rwa/assignments/finding-trusses/, illustrating 3-trusses in couple example graphs.

NSA R-Division: The Laboratory for Physical Sciences

The team interacted with members of the technical staff at the Laboratory for Physical Sciences, a division of NSA's Research organization. Their interests were in unclassified graph processing workloads including breadth-first search, triangle counting, Jaccard, PageRank, Giga-updates per second (GUPS), computation of connected components and k-trusses. For example, a k-truss in a graph is a subset of the graph such that every edge in the subset is supported by at least k−2 other edges that form triangles with that particular edge. An example of a truss is shown above.

PRIMITIVE
ADD / DELETE

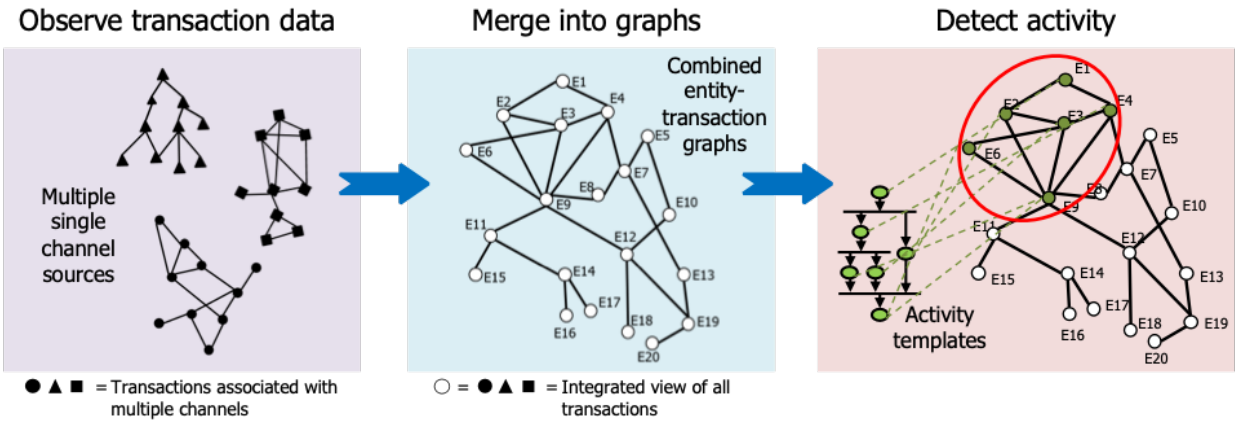


Figure: Example from the DARPA MAA Program of finding patterns of activities in a labeled graph.

Modeling Adversarial Activity (DARPA MAA)

The DARPA Modeling Adversarial Activity program was developed to create algorithmic approaches to identify patterns of suspect behavior. The original conception of the program was to develop graph-based approaches for integrating sensor observations and pattern-analysis algorithms implemented as variations on graph matching techniques. The program sponsored a variety of researchers and the team examined approaches being developed at the University of

Maryland Department of Mathematics as well as at UCLA. These technique were advancing new forms of heuristic approaches based on probabilistic algorithms and mathematical transformations.

In the context of hardware technologies emerging in HIVE/SDH, the algorithmic developments in MAA were mostly theoretical at the time of this study and not ready for empirical assessment at scale on operational datasets. Techniques for which implementations did exist, in particular from researchers at UMD and UCLA, were at a level of complexity beyond the scope that could be experimented with using the hardware emulation technologies available for HIVE/SDH.

DARPA's SIGMA+ Program

The SIGMA+ Program is developing an integrated approach for comprehensive sensing aimed to avert terrorist threats posed by chemical, biological, radiological or nuclear devices (CBRNE). In discussions with this program it was determined that significant efforts were invested in sensor modeling, sensor development and the creation of an integrated sensing framework that could integrate all modalities to produce improved situational awareness.

In the context of the emerging AI and graph processing hardware under study, the SIGMA+ "back end" was a cloud-based system for sensor fusion and creation of a common operating picture. There may be applicability for emerging hardware techniques, such as those from HIVE and SDH, to support improved cloud analytics.

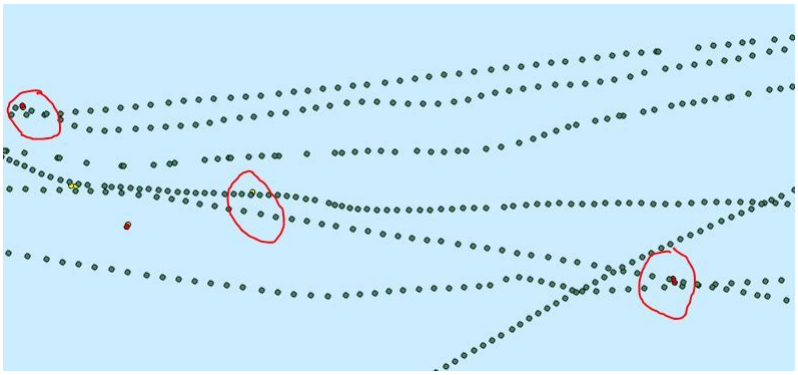


Figure: A highly simplified example of co-travelers (showing ships at sea or the motion of individuals in a crowd over time). The different paths of dots each represent the location of a ship, person or other item of interest at a certain location at a certain time. If the distance between them goes below a threshold, then the objects are said to be co-travelers.

Co-traveler Analysis

The problem of identifying "co-travelers" has been studied extensively in a variety of applications for the intelligence community. At an abstract level, the problem can be summarized as follows: given an entity X, identify the set {Y_1, Y_2, … Y_n} of other entities that have a spatial/temporal relationship with entity X. Examples of co-traveler problems include:

- Contact tracing of individuals for pandemic control in the context of public health;
- Co-location of individuals to determine relationships in the context of social media (i.e., people that arrive at the same building at the same time likely work together or know each other); and,
- Interdiction of illicit substances (i.e., a truck known to be carrying illicit cargo may pass this cargo off to other vehicles---to do so means they need to be in the same place at the same time).

During the summer of 2020, the team worked with NGA on a problem of tracking using AIS shipping data. Computing of queries related to co-traveler problems could be implemented in a number of ways, including via graph structures or hash tables, and is a potential challenge problem for graph search hardware. The challenge of assessment would be to develop a large and sophisticated enough database, along with baseline computational benchmarks, to test against. Such a database could be created using commercially available AIS or FAA flight data.

How to generate co-travelers in a dataset?

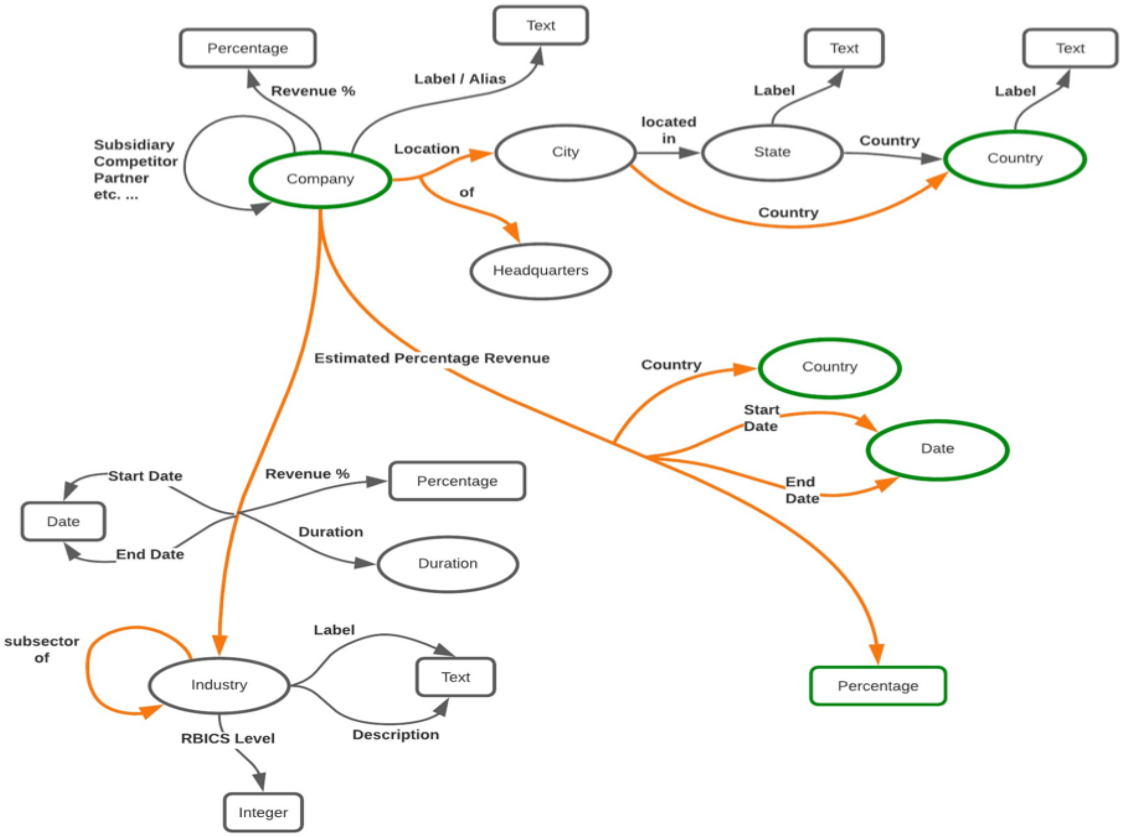


Figure: An example knowledge graph structure.

Knowledge Graphs

A knowledge graph, or triple store, was found to be a common structure used across several program activities surveyed. In a knowledge graph, semantic information is encoded in the structure of the graph with nodes corresponding to entities (sometime entities that have

associated ontological properties) and edges corresponding to relationships among the entities i.e., set membership, property relationships, etc). In recent years a number of commercial and opensource products have emerged for managing triple stores and knowledge graph datasets, all of which are generally known to have problems with issues of scale. → According to?

The main interrogations of a knowledge graph relate to using it to conduct patterns of inference. Rather than searching within the graph or doing graph matching, a knowledge graph is viewed as a knowledge base and the principal queries are logical inferences drawn from the data. Logical inference of this type can be viewed as a form of search (i.e., forward chaining in Horn-clause logic), hence there are at least two types of computational costs: graph traversals (to find facts or rules in the graph) and the execution of inference procedures (i.e., as heuristic search). In practice, these two are interleaved as available inferences would be determined by traversing the graph for available inference rules or matching facts.

