A Hybrid Attributed Generic Graph Library Environment

Andrew Lumsdaine (PI), John Feo (PM) , Giovanni Castellana (PNNL)

Architectures Thrust: Hierarchical Identify Verify Exploit (HIVE)

Improve analyst productive while maximizing performance on purpose-built systems

Break the owner-compute, superstep programming model

Shared memory, hybrid data structures; dynamic data/task scheduling; latency tolerant data movement

High-productivity, high-performance, SCALABILITY

Principal features

Extensible and generic SDK with three levels of graph primitives

Support for static, persistent, dynamic, and streaming data

Hybrid data model --- relational tables, property graphs, edge matrices

Extended Abstract Graph Machine to reason about data flow, data locality, and task scheduling

Control data flow intermediate representation for code transformation, optimization, and scheduling with introspection

Abstract runtime model supporting a variety of memory and execution models

ARTS

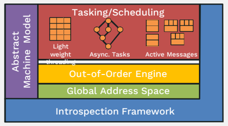
Efficient, a/synchronous resource aware task scheduling

Global address space to facilitate data movement

Dynamic dependency based synchronization

Light-weight multi-threading

- Active messages
- Asynchronous put/get/AMOs



Flexible distributed coherency protocols to trade-off data duplication with data movement

Introspection framework to support performance analysis and adaptive execution

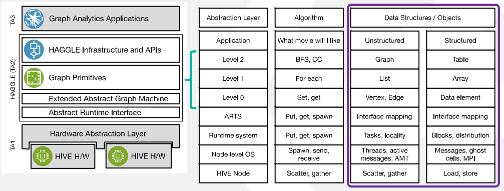
HAGGLE architecture

HAGGLE is organized in layers.

SHAD, the Scalable High Performance Algorithms and Data structures Library, provides flexible, high-performance data structures and methods

The Extended Abstract Graph Machine (EAGM) applies code transformations to optimize data and task mappings

The Abstract Runtime System (ARTS) implements data movements, task scheduling and control operations for the unique hardware features of target systems

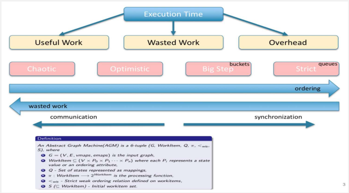

TA3
Graph Analytics Applications
HAGGLE (TA2)
HAGGLE Infrastructure and APIs
Graph Primitives
Extended Abstract Graph Machine
Abstract Runtime Interface
TA1
Hardware Abstraction Layer
HIVE H/W
HIVE H/W

Abstraction Layer	Algorithm	Data Structures / Objects	
Application	What movie will I like	Unstructured	Structured
Level 2	BFS, CC	Graph	Table
Level 1	For each	List	Array
Level 0	Set, get	Vertex, Edge	Data element
ARTS	Put, get, spawn	Interface mapping	Interface mapping
Runtime system	Put, get, spawn	Tasks, locality	Blocks, distribution
Node level OS	Spawn, send, receive	Threads, active messages, AMT	Messages, ghost cells, MPI
HIVE Node	Scatter, gather	Scatter, gather	Load, store

EAGM

AGM model decomposes graph algorithms into processing functions and strict weak ordering of work items → algorithm taxonomy

EAGM model describes hybrid hierarchical algorithms using different orderings at different algorithmic levels (to match hardware performance)



Definition

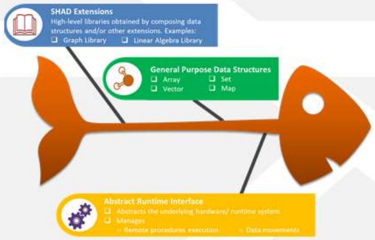
An Abstract Graph Machine(AGM) is a 6-tuple (G, $WorkItem$, Q, π, $<_{wis}$, S), where

- $G = (V, E, vmaps, emaps)$ is the input graph,
- $WorkItem \subseteq (V \times P_0 \times P_1 \cdots \times P_n)$ where each P_i represents a state value or an ordering attribute,
- Q - Set of states represented as mappings,
- $\pi : WorkItem \longrightarrow 2^{WorkItem}$ is the processing function,
- $<_{wis}$ - Strict weak ordering relation defined on workitems,
- S ($\subseteq WorkItem$) - Initial workitem set.

SHAD

SHAD is a C++ library of data structures and algorithms that hides the complexity of programming a distributed system.

SHAD implements HAGGLE's Hybrid Data Model and supports dynamic insertion and removal of elements from the data structures.

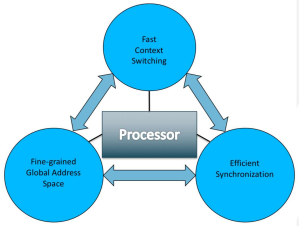

SHAD Extensions
High-level libraries obtained by composing data structures and/or other extensions. Examples:
❑ Graph Library
❑ Linear Algebra Library
General Purpose Data Structures
❑ Array
❑ Vector
❑ Set
❑ Map
Abstract Runtime Interface
❑ Abstracts the underlying hardware/ runtime system
❑ Manages
○ Remote procedures execution
○ Data movements

HAGGLE HARDWARE ASKS

Fast context switching: latency tolerance for unpredictable data accesses

Fine-grained global address space: no partitioning, simplifies code development

Efficient synchronization: reduces hotspots with synchronization intensive workloads



This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).
The views, opinions and/or findings are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Distribution Statement A – Approved for Public Release, Distribution Unlimited

Pacific Northwest NATIONAL LABORATORY

THE ELECTRONICS RESURGENCE INITIATIVE