In the context of knowledge graphs, there are two opportunities for hardware acceleration. The first is to change the information storage paradigm. In the same way that a GPU enables manipulation of triangles and the execution of linear algebra computations, a graph processing hardware system (such as Intel's PIUMA) can be used for direct storage of the graph structure and provide for hardware-accelerated access and traversal of the graph. This use of a graph processing system could produce significant improvements as the alternative is that the graph is stored in a traditional manner in which traversals of a graph could require complex and costly patterns of memory access (i.e., lots of paging and fetching would be required as there would be no guarantee that adjacent nodes in the graph are anywhere near each other in the memory architecture).

The second area is the possible application of hardware for accelerated graph searches that could improve the performance of existing, search-based, inference algorithms.

Figure: A screenshot of the user interface for the NSA AGLIA effort.

Patterns-of-Life Analysis (NGA's AGLIA)

The NGA's Anticipatory Ground-Level Imagery Analytics (AGLIA) program is developing an integration information system that fuses computer vision technologies with geo-spatial analytics to enable tracking of vehicular activity across large, urban-scale, environments. The principal performer on this effort is the University of Maryland's Center for Advanced Transportation Technology, an organization that maintains an database of realtime and historic transportation information from state and urban departments of transportation from across the United States (with a particular focus on the east coast and I-95 corridor). This information includes sensor data from highway systems as well as video information from imaging systems associated with traffic analytics, toll plazas, and other monitoring devices.

Typical uses for the CATT lab datasets are in support of the operational needs of the departments of transportation that sponsor the resource: predictive maintenance, traffic forecasting, etc. With AGLIA, NGA asked the team to develop algorithmic infrastructure for object recognition and tracking across urban environments. Functions enabled by this capability include tracking vehicles as they move across different areas of the city (and different cameras, camera angles, etc); tracking vehicles in spite of occlusions and gaps in contact; and predicting where vehicles will be at a given time in the future.

Processing in AGLIA includes considerable computation associated with computer vision algorithms, object detection and tracking. Hence, hardware acceleration that can improve processing of the video pipeline could enhance the system's capabilities.

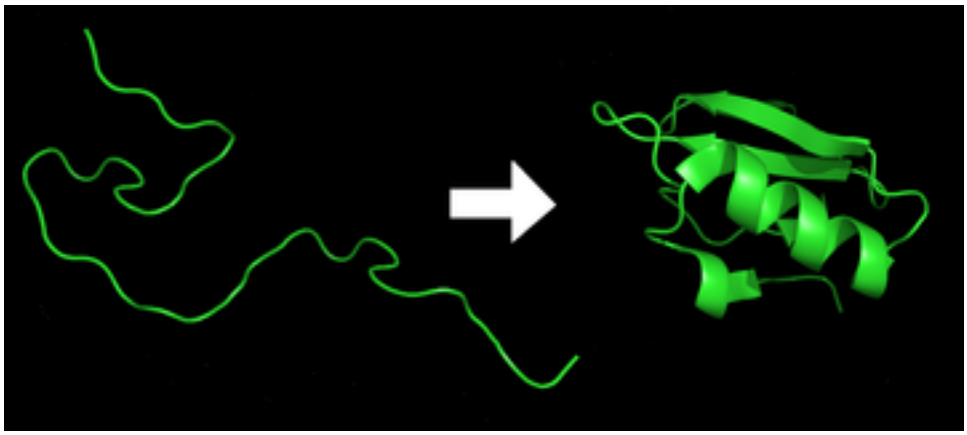


Figure: Example of Protien folding taken from https://en.wikipedia.org/wiki/Protein_folding

Protein Folding

Broadly speaking, protein folding encompasses many related tasks, including: (1) protein sequence alignment; (2) the prediction of a protein's 3-D structure given its amino acid sequence; (3) identification of protein folding mechanism(s) and pathways through structural state-space; and (4) the prediction of a protein's amino acid sequence, given its 3-D structure.

The task of identifying folding mechanism(s) focuses on identifying and/or predicting a protein's sequence of intermediate structural transitions (pathways), and the kinetic/topological mechanisms driving these transitions. Atomistic molecular dynamics (MD) simulations provide high-resolution information about the transition process, but require large amounts of computational power, since it is desirable to simulate as many time steps and trajectories as possible. Sampling from a large state space and accurate modeling of the potential energy function are persistent challenges. The calculations required to run these simulations involve floating point operations on large arrays; as such, they are parallelizable, and have benefitted from a tremendous amount of computational resources in recent years, in the form of GPUs, supercomputers, specialized ASICs, and crowd-sourcing applications, such as Folding@Home.

Protein sequence prediction is often referred to as the inverse protein folding problem, in that it focuses on predicting a protein's amino acid residue sequence from its 3-D structure for the purpose of biomedical and/or materials engineering. One example of recent work in this represents 3-D protein structures as graphs over the amino acid residues, and develop a transformer-based model to generate protein sequences conditional on the connectivity, topology, and atomic information of the input graph. This type of approach contains hybrid dense-sparse computations; however, additional investigation would be required to determine whether or not the scale of problem instances, and/or intermediate memory access patterns poses HIVE/SDH suitability challenges/opportunities.

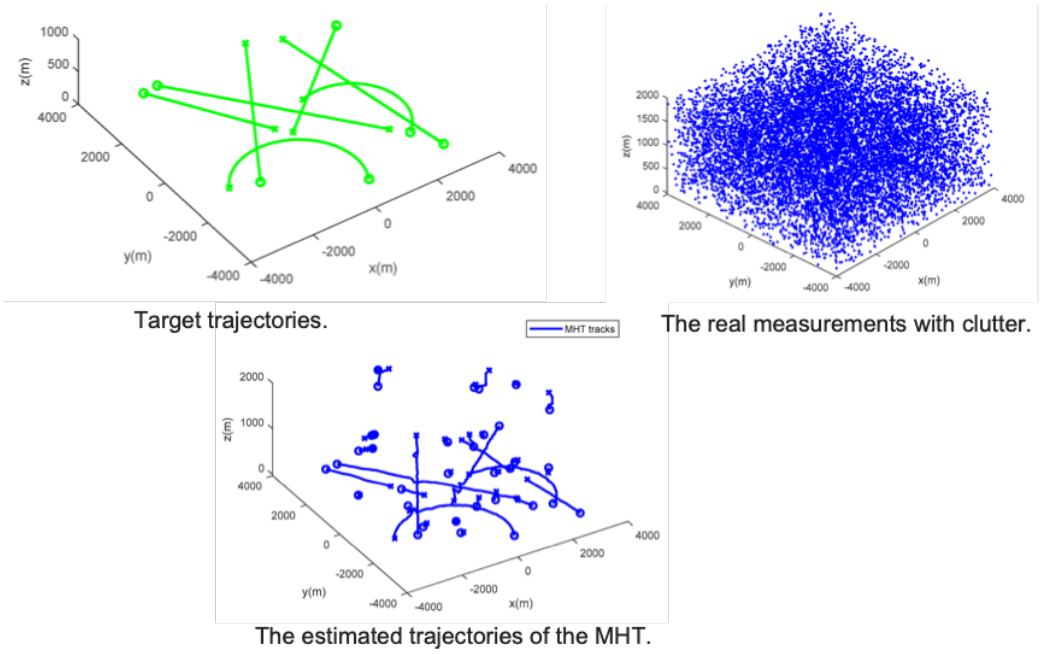


Figure: This example is taken from https://www.mdpi.com/1424-8220/20/17/4816/htm.

Multi-Hypothesis Tracking

The multi-hypothesis tracking problem has its origins in radar systems, for which one wants to track multiple targets and maintain active predictions of their future location and tracks. These techniques can be viewed as extensions of prediction systems, such Bayesian and Kalman Filters, but extended to non-linear systems and the possibility of their being many interacting objects that need to be tracked. The basic idea is that for each object, an MHT algorithm maintains a set of predictions of future locations. These predictions are constantly updated based on incoming sensor signals. Objects need to be deconflicted, sensor data needs to be associated with the right objects, and these functions need to be executed at each time cycle. The computational complexity of MHT is highly dependent on the number of tracks and the degree to which these tracks interact.

There are a number of algorithms for MHT, as it is a well-studied problem over the last decades. The current generation of tracking algorithms include, MS-MHT, Multi-INT Graph-Based Tracker (MI-GBT), and Markov Chain Monte Carlo (MCMC). Techniques such as MI-GBT incorporate graph-based methods which provide comparable performance with much less complexity. This approach provides efficiency in situations where sensor feeds are potentially sporadic but highly informative such as SIGINT feeds. Other tracking approaches are based on Markov Chain Monte Carlo (MCMC). The fundamental mathematical operations in these algorithms vary, some are graph based (maintaining associations among objects being tracked); others rely on matrix computations to produce maximum likelihood estimations of tracks for each object; yet others work by computing filters and probability densities.

Possible Algo?

Hardware acceleration could enhance the performance of MHT algorithms in a variety of ways, however the specific impact would be highly dependent on the nature of the underlying algorithm

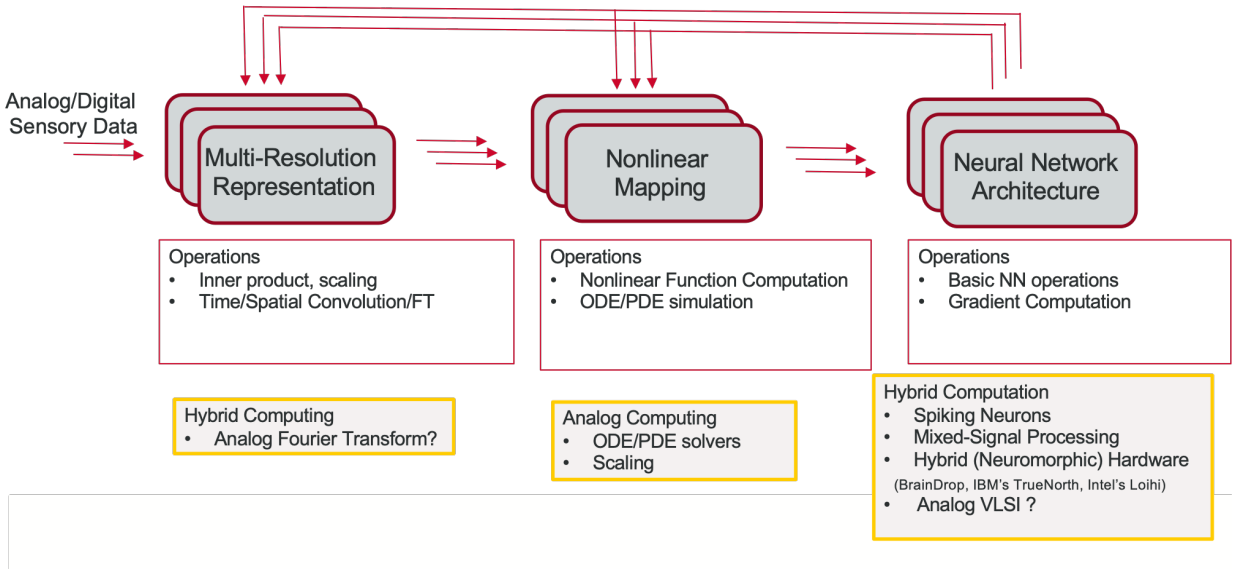


Figure: A conceptional example of a generic signal processing pipeline.

Signal Processing

Signal processing is a common computational workflow that occurs in nearly every system in which there is any kind of sensor. The steps are well understood and have been deeply studied for the last 75 years. There are computational challenges in each step of a signal processing workflow, ranging from the digital to analog conversion, to the mapping of the signals to the information used by systems to make decisions and take actions.

Emerging hardware technologies could be used to accelerate any of the key steps in this workflow. Indeed, technologies such as FPGA or other ASICs are often found in the processing pipelines for radars or tracking systems precisely because of the need for extremely accurate real-time performance. The design of an FPGA or an ASIC is the extremal case in which the needs of the processing requirement is so high that the design of a custom hardware implementation is warranted.

One opportunity in this application scenario is in the application of AI technologies, such as machine learning or deep learning, to the processing pipeline. As such algorithms are trained and, potentially, routinely updated over time, the idea of a fixed hardware implementation is potentially too rigid. Hence, programmable hardware in the processing pipeline may create added flexibility and upgradability with a relatively small performance impact.

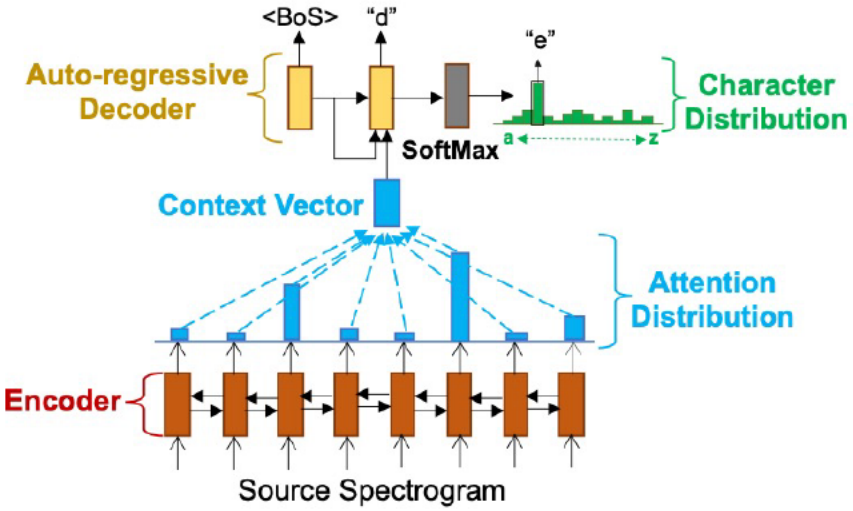


Figure: From the work of T. Tambe at Harvard and the 2021 IEEE International Solid-State Circuits Conference.

Natural Language Processing

Language processing offers several tasks in which hardware acceleration can have an impact. Driven by the emergence of many language applications that have resource and power constraints in mobile applications. An example of which is the work of T. Tambe at Harvard University, which aims to use hardware acceleration to improve the use of NLP Attention algorithms---those Bayesian techniques that map real-time signal processing of voice onto possible phonemes and contextually-driven text. Other drivers for custom hardware beyond energy consumption can exist elsewhere in the processing pipeline. Techniques for improving weight pruning in neural network approaches, improving activation levels and weights, and enhancing matrix computations have all been explored in the literature. For additional information, readers are referred to the appendix of this report.

Summary Observations

Several common themes emerged in these assessments and discussions with operational partners and possible transition agencies:

- **Scale**: The AI problems being addressed by these organizations had massive scales, on par with (or at least analogous to) the AI challenges of the major commercial vendors. Scale usually implied "billions to trillions" of objects that were subject to analysis.

- **Time**: The problems almost uniformly had a temporal component, requiring solutions that must execute successfully in a fixed time allocation. Common examples were processing tasks that were time bounded in one of two ways. For the first case, time was bounded by the interactive needs of a specific user (i.e., respond to a query in real-time or reasonable time); in the second case, time was based on the need to create a certain

analytical product or report (i.e., given a pile of fresh data, produce an analysis of the data in 2 hours).

- **Data Velocity**: In several, but not all, cases the data being analyzed had velocity. In this context, velocity of data is an informal parameterization of the rate of change in the data or the amount being accumulated (i.e., velocity can be thought of as a "first-order derivative" in the data). Examples in this case included streaming data, such as found in media analytics applications.

- **Classes of Data**: There are two broad classes of data identified among these agencies and problems:
  - Discrete data, such as that in the form of a graph (some set of nodes/edges, usually representing objects and relationships) or textual dataset (i.e., natural language documents, commercial or public data feeds, etc). Processing of these datasets is principally symbolic.
  - Continuous data, such as data created by sensors or associated with media streams. Such data streams are common in analog-to-digital conversion or sensor processing applications.
- **The Nature of Canonical Queries**: For datasets that were graphs, the nature of the interrogation of those graphs was largely similar across applications: reachability of nodes, connectivity and neighborhood analysis, and various forms of graph pattern matching.
- **Phases of Activity:** There are three principal phases of computational activities encountered in this survey.
  - A pre-processing or "training" phase, in which machine learning algorithms are trained, database indices are created, etc.
  - An update phase, in which new data is assimilated into an existing corpus.
  - An active phase, in which queries are executed and the data is interrogated.

  The potential impact of enhanced hardware varies depending on the phase, i.e., improving pre-processing training time by 10% is probably not as impactful as enhancing query performance by 10%.

Conclusions and Recommendations

This report summarizes the findings from an ongoing effort to document the computational workloads associated with applications from the Defense and Intelligence Communities that could benefit from hardware acceleration. In doing so, the team has worked to identify properties of the datasets being used, the nature of the queries being executed, and the computational pipeline associated with the execution of these queries. Some specific recommendations for ongoing or future research follow from these investigations:

- **"Edge AI":** In conduct of this survey, the team focused mostly on enterprise-scale applications of AI technologies rather than on processing pipelines sensitive to issues of power (i.e., AI "at the edge"). The applications surveyed were typically enterprise

applications that service activities with abundant computational and energy resources. A dimension not fully explored with the existing examples are challenges posed by limits in energy availability or constraints created by weight and size. Highly mobile applications, such as those on human-portable systems or on lightweight UAS, have SWAP concerns that could motivate the adoption of new hardware.

- **Testbeds and Experimentation:** It was evident that even with the most compelling of hardware technologies, their acceptance and integration into operational use would require extensive experimentation and evaluation. Organizations we spoke with did not have such experimentation capacity in their scope, hence the team concluded that there may be a government-wide need for testing and experimentation support to enable potential users to explore new hardware technologies and make acquisition and engineering decisions.

- **System Redesign:** Adoption of new hardware paradigms would, in almost every case, require a redesign of the software stack being used in the operational system. There was never an example in which technologies could just be applied by "swapping" out some subsystem. Execution processes and how the algorithms interact with the hardware needs to be rethought to adopt the new technologies.

- **New Functionality:** As expected, the organizations we spoke with were intensely fixated on the conduct of the missions associated with their datasets. The systems are designed based on their current tactics, techniques, and procedures (TTPs) and how system users are trained. New hardware that can radically accelerate key functions could create new capabilities that are not within current practice. The conception, design, implementation and training of such capabilities may be required to fully realize the benefits from new hardware.

APPENDIX

1. Hardware Accelerated Natural Language Processing: A Brief Literature Review
2. Protein Folding: A brief survey

Hardware Accelerated Natural Language Processing: A Brief Literature Review

Connor Baumler [1] and William Regli
The Applied Research Laboratory for Intelligence and Security (ARLIS)
University of Maryland
College Park, Maryland 20742

Pointless Read.

Contents

1. Introduction

Natural Language Processing, Natural Language Understanding, and Natural Language Translation are critical technology areas of Artificial Intelligence. Their wide application in areas ranging from media analytics to real-time translations are burgeoning across civilian and military use cases in which previous human agents were the main method of interpretation. This broad application of AI occurs across the processing pipeline: starting with the signal processing needs to perform speech-to-text to the neural networks that capture the statistical language models. The volume of media to be processed and the requirements for real-time (or better) performance make this a fertile domain for those exploring specific AI hardware technologies.

This paper presents a brief survey of techniques for the use of custom hardware for accelerated language processing. The specific focus is on neural encoder-decoder methods (Recurrent Neural Network (RNN) variants such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) as well as Transformers) that are widely used by the NLP community for a variety of tasks but that face challenges with computation and latency overheads.

We consider a number of optimizations commonly found in the literature. For example, weight pruning and quantization help limit the size of the model and reduce or simplify the required calculations. Calculations can also be simplified by using an appropriate approximation of a network's non-linear activation functions. Finally, we consider optimizations to overall Matrix-Vector Multiplication (MVM) and Element-Wise (EW) operations to take better advantage of the hardware. These methods are all designed to improve time, space, or energy efficiency with a minimal effect on accuracy.

2. Method 1: Weight Pruning

Not all Recurrent Neural Network (RNN) or Transformer model weights have an equal impact on the final output. In fact, some weights may have a negligible effect during inference. This means that some weights may be pruned without affecting the overall accuracy but allowing for faster computation or decreasing the amount of space needed to store the model [1–9].

This is the idea behind distilled models.

[1]Corresponding author: baumler@umd.edu

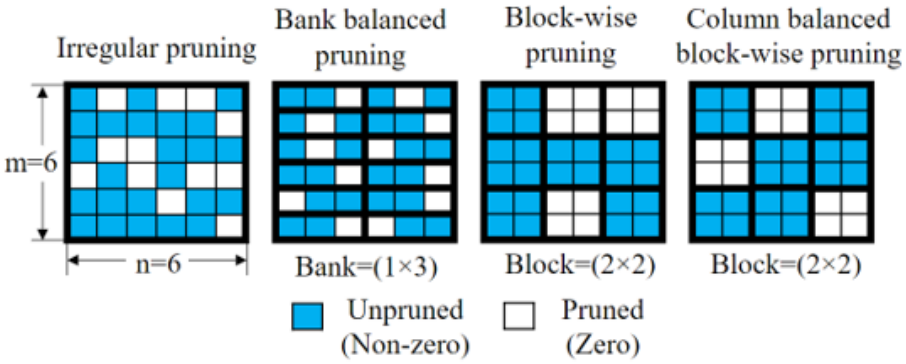


Figure 1. Via [1]. Four types of pruning pattern with 0.33 pruning ratio: irregular pruning, bank balanced pruning, block-wise pruning, and column balanced block-wise pruning

Top-*k* Top-*k* pruning removes all but the top-*k* non-zero values in the weight matrix [6]. Top-*k* pruning can also be done within sub-matrices for a result with more balanced density throughout the matrix [7, 9].

Bank balanced Similar to structured Top-*k* pruning, Bank balanced pruning (Figure 1) splits the weight matrix into sub-matrices, keeping the highest value weights. Here, the split and number of weights included or excluded is determined by some overall pruning rate.

Block-wise Block-wise pruning (Figure 1) also breaks the weight matrix into sub-matricides. Instead of removing some proportion of weights from each sub-matrix (as in bank balanced pruning), entire blocks are pruned based on their L2 norm.

Column Balanced Block-wise Column balanced block-wise pruning (Figure 1) builds on bank balanced and block-wise pruning [1]. Like the block-wise pruning method, it splits the weight matrix into sub-matrices and remove weights in certain blocks with low L2 norm. Instead of removing blocks with the overall lowest L2 norm, the column balanced version will remove the same proportion of blocks in each column. This method creates a representation with lower memory consumption and simpler indexing than previous sparse matrix patterns. It also allows for easy inter and intra-block parallelism.

Column Sparse Weight matrix Pruning [8]'s method prunes Weight matrix columns based on the sum of the absolute value of their weights being below some threshold. By pruning entire columns, the pruned matrix can be stored as a dense matrix with pruned columns removed entirely.

Cascade Pruning Cascade Pruning takes advantage of how, in a Transformer, once a value has been pruned in a layer, it can also be pruned in all subsequent layers [6]. This can applied by pruning tokens and heads with low importance scores. The importance of a token is determined by the cumulative attention probabilities across layers. Head importance is determined by cumulative magnitude.

Permuted Block Diagonal Mask Matrices [10]'s pruning method uses permuted block diagonal matrices where the weight matrix is broken down into sub-matricides and only an single offset diagonal in each sub-matrix is kept. This method retains accuracy as the level of sparsity increases better than Bank-Balance Sparsity.

Summary

3. Method 2: Quantization of Weights and Activations

As we have seen, some model weights can be pruned out entirely, but for those that are still necessary, it is desirable to store the values as efficiently as possible. Weights and calculated activation values can be quantized to a fixed-point representation [5–7, 10–13]. If done correctly, this will not affect model accuracy but will decrease storage requirements and potentially simplify calculations.

Figure 2. Via [10]. Normalized linear quantization flow of model weights

Linear quantization One simple method to quantize model weights and activations is linear quantization which converts floating point numbers to a fixed-point representation [10, 11]. For instance, weights can be quantized by taking

$$w_{i,j}^{q} = \frac{round(2^{k-1} \times w_{i,j})}{2^{k-1}}$$

where k is the desired number of bits and W is the weight matrix [10]. The values also have to be saturated or scaled to make sure they do not under or overflow the possible values given the level of precision. For instance, in [10]:

$$w_{i,j}^{s} = \max\left(\min\left(1 - \frac{1}{2^{k-1}}, w_{i,j}^{q}\right), -1 + \frac{1}{2^{k-1}}\right)$$

However, this saturation method can lead to vanishing gradients which lead [10] to normalize values between $[-1, 1]$ before quantizing (Figure 2).

Progressive Quantization Progressive Quantization trades computation for memory reduction [6]. For attention probability distributions, if a small number of tokens dominate the distribution, only the MSB and sometimes LSB are required. Progressive Quantization starts by fetching the MSBs only. If the confidence in the attention output is low, it also fetches the LSBs and recomputes.

Mixed schemes While many methods adopt a single quantization scheme, others select a mixed scheme, applying different quantization methods to specific parameters. [7] use both a fixed-point and a log-domain quantization scheme. For the weight matrices, they use log-domain quantization where values are quantized to a fixed number of positive and negative integer powers of two. For any remaining parameters, their fixed-point method is similar to the linear quantization approach described above.

Summary

4. Method 3: Activation Function Approximations

Neural models use a number of non-linear activation functions such as tanh and sigmoid in their calculations. Using these functions is computationally expensive, so accelerated methods often approximate them using a number of methods.

Look-up tables A simple method of mitigating the computational complexity of using tanh or sigmoid is to create a lookup table of activation values. The drawback of this approach is the potentially large amount of memory needed to get a robust approximation.

Polynomial Instead of resorting to a look-up table, other methods try to simplify the activation function calculations. One way to do this is to use a polynomial approximation [14, 15]. This will be more space efficient than the look-up table and more computationally efficient than using the full tanh or sigmoid, but the calculations can still be somewhat expensive.

Piece-wise linear A piece-wise linear approximation can be more computationally efficient method [8, 10, 16–20]. Given enough segments, this approach can achieve a good approximation of the tanh and sigmoid activation functions, using less computational power than polynomial approximations and less space than a look-up table.

Summary

5. Method 4: Matrix-Vector Multiplication and Element-Wise Operation Optimizations

As much of the expensive computation in these neural encoder-decoder methods comes from Element-Wise (EW) operations and especially Matrix-Vector Multiplication (MVM), many NLP accelerators seek to optimize these calculations.

Overall, the major takeaways are to parallelize these operations as much as possible [1, 2, 4, 6–8, 15–18, 21–25] and to pipeline them [6, 16, 20, 21, 21, 25, 25, 26] to make sure all hardware is in use as much as possible, without needing to wait for other calculations or retrievals from memory.

For instance, some accelerators will **re-order network calculations** [23, 27, 28]. These changes can help eliminate data-dependencies (waiting for data to be retrieved before a calculation can be done) or allow the calculated output to be quickly consumed without needing to be fully stored.

Using **Tiled Matrix-Vector Multiplication (MVM)** [16] can also help in parallelization. It breaks the weight matrix and input vector into smaller tiles, accumulating the resulting products using a binary adder tree. Since this computation takes longer than data access, they can be overlapped, saving time in the overall pipeline.

[27] use a novel "Omni"-Processing Element that **maximizes the logic reused between matrix multiplication and element-wise operations**. This also allows for the dynamic distribution of resources, handling the irregular workloads resulting from moving some back-propagation computation to forward cells.

Element-Wise (EW) operations can also be optimized in a variety of ways. [25] **time-division multiplex EW modules**, dividing them into three states. Using this method, the modules only require one sigmoid, tanh, adder, and multiplier, as none of these operations will be needed more than once per state.

There are further optimizations that can take advantage of pruned or binarized model weights [7, 9, 24]. For instance, unstructured sparsity can cause the workload between processing elements to be unbalanced (called the load-imbalance problem). [7] use a structured top-k prune leaving only a certain number of non-zero weights in each group of weights. Then they can assign the grouped weights to parallel processing elements, knowing that the load will be balanced.

Summary

6. Conclusion

This paper presents a number of optimizations commonly found in the accelerated NLP literature: pruning, quantization, activation function approximation, and MVM and EW operation optimizations. These methods can be applied to RNN variants such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) as well as Transformers to improve time, space, or energy efficiency with a minimal effect on accuracy. It is our belief that hardware accelerators that focus on these mathematical optimizations could have significant positive impacts.

Disclaimers

About ARLIS

The Applied Research Laboratory for Intelligence and Security (ARLIS) is the University Affiliated Research Center (UARC) of the Office of the Undersecretary of Defense for Intelligence and Security (OUSD(I&S)).
Find this and other ARLIS reports at:
https://go.umd.edu/ARLISreports.

References

[1] Peng H, Huang S, Geng T, Li A, Jiang W, Liu H, Wang S, Ding C (2021) Accelerating transformer-based deep learning models on fpgas using column balanced block pruning. *2021 22nd International Symposium on Quality Electronic Design (ISQED)* (IEEE), , pp 142–148.

This paper investigates a novel method for pruning transformer weights, column balanced block-wise pruning, and presents an accompanying Field Programmable Gate Array (FPGA) accelerator. Column balanced block-wise pruning builds on bank balanced and block-wise pruning. Like the block-wise pruning method, they split the weight matrix into sub-matrices and remove weights in certain blocks with low L2 norm. Instead of removing blocks with the overall lowest L2 norm, the column balanced version will remove the same proportion of blocks in each column. This method creates a representation with lower memory consumption and simpler indexing than previous sparse matrix patterns. It also allows for easy inter and intra-block parallelism. The paper also introduces a specialized process element to accelerate the multiplication of these spare matrices. Implementing this method on an FPGA with optimized resource scheduling, they achieve high overall throughput. Comparing to a Graphics Processing Unit (GPU) implementation, they find their implementation achieves a 2.08× speed up.

[2] Diamantopoulos D, Hagleitner C (2018) A system-level transprecision FPGA accelerator for blstm using on-chip memory reshaping. *2018 International Conference on Field-Programmable Technology (FPT)* (IEEE), , pp 338–341.

This paper introduces a system-level implementation framework for the acceleration of BLSTM-based (bi-directional LSTM) neural networks. Their implementation utilizes a number of optimizations in computations and on the data. They apply bit-with optimization (as LSTM variables are, in narrow regions of computational progress, updated with values with low dynamic range), use approximate activation functions, apply loop and task pipelining and explore parallelization in every layer, and instantiate multiple accelerators on the same FPGA. They also prune their model to store more parameters per FPGA memory element. They find their proposed architecture yields a 2.6× decrease in energy-to-solution versus a GPU implementation.

[3] Ribes S, Trancoso P, Sourdis I, Bouganis CS (2020) Mapping multiple LSTM models on fpgas. *2020 International Conference on Field-Programmable Technology (ICFPT)* (IEEE), , pp 1–9.

This paper presents the first work on mapping multiple LSTMs (i.e., Hierarchical or Stacked LSTMs) to a single device using FPGAs. They focus on approximating multiple LSTMs together, co-optimizing the scheduling of computation and of hardware parameters. They use structured pruning to allow for approximations with better device utilization. However, their approach can be extended to other pruning frameworks to control the computation to communication ratio. They found their implementation to have a 3.1× to 5.6× performance improvement over state-of-the-art approaches for the same accuracy loss.

[4] Rizakis M, Venieris SI, Kouris A, Bouganis CS (2018) Approximate FPGA-based lstms under computation time constraints. *International Symposium on Applied Reconfigurable Computing*, , pp 3–15.

This paper proposes an approximate LSTM scheme with novel FPGA hardware architecture focusing on time-constrained settings. Their approach introduces an iterative approximation method which applies low-rank compression using SVD-based decomposition and structured pruning to the model. Their FPGA architecture is designed to exploit the coarse-grained parallelism of LSTM gates and is parameterized wrt the fine-grained parallelism iof LSTM dot product and EW operations. This strategy allows for tuning of the performance-resource trade-off. Evaluating their method on an image captioning task, they find their method requires up to 6.5× less time to achieve the same accuracy as their FPGA baseline and achieves an average 25× higher accuracy under equal computation time constraints.

[5] Saxena S, Kumari S, Kumar S, Singh S (2021) Text classification for embedded FPGA devices using character-level cnn. *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)* (IEEE), Vol. 1, pp 802–807.

This paper presents a FPGA-accelerated 1D Temporal CNN architecture for multi-class character-level text classification. Their implementation quantizes and prunes weights, accelerating convolution with a minimal loss in accuracy. Their method achieved a 50% decrease in latency and 1500% decrease in power usage over a GPU implementation. They also profile TCNN layers on CPUs, GPUs, and FPGAs to determine how efficiently they can optimize TCNN wrt power and speed.

[6] Wang H, Zhang Z, Han S (2021) Spatten: Efficient sparse attention architecture with cascade token and head pruning. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (IEEE), , pp 97–110.

This paper presents an efficient algorithm-architecture for transformers using an Application-Specific Integrated Circuit (ASIC). They make a number of contributions in their pruning and quantization methdology.

Instead of pruning weights, Cascade Token Pruning and Cascade Head Pruning remove unimportant tokens and heads respectively. Here, "cascade" refers to how, after removing a token/head from the current layer, the pruners automatically removes it in all subsequent layers after. The importance of a token is determined by the cumulative attention probabilities across layers. Head importance is determined by the cumulative magnitude. This requires sorting the token and head importance scores on the fly, which is done using a highly parallel top-k engine with specialized memory hierarchy and a fully-pipelined datapath which achieves $O(n)$ time complexity. Token and head pruning reduce DRAM access and computation by an average $3.8\times$ and $1.1\times$, respectively.

Progressive Quantization trades computation for memory reduction. They observer that, for attention probability distributions, if a small number of tokens dominate the distribution, only the MSB and sometimes LSB are required. The proposed Progressive Quantization starts by fetching the MSBs only. If the confidence in the attention output is low, it also fetches the LSBs and recomputes. This method decreases DRAM access by $5.1\times$.

They evaluate their method using BERT and GPT-2 on a number of benchmarks such as GLUE and SQuAD finding no accuracy loss and a $10.0\times$ reduction in DRAM access. They find a $1.6\times$ speedup and $1.4\times$ speedup over [21] and a $162\times$ speedup and $1193\times$ speedup over a GPU implementation.

[7] Wang M, Wang Z, Lu J, Lin J, Wang Z (2019) E-LSTM: An efficient hardware architecture for long short-term memory. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9:280–291.

They employ a Hardware-Oriented Compression Algorithm (HOCA) including clipped gating, structured top-k pruning, and multiplication-free quantization to reduce the size of the model and number of matrix operations, without sacrificing accuracy (largely following [9]). Clipped gating zeros out results from the output gate that are below some threshold. Top-k pruning removes all but the top-k non-zero values in sub-matricies of the weights. The HOCA also uses two quantization schemes: a fixed-point and a log-domain quantization scheme. This allows most of the LSTM multiplications to be replaced with bit shifts.

Beyond compression, they also reorganized the flow of computation and optimized the memory access pattern, improving throughput by mitigating the memory bandwidth bottleneck. They also utilize a parallel processing strategy to take advantage of the HOCA compressed model's sparsity,

leading to high hardware utilization efficiency.

Their FPGA-accelerated design achieved a $1.4 - 2.2\times$ improvement in energy efficiency compared with the state-of-the-art FPGA-accelerated LSTM implementations.

[8] Wang S, Lin P, Hu R, Wang H, He J, Huang Q, Chang S (2019) Acceleration of LSTM with structured pruning method on FPGA. *IEEE Access* 7:62930–62937.

This paper presents an FPGA-accelerated LSTM using a novel structured-pruning approach to reduce model size and and balance computation and regulate memory access. Weight matrix columns are pruned based on the sum of the absolute value of their weights being below some threshold. By pruning entire columns, the pruned matrix can be stored as a dense matrix with pruned columns removed entirely. In the hardware implementation, they make a number of optimizations such as processing independent MVMs in parallel and using piece-wise linear approximations of the sigmoid and tanh activation functions. They find that their pruning method achieves a $7.82\times$ speedup using $\frac{1}{8}$ the parameters.

[9] Wang Z, Lin J, Wang Z (2018) Hardware-oriented compression of long short-term memory for efficient inference. *IEEE Signal Processing Letters* 25:984–988.

This paper examines a number of methods for compressing LSTM models for hardware acceleration. To keep activations sparse, they employ clipped gating on the forward passes of the output gate. Here, they zero any output below some given threshold. To enforce weight sparsity, they top-k prune the weights. To do this, the weights are split into groups with some number of values. The values outside of the k-highest in magnitude are set to zero. They also employ a mixed quantization scheme that allows most of the LSTM multiplications to be replaced with bit shifts. Evaluating on a word-level language modeling task, they find that their methods reduce the model size and number of matrix operations by $32\times$ and $18.5\times$, respectively, while losing less than 1% accuracy.

[10] Zheng Y, Yang H, Jia Y, Huang Z (2021) Permlstm: A high energy-efficiency LSTM accelerator architecture. *Electronics (Switzerland)* 10.

This paper presents an FPGA-based LSTM accelerator. Their methods focus on solving two issues in model compression: irregular sparsity and large overhead in weight pruning and gradient vanishing in linear quantization. Their pruning method uses permuted block diagonal matrices where the weight matrix is broken down into sub-matricies and only an single offset diagonal in each sub-matrix is kept. This method retains accuracy as the level of sparsity increases better than Bank-Balance Sparsity. They quantize values using linear quantization with local normalization. Model weights and activations are normalized within $[-1, 1]$ and and then quantize them in a local mix-max range. They also employ a strategy of piece-wise linear approximation of the sigmoind and tanh activation functions which matches the operand precision for better accuracy.

Using these methods, they create a sparse LSTM, which can be accelerated using their proposed PermLSTM accelerator. Their method especially focuses on using their enforced sparsity to speed up MVMs. These changes resulted in a 55.1% reduction in power consumption. Their accelerator achieved a $2.19\times - 24.4\times$ improvement in energy efficiency over previous FPGA-based LSTM accelerators including [7, 13].

[11] Azari E, Vrudhula S (2019) An energy-efficient reconfigurable LSTM accelerator for natural language processing. *2019 IEEE International Conference on Big Data (Big Data)* (IEEE), , pp 4450–4459.

This paper proposes several improvements to FPGA implementations of LSTMs to be more energy efficient. They integrate a an improved approximate multiplier into the compute-intensive units of the LSTM. The basic multipliers, which require a variable number of clock cycle depending on the

magnitude of operands. In contrast, this design synchronizes these variable-cycle multiply operations with other single-cycle units using hierarchical controllers. This work further improves performance and energy efficiency via the post-training, range-based linear quantization of the model parameters. These changes led to improvements in energy-efficiency of up to 45.26× against existing implementations.

[12] Lee M, Hwang K, Park J, Choi S, Shin S, Sung W (2016) FPGA-based low-power speech recognition with recurrent neural networks. *2016 IEEE International Workshop on Signal Processing Systems (SiPS)* (IEEE), , pp 230–235.

This paper introduces an FPGA implementation of an RNN-based real-time speech recognition system. Their system consists of two RNNs, an acoustic model and a character-level language model. Instead of an HMM, they use an N-best Beam Search to combine these two models (as well as an additional word-level statistical language model). The implementation is optimized for FPGA usage by using high throughput MVMs and by quantizing the RNN weights to store them all in on-chip memory. They find their system can achieve 4.12× real-time speed and is about 10× more power efficient than a GPU-based system.

[13] Rybalkin V, Pappalardo A, Ghaffar MM, Gambardella G, Wehn N, Blott M (2018) Finn-l: Library extensions and design trade-off analysis for variable precision LSTM networks on fpgas. *2018 28th international conference on field programmable logic and applications (FPL)* (IEEE), , pp 89–897.

This paper explores FPGA implementations of BiLSTMs, investigating tradeoffs between precision vs accuracy, "hardware computation cost, storage cost, power and throughput scalability." They also examine "the effects of quantization of weights,input, output, recurrent, and in-memory cell activations during training." They present an FPGA-accelerated implementation of an LSTM and BiLSTM with binarized weights and activations. Applying their method to OCR, they find it achieved the highest throughput of any comparable FPGA-accelerated model.

[14] Chen K, Huang L, Li M, Zeng X, Fan Y (2018) A compact and configurable long short-term memory neural network hardware architecture. *2018 25th IEEE International Conference on Image Processing (ICIP)* (IEEE), , pp 4168–4172.

This paper presents a compact and configurable FPGA implementation of an LSTM. They provide a large number of hardware parameters to allow for the balancing of area, power, and performance of the network. They approximate the LSTM activation functions using second-order polynomials to balance accuracy and resource utilization. Their implementation was found to use 77% less resources while still being highly accurate.

[15] Ferreira JC, Fonseca J (2016) An FPGA implementation of a long short-term memory neural network. *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, , pp 1–8.

This paper provides an early FPGA LSTM implementation. They use polynomial approximations of activation functions as their evaluation has low memory usage requirements. They also parallelize the MVM to perform in linear time. Additionally, they find that certain signals in the model do not depend on each other and can be computed via parallel gate modules. They find that their method yields a 251× speedup in comparison to a software network.

[16] Zhang Y, Wang C, Gong L, Lu Y, Sun F, Xu C, Li X, Zhou X (2017) Implementation and optimization of the accelerator based on FPGA hardware for LSTM network. *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)* (IEEE), , pp 614–621.

This paper presents an FPGA-based accelerator for LSTMs. They use tiled MVM which breaks the weight matrix and input vector into smaller tiles, accumulating the resulting products using a binary

adder tree. Since this computation takes longer than data access, they can be overlapped, saving time in the overall pipeline. They also save time by using a piece-wise linear approximation of the tanh and sigmoid activation functions which, when pipelined, can be computed in parallel. They find that their optimized LSTM can out-speed a CPU baseline by 11× while being 57× more energy efficient.

[17] Xiang L, Lu S, Wang X, Liu H, Pang W, Yu H (2019) Implementation of LSTM accelerator for speech keywords recognition. *2019 IEEE 4th International Conference on Integrated Circuits and Microsystems (ICICM)* (IEEE), , pp 195–198.

This paper presents an FPGA-based LSTM accelerator. They propose using iterative linear approximation (ILA) for the activation functions. These piece-wise linear sigmoid approximations are more memory efficient than look-up tables and less computationally complex than linear approximations. They also use a matrix of PE units witch allow for the parallel operation of five LSTM units. Applying their implementation to speech keyword recognition, they find that their approach is 30× more energy efficient than a GPU baseline and achieves and accuracy of 99.2%.

[18] Li CL, Huang YJ, Cai YJ, Han J, Zeng XY (2018) FPGA implementation of LSTM based on automatic speech recognition. *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)* (IEEE), , pp 1–3.

This paper proposes an FPGA implementation of an LSTM for ASR. They use piece-wise linear approximations of the sigmoid and tanh functions for their activations, and use four parallel MAC units to compute the matrix-vector product of the input weight matrices and the hidden layer output of the previous timestep. To compress their LSTM parameters, they use UV Decomposition and fixed-point data conversion. They find that their implementation a low word error and character error rate while achieving a 38× speedup and 7× decrease in parameter size in comparison to a CPU implementation.

[19] Kouretas I, Paliouras V (2018) Hardware aspects of long short term memory. *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (IEEE), , pp 525–528.

This paper examines the hardware implementation aspects of LSTMs. They propose using piece-wise approximations of the activation functions σ and tanh and a corresponding training procedure which exploits this approximation. When compared to training with exact σ and tanh derivatives, they find their procedure leads to faster convergence and a lower error floor. They also compare their work to Look Up Table (LUT)-based implementations, finding up to a 31% decrease in complexity in area×delay terms.

[20] Sun Z, Zhu Y, Zheng Y, Wu H, Cao Z, Xiong P, Hou J, Huang T, Que Z (2018) FPGA acceleration of LSTM based on data for test flight. *2018 IEEE International Conference on Smart Cloud (SmartCloud)* (IEEE), , pp 1–6.

This paper proposes an FPGA-based LSTM-RNN accelerator that focuses on accuracy and speed. Their work does not focus on an NLP application, but instead considers aircraft anomaly detection, a task with short data sampling intervals and high speed and precision requirements. Their method unrolls all LSTM layer loops and break the computation into small blocks. They pipeline this process to limit hardware reads and writes and speedup the necessary accesses. They also approximate the sigmoid function a using "hard" piece-wise approximation. They do not use "hard" tanh as it increases the error rate to an unacceptable level. They find that their approach reaches 13.45 GOP/s, with an overall 28.76× acceleration over a CPU baseline.

[21] Ham TJ, Jung SJ, Kim S, Oh YH, Park Y, Song Y, Park JH, Lee S, Park K, Lee JW, et al. (2020) A^3: Accelerating attention mechanisms in neural networks with approximation. *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (IEEE), , pp 328–341.

This paper proposes an accelerator for neural network attention mechanisms. They observe that, since attention mechanisms are functionally performing a content-based search, the current practice of implementing attention using MVM is suboptimal as much of the computation ends up unused. Their proposed accelerator addresses this via algorithmic approximation which allows for finding potentially relevant search targets non-exhaustively as well as designing a hardware pipeline to exploit parallelism and datapath specialization. They find their accelerator achieves "multiple orders of magnitude speedup and energy efficiency over conventional hardware" while maintaining model accuracy.

[22] Zhang X, Jiang W, Hu J (2020) Achieving full parallelism in LSTM via a unified accelerator design. *2020 IEEE 38th International Conference on Computer Design (ICCD)* (IEEE), , pp 469–477.

This work presents a generic LSTM accelerator design for FPGA and ASIC. The main contribution of the design is a unified computing kernel that incorporates EW multiplication and addition (MVM is replaced with unified EW multiplication followed by addition.) to allow for the parallel execution of all LSTM gates. They also propose a model that uses LSTM network size and device specifications to determine accelerator hyperparameters, accelerator running schedule, and predict the system's performance. Implementing their system on an FPGA, they find their methods result in a 10× speedup in inference and a 15.2× improvement in computing power efficiency over an FPGA baseline.

[23] Azari E, Vrudhula S (2020) Elsa: A throughput-optimized design of an LSTM accelerator for energy-constrained devices. *ACM Transactions on Embedded Computing Systems (TECS)* 19(1):1–21.

This paper presents a scalable ASIC design of an LSTM that is geared toward energy-constrained devices like mobile systems. Its implementation makes a number of contributions such as using approximate multiplication to reduce area and power consumption without sacrificing performance. This approximate multiplier uses four controllers to synchronize single-cycle and variable-cycle operations. Since some LSTM computation units have data dependencies, some of which can be executed in parallel, their system incorporates pipelining at three levels (MAC, LSTM, and application) to maximize utilization. Comparing this implementation to a baseline LSTM, their approach exceeded the baseline's energy and area efficiency by factors of 1.2× and 3.6×, respectively. Comparing to other ASIC LSTM implementations, their approach was up to 1.2× more energy efficient.

[24] Mealey T, Taha TM (2018) Accelerating inference in long short-term memory neural networks. *NAECON 2018-IEEE National Aerospace and Electronics Conference* (IEEE), , pp 382–390.

This paper investigates acceleating LSTM inference using FPGAs. They binarize model parameters to reduce necessary storage size and simplify computations. To take advantage of this, they propose a Binary Recurrent Unit (BRU) which uses model weights and on-chip memory resources to parallelize inference. They find that their implementation provides up to a 3.8× speedup over a GPU benchmark on a speech recognition task, with a 3.18% loss in accuracy.

[25] Zhang W, Ge F, Cui C, Yang Y, Zhou F, Wu N (2020) Design and implementation of LSTM accelerator based on FPGA. *2020 IEEE 20th International Conference on Communication Technology (ICCT)* (IEEE), , pp 1675–1679.

This paper presents an FPGA-based LSTM accelerator. Their method includes a multi-level storage strategy with off-chip DRAM and three buffers. This reduces the FPGA's main resource utilization rate. Most LSTM computations are completed in an MVM module and a EW modules. With careful arrangement of multiple Processing Engines, the MVM can be parallelized and pipelined. The EW modules are time-division multiplexed, dividing them into three states. Using this method, the modules only require one sigmoid, tanh, adder, and multiplier, as none of these operations will be needed more than once per state. They find their method achieves a 3× acceleration over a

GPU using only 11% of the power. Their system achieves 10.90 GOP/s performance, a significant improvement over [36]'s 7.26 GOP/s.

[26] Bank-Tavakoli E, Ghasemzadeh SA, Kamal M, Afzali-Kusha A, Pedram M (2019) Polar: A pipelined/overlapped FPGA-based LSTM accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28(3):838–842.

This paper proposes a FPGA-based LSTM network architecture designed for an accelerated inference phase. The acceleration is acheived through overlapping the operations of the *Function* and *Gate* modules (the former performing activation functions and dot product multiplications and the latter performing matrix and vector arithmetic operations) as well as through pipelining the datapath. This changes lead the architecture to require negligible internal memory for storing intermediate data which allows for simple routing (improving operating frequency) and low resource utilization. Compared with recent FPGA LSTM implementations, this work "provides up to about 1.6×, 43.6×, 21.9×, and 114.5× improvements in frequency, power efficiency, GOP/s, and GOP/s/W, respectively."

[27] Zhang X, Xia H, Zhuang D, Sun H, Fu X, Taylor MB, Song SL (2021) η-LSTM: co-designing highly-efficient large LSTM training via exploiting memory-saving and architectural design opportunities. *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)* (IEEE), , pp 567–580.

This paper presents the first "cross-stack training solution" for large LSTMs, using customized NPU hardware. They identify a number of causes of training inefficiency in large LSTMs, with large intermediate variables being the main culprit. Their methods try to combat these issues in a number of ways.

They reduce the number of cell-level intermediate variables. Intermediate variables generated in the forward cells generally have to be stored for back-propagation. However, by re-ordering execution so that these variables can be consumed quickly without fully storing them. The variables that are created by these re-ordered computations can be more efficiently compressed.

They also decrease the length of the back-propagation layer. Depending on how an LSTM computes loss, the magnitudes of the gradients either decrease or increase from the last to first BP cell. Knowing this, they predict which cells are likely to have insignificant gradients for weight updating and skip them.

To take advantage of these software innovations, they also propose a new hardware accelerator. They present a novel "Omni"-Processing Element designed to reduce resource consumption by maximising the logic reused between matrix multiplication and EW operations. This also allows for the dynamic distribution of resources, handling the irregular workloads resulting from moving some BP computation to FW cells. They also a Runtime Resource Allocation scheduler to better asign the omni-PEs to MatMul and EW operations.

They find that their methods decrease the required memory footprint for a large LSTM by an average of 57.5%. Comparing to a state-of-the-art GPU implementation, they find their method improves performance by an average 3.99× while using an average 2.75× less energy.

[28] Que Z, Zhu Y, Fan H, Meng J, Niu X, Luk W (2020) Mapping large lstms to fpgas with weight reuse. *Journal of Signal Processing Systems* 92:965–979.

This paper focuses on an improved FPGA implementation for large LSTMs. They present a Block-batching stratgey which re-uses LSTM weights fetched from external memory. To do this they split "the weight matrix into multiple blocks while batching the input activations vectors" which allows

them to handle "calculations block by block with weight reuse." They also propose an architecture with reorganized multiplications to eliminate date dependency and stalls. They find that their approach achieves 1.65× higher performance-per-watt efficiency and 1.60× higher performance-per-DSP efficiency over the state-of-the-art approach that also stores weights in off-chip memory. They also find that their method is 1.3× faster than a GPU implementation while consuming 19.2× lower energy.

[29] Tavcar R, Dedic J, Bokal D, Zemva A (2013) Transforming the LSTM training algorithm for efficient FPGA-based adaptive control of nonlinear dynamic systems. *Informacije Midem-Journal of Microelectronics Electronic Components and Materials* 43(2):131–138.

This paper presents a LSTM training algorithm for use on FPGA-based control systems. The algorithm, Simultaneous Perturbation Stochastic Approximation (SPSA), uses a low-complexity, high-efficiency gradient approximation using two objective function measurements. The same absolute value is used to update all network weights. Though this lack of detail in update may hurt learning, they find that this change does not significantly reduce LSTM accuracy.

[30] Maeda Y, Wakamura M (2005) Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation. *IEEE Transactions on Neural Networks* 16:1664–1672.

While not directly about NLP, this paper's FPGA RNN implementation has been an important stepping stone for many hardware-accelerated LSTM papers. Intead of using backpropagation through time, they use simultaneous perturbation which only requires two error values. The same absolute update value is used for all network weights. They use their method to implement a HNN (Hopfield Neural Network), without needing an energy function. They find that their method can be used to accurately learn various patterns, but leave investigating capability of the simultaneous perturbation learning rule as future work.

[31] Silfa F, Arnau JM, González A (2020) Boosting LSTM performance through dynamic precision selection. *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)* (IEEE), , pp 323–333.

This paper presents a hardware accellerated LSTM that dynamically determines precision during inference. They observe that time steps where the LSTM cell state changes significantly require higher bit precision, while more stable time steps can be accurately computer with lower precision. The precision is selected using a state machine. After some number of timesteps in a profiling state, the state switches between stable and in-a-peak based on whether the LSTM cell state value is has left the range it was in during the profiling state. During the profiling and stable states, the precision is lowered. Using this method, the lowest possible accuracy preserving precision is chosen 57% of the time, outperforming methods with a fixed precision. They implement their method on top of an E-PUR, a state-of-the-art processing unit for RNN acceleration, finding that it improves evaluation speed by 1.46× and reduces energy consumption by 19.2% over E-PUR without sacrificing accuracy.

[32] Wang T, Wang C, Zhou X, Chen H (2019) An overview of FPGA based deep learning accelerators: challenges and opportunities. *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (IEEE), , pp 1674–1681.

[33] Liu J, Wang J, Zhou Y, Liu F (2019) A cloud server oriented FPGA accelerator for LSTM recurrent neural network. *IEEE Access* 7:122408–122418.

This paper presents a cloud-oriented FPGA accelerator for LSTMs. Their work optimizes both the on-chip LSTM computation and the cloud communication elements. While previous works focused on improving MVMs in the LSTM layer, cloud environments require the handling of multiple input

streams, leading this work to group vectors from multiple input streams into an input matrix. This method improves their throughput and resource usage, even as the number of input streams become high. Allowing for many input streams helps to optimize the transmission in the cloud environment, with the system achieving close to zero transmission time with a sufficiently high number of input streams. Comparing to similar FPGA schemes, they find their implementation achieves up to an 18× increase in GFLOP/s.

[34] Sheikhfaal S, Vangala MR, Adepegba A, DeMara RF (2021) Long short-term memory with spin-based binary and non-binary neurons. *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)* (IEEE), , pp 317–320.

This paper proposes a hardware implementation of an LSTM using Resistive Random-Access Memory (ReRAM) and non-binary neurons. Previous methods used Non-Volitile Memory devices, like ReRAM, for the Multiplicication and Accumulation operation which can require an undesirable level of power and area due to their use of non-linear sigmoid and tanh neurons as their main thresholding functions. This work uses non-binary neurons, which have five levels of output, to mimic sigmoid and tanh activation functions. They find that this method can achieve up to 85% accuracy which leads to similar algorithmic expectations to near-ideal neurons. Their implementation achieves a 34× improvement in energy efficiency and 2× reduction in area over competing non-binary designs.

[35] Rybalkin V, Wehn N, Yousefi MR, Stricker D (2017) Hardware architecture of bidirectional long short-term memory neural network for optical character recognition. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017* (IEEE), , pp 1390–1395.

This paper proposes the first (FPGA accelerated) BiLSTM hardware architecture with Connectionist Temporal Classification (CTC) for OCR. CTC allows the BiLSTM to directly label unsegmented textlines. Their architecture rearranges memory access patterns, interleaving the forward and backward columns during image processing. This avoids a duplication of a hidden layer and reduces the processing time and memory requirement. Their FPGA accelerated BiLSTM uses only on-chip memory, saving power and improving performance. They also decrease their weight representation to use 5-bit fixed-point numbers in comparison to the minimum 16-bits used by previous work. This method achieved still achieved 97.5120% recognition accuracy, a small decrease from a method using single-precision floating-point format which achieved 98.2337% accuracy. Their method achieved a throughput of 152.16 Giga-Operation Per Second (GOPS) at1 66 MHZ, 459× higher throughput than the state-of-the-art.

[36] Guan Y, Yuan Z, Sun G, Cong J (2017) FPGA-based accelerator for long short-term memory recurrent neural networks. *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE), , pp 629–634.

This paper introduces an FPGA implementation of an LSTM-RNN. They optimize computation by tiling the loop computation in each gate as they find that the computations done in LSTM gates is the most resource-consuming part of LSTM-RNN inference. They find their LSTM-RNNs have few enough parameters to be stored partially or entirely in FPGA on-chip memory. For the data that cannot be stored in on-chip memory, they introduce a data dispatcher which maximizes the utilization of data band-with between the external DRAM and the FPGA. They find that their accelerator achieves a peak performance of 7.26 GFLOP/S, a singificant increase over existing FPGA RNN implemenations.

[37] Lu S, Wang M, Liang S, Lin J, Wang Z (2020) Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*, , pp 84–89.

This work proposes a FPGA hardware accelerator for transformers. They split the encoder and decoder layers into two kinds of ResBlocks: multi-headed attention (MHA) and feed-forward network

(FFN) ResBlocks. These blocks are implemented such that their general matrix-matrix multiplications can be done one a single systolic array of limited size. They build on existing high-speed and low-complexity architecture for softmax. Comparing with a GPU baseline, their method achieved a 14.6× speedup in the MHA ResBlock, and 3.4× in the FFN ResBlock.

[38] Li S, Wu C, Li H, Li B, Wang Y, Qiu Q (2015) FPGA acceleration of recurrent neural network based language model. *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines* (IEEE), , pp 111–118.

This paper introduces an FPGA implementation of a RNN based language model (RNNLM). In their implementation, they mainly target reducing the data communication load. This is achieved by integrating a group of computational engines with a multi-thread management unit to mask the RNN's long memory latency during back-propagation and allow for the reuse of frequently accessed data. They find that their system maintains competitive accuracy. Though their FPGA implantation ran at a lower frequency than their tested CPU implementation, the FPGA had a higher sustained performance with an improved average efficiency.

[39] Maor G, Zeng X, Wang Z, Hu Y (2019) An FPGA implementation of stochastic computing-based LSTM. *2019 IEEE 37th International Conference on Computer Design (ICCD)* (IEEE), , pp 38–46.

This paper presents the first scalable Stochastic Computing-based (SC) LSTM architecture implementation. In SC, instead of directly representing numbers in static binary, there is a probability that each bit is 1 or 0 which is sampled over a number of cycles. Using this method greatly simplifies the arithmetic circuit, especially for multiplication. This lead to significantly reduced power consumption (73.24%) without much loss of accuracy when being compared to the binary LSTM design.

[40] Chang AXM, Culurciello E (2017) Hardware accelerators for recurrent neural networks on FPGA. *2017 IEEE International symposium on circuits and systems (ISCAS)* (IEEE), , pp 1–4.

This paper presents three strategies to implement LSTMs on FPGAs. These strategies address the issue of with high off-chip memory bandwidth or large internal storage requirements caused by the limited data reuse in RNNs. The first streams all data from off-chip memory into the co-processor (leading to poor off-chip memory bandwidth), the second stores all necessary data internally using on-chip memory (leading to low avalibility of on-chip memory), and the third balances the other two. The found the second approach to have the best performance per unit power; however, the third approach showed competitive performance with better scalability with respect to the number of MAC units.

[41] Nurvitadhi E, Sim J, Sheffield D, Mishra A, Krishnan S, Marr D (2016) Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, cpu, GPU, and ASIC. *2016 26th International Conference on Field Programmable Logic and Applications (FPL)* (IEEE), , pp 1–4.

This paper presents methods for accelerating GRU models using FPGAs. They propose a memoization optimization method which avoids 3 out of 6 of GRU's dense matrix vector multiplications (SGEMVs) but increases memory space usage by 3×. The other half are accelerated using FPGAs. They find their implementation to achieve better efficiency performance than CPU and GPU alternatives. They also find their implementation to be $\sim 7\times$ less efficient than an ASIC implementation; however, this gap is expected to lessen with newer-generation FPGAs.

PROTEIN FOLDING SUB-PROBLEMS: A HIGH-LEVEL OVERVIEW

ARLIS
University of Maryland, College Park

July 21, 2020

Broadly speaking, protein folding encompasses many related tasks, including: (1) protein sequence alignment; (2) the prediction of a protein's 3-D structure given its amino acid sequence; (3) identification of protein folding mechanism(s) and pathways through structural state-space; and (4) the prediction of a protein's amino acid sequence, given its 3-D structure [3]. A high-level overview of each of these tasks is provided below.

Protein sequence alignment Protein sequences are polypeptide chains of amino acid residues. The genetic code contains 20 unique amino acids; thus, from a computational perspective, protein sequences can be thought of as strings containing "letters" drawn from a common alphabet Σ, and protein sequence alignment can be thought of as an edit distance problem, where the objective is to identify the minimum number (or cost, in the event of non-uniform operation costs) of edit operations to perform in order to transform the input sequence, a, into the target sequence, b [3, 12].

The biological motivation for aligning protein sequences is that proteins which share a common evolutionary ancestor often share portion(s) of their sequences and/or structures, and variation in otherwise similar portion(s) of a sequence can be attributed to evolutionary dynamics (e.g., natural selection; transcription errors). Thus, pairwise alignment and multiple sequence alignment (MSA) can be used to infer/reason about homology [3, 9].

Protein structure prediction Protein structure prediction is the task of predicting the three-dimensional structure of a protein from its amino acid residue sequence. [1] This is an open problem with important biochemical and clinical implications, given that a protein's function is determined in large part by its structure [8]. As such, researchers have developed a wide range of experimental and computational approaches over the years [7].

Historically, computational work in this space has incorporated MSA techniques to identify homologous sequences, and/or been informed by thermodynamic and kinetic hypotheses regarding folding pathways (e.g., the idea implied by Levinthal's paradox that because proteins fold to their native states rapidly in nature, they must not do so by sequential random sampling, since such an approach would not be tractable, given the combinatorially large conformational state space, but are instead guided by a series of intermediate states which serve to minimize Gibbs free energy subject to kinetic constraints which may result in local minima) [4]. Two computational challenges which dominate work in this space include: (1) the difficulty of identifying and efficiently sampling from promising regions in the vast conformational state space, and (2) accurately modeling energy-related characteristics/dynamics of candidate structures [7, 1].

In recent years, deep learning-based approaches have helped to advance the state of the art, as demonstrated by the success of such approaches on the template-free (ab initio) modeling portion of the Critical Assessment of Structure Prediction (CASP) benchmark tests [6, 10]. AlphaFold is one recent example; this pipeline feeds protein amino-acid residue sequences and MSA features to a convolutional neural network (CNN), which outputs pairwise residue distance and backbone torsion angle distribution predictions. These distributional predictions are in turn used to compute the input protein-specific potential of mean force (PMF), which can be minimized via gradient descent to predict the 3-D structure [8]. Another example is trRosetta, which also takes MSA features as input to a CNN but predicts pairwise inter-residue geometric orientations and incorporates the Rosetta energy function into the optimization and downstream structure prediction step [11].

We note that the input in this space is often a set of protein amino-acid residue sequence(s), and many intermediate prediction tasks are pairwise; as such, dense computation is involved. With respect to pipeline composition, many inference workflows in this space, including those profiled above, contain three high-level steps, including: (1) feature

[1] A protein's 3-D structure is also referred to as its tertiary structure within the literature.

extraction; (2) DNN-based inference; and (3) prediction and/or realization of 3D-structure. Each of these pipeline steps has distinct computational characteristics, which may make some pipeline components more suitable for porting to SDH hardware than others.

Protein folding mechanisms and pathways While the protein structure prediction task outlined in the preceding section focuses on predicting a protein's native state, the related task of identifying folding mechanism(s) focuses on identifying and/or predicting a protein's sequence of intermediate structural transitions (pathways), and the kinetic/topological mechanisms driving these transitions [7]. Atomistic molecular dynamics (MD) simulations provide high-resolution information about the transition process, but require large amounts of computational power, since it is desirable to simulate as many time steps and trajectories as possible [2]. Sampling from a large state space and accurate modeling of the potential energy function are persistent challenges. The calculations required to run these simulations involve floating point operations on large arrays; as such, they are parallelizable, and have benefitted from a tremendous amount of computational resources in recent years, in the form of GPUs, supercomputers, specialized ASICs, and crowd-sourcing applications, such as Folding@Home [2].

Folding@Home works by crowd-sourcing idle CPU and GPU cycles from users; these cycles are used to complete molecular dynamics simulations of protein folding kinetics, as well as to study protein misfolding, and how it relates to specific cancers, infectious, and neurological diseases and associated drug development efforts. [2] These molecular dynamics siumlations involve Markov State Models (MSMs), in which states correspond to protein conformations, and the transition matrix conveys information about the transition rates between states.

The distributed nature of the project allows for the set of simulations (called Runs) associated with a particular protein and initial conformation (selected via adaptive sampling), to be partitioned into a set of trajectories called Clones, each of which feature a different initial velocity. These clones are then further segmented into short sub-trajectories to ensure tractability for individual users. Given the temporal nature of the simulation, clones are sequentially dependent, and users are incentivized via gamification to complete a single assigned trajectory within a specific time frame (e.g., before receiving a new trajectory to process). The results are aggregated across users for downstream analysis. [3]

Over the years, Folding@Home's efforts have lead to more than 200 peer-reviewed papers in high-impact journals, and many simulation results have been empirically verified. [4] Notable areas of research which Folding@Home has helped to support include: (1) protein folding simulations followed by quantitative validation/comparison to experimental results; (2) computational drug design; (3) development of distributed approaches to simulating folding; and (4) disease-specific applications related to cancer, Alzheimer's Disease, and viral infections (e.g., Zika and Ebola). [5]

With respect to performance, Folding@Home reports metrics of their distributed network broken down by OS on a rolling basis. Table 1 provides the most up-to-date statistics available at the time of this report, which were released on May 21, 2020, and include CPUs and GPUs which have returned Work Units (WU) within the last 50 days. [6] Note that in this table, FLOPS per core are estimated, and x86 FLOPS represent how many FLOPS it would take to compute calculations empirically computed on GPUs on x86 class CPUs instead. [7].

Table 1: Folding@Home Active CPUs & GPUs by OS: Counts and Estimated FLOPS

OS	AMD GPUs	NVidia GPUs	CPUs	CPU cores	TFLOPS	x86 TFLOPS
Windows	88,577	307,012	751,561	4,612,375	698,301	1,408,394
Linux	15,978	118,771	3,365,026	24,878,893	559,739	805,477
macOSX	172	0	86,709	402,366	4,752	5,024
Totals	104,727	425,783	4,203,296	29,893,634	1,262,792	2,218,895

The energy consumption of the Folding@Home network is dynamic, in that it is a function of the size and device composition of the distributed network at any particular point in time. The COVID-19 pandemic has renewed interest in the platform, resulting in a sharp uptick in the number of active users, which allowed Folding@Home to break the

[2] https://foldingathome.org/support/faq/press-info/
[3] https://foldingathome.org/dig-deeper/
[4] https://foldingathome.org/papers-results/
[5] https://foldingathome.org/support/faq/press-info/
[6] https://stats.foldingathome.org/os
[7] https://foldingathome.org/support/faq/flops/

exaFLOP barrier in March. [8] [9] The project documents estimate that the difference between running Folding@Home 24-hours a day on a Pentium-type computer with the monitor off and having the computer turned off is approximately 100 watts per hour, for a total of 2.4 kWh. [10]

Protein sequence prediction Protein sequence prediction is often referred to as the inverse protein folding problem, in that it focuses on predicting a protein's amino acid residue sequence from its 3-D structure for the purpose of biomedical and/or materials engineering [5]. One example of recent work in this space is Ingrahm et al., who represent 3-D protein structures as graphs over the amino acid residues, and develop a transformer-based model to generate protein sequences conditional on the connectivity, topology, and atomic information of the input graph [5]. This type of approach does contain hybrid dense-sparse computations; however, additional investigation would be required to determine whether or not the scale of problem instances, and/or intermediate memory access patterns poses HIVE/SDH suitability challenges/opportunities.

References

[1] Biswas, S., G. Kuznetsov, P. J. Ogden, N. J. Conway, R. P. Adams, and G. M. Church (2018). Toward machine-guided design of proteins. *bioRxiv*, 337154.

[2] Freddolino, P. L., C. B. Harrison, Y. Liu, and K. Schulten (2010). Challenges in protein folding simulations: Timescale, representation, and analysis. *Nature physics 6 10*, 751–758.

[3] Haubold, B. and T. Wiehe (2006). *Introduction to Computational Biology: An Evolutionary Approach*. Birkhauser.

[4] Hunter, P. (2006, 03). Into the fold. advances in technology and algorithms facilitate great strides in protein structure prediction. *EMBO reports 7*(3), 249–252.

[5] Ingraham, J., V. Garg, R. Barzilay, and T. Jaakkola (2019). Generative models for graph-based protein design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32*, pp. 15820–15831. Curran Associates, Inc.

[6] Kryshtafovych, A., T. Schwede, M. Topf, K. Fidelis, and J. Moult (2019). Critical assessment of methods of protein structure prediction (casp)round xiii. *Proteins: Structure, Function, and Bioinformatics 87*(12), 1011–1020.

[7] Li, B., M. Fooksa, S. Heinze, and J. Meiler (2018, February). Finding the needle in the haystack: towards solving the protein-folding problem computationally. *Critical reviews in biochemistry and molecular biology 53*(1).

[8] Senior, A. W., R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. ÅœÃdek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis (2020). Improved protein structure prediction using potentials from deep learning. *Nature 577*(7792), 706–710.

[9] Singh, A. (2020a). Deep learning 3d structures. *Nature Methods 17*(3), 249–249.

[10] Singh, A. (2020b). Deep learning 3D structures. *Nature Methods 17*(3), 249–249.

[11] Yang, J., I. Anishchenko, H. Park, Z. Peng, S. Ovchinnikov, and D. Baker (2019). Improved protein structure prediction using predicted inter-residue orientations. *bioRxiv*, 846279.

[12] Zeng, Z., S. Hua, Y. Wu, and Z. Hong (2015, 10). Survey of natural language processing techniques in bioinformatics. *Computational and Mathematical Methods in Medicine 2015*, 1–10.

[8] https://blogs.nvidia.com/blog/2020/04/01/foldingathome-exaflop-coronavirus/

[9] https://arstechnica.com/?post_type=post&p=1667949

[10] https://foldingathome.org/support/faq/miscellaneous/