Towards a GraphBLAS Library in Chapel

Ariful Azad & Aydin Buluç
Lawrence Berkeley National Laboratory (LBNL)

CHIUW, IPDPS 2017
High-level research objective:

- Enable **productive** and **high-performance** graph analytics
- We used GraphBLAS and Chapel to achieve this goal

GraphBLAS:
Building blocks for graph algorithms in the language of sparse linear algebra

Chapel:
An emerging parallel language designed for productive parallel computing at scale

Both promise: Productivity + Performance

Scope of this paper: A GraphBLAS library in Chapel
1. Overview of GraphBLAS primitives
2. Implementation of a subset of GraphBLAS primitives in Chapel with experimental results

Warning: this is just an early evaluation as Chapel’s sparse matrix support is actively under development. All experiments were conducted on Chapel 1.13.1. The performance numbers are expected to improve significantly in future releases of Chapel.
Part 1. GraphBLAS overview
GraphBLAS analogy
A ready-to-assemble furniture shop (Ikea)

Building blocks

Objects (Algorithms)

Final product (Applications)
Graph algorithm building blocks

- GraphBLAS (http://graphblas.org)
  - Standard building blocks for graph algorithms in the language of sparse linear algebra
  - Inspired by the Basic Linear Algebra Subprograms (BLAS)
  - Participants from industry, academia and national labs
  - C API is available in the website

*(Design of the GraphBLAS API for C, A Buluç, T Mattson, S McMillan, J Moreira, C Yang, IPDPS Workshops 2017)*
Employs graph-matrix duality
- Graphs => sparse matrix
- A subset of vertex/edges => sparse/dense vector

Benefits
- Standard set of operations
- Learn from the rich history of numerical linear algebra
- Offers **structured and regular memory accesses and communications** (as opposed to irregular memory accesses in tradition graph algorithm)
- Opportunity for **communication avoiding algorithms**
### Some GraphBLAS basic primitives

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Matlab notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxM (SpGEMM)</td>
<td>- sparse matrices A and B</td>
<td>sparse matrix</td>
<td>C = A * B</td>
</tr>
<tr>
<td></td>
<td>- optional unary functs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MxV (SpM{Sp}V)</td>
<td>- sparse matrix A</td>
<td>sparse/dense vector</td>
<td>y = A * x</td>
</tr>
<tr>
<td></td>
<td>- sparse/dense vector x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EwiseMult, Add, ... (SpE-wiseX)</td>
<td>- sparse matrices or vectors</td>
<td>in place or sparse matrix/vector</td>
<td>C = A .* B</td>
</tr>
<tr>
<td></td>
<td>- binary funct, optional unarys</td>
<td></td>
<td>C = A + B</td>
</tr>
<tr>
<td>Reduce (Reduce)</td>
<td>- sparse matrix A and funct</td>
<td>dense vector</td>
<td>y = sum(A, op)</td>
</tr>
<tr>
<td>Extract (SpRef)</td>
<td>- sparse matrix A</td>
<td>sparse matrix</td>
<td>B = A(p, q)</td>
</tr>
<tr>
<td></td>
<td>- index vectors p and q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assign (SpAsgn)</td>
<td>- sparse matrices A and B</td>
<td>none</td>
<td>A(p, q) = B</td>
</tr>
<tr>
<td></td>
<td>- index vectors p and q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BuildMatrix (Sparse)</td>
<td>- list of edges/triples (i, j, v)</td>
<td>sparse matrix</td>
<td>A = sparse(i, j, v, m, n)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExtractTuples (Find)</td>
<td>- sparse matrix A</td>
<td>edge list</td>
<td>[i, j, v] = find(A)</td>
</tr>
</tbody>
</table>

- MxM: Matrix-Matrix multiplication
- MxV: Matrix-Vector multiplication
- EwiseMult: Element-wise multiplication
- Reduce: Reduction operation
- Extract: Extraction of matrix elements
- Assign: Assignment to matrix elements
- BuildMatrix: Construction of a sparse matrix from edges/triples
- ExtractTuples: Extraction of edge lists from a sparse matrix
### General purpose operations via semirings (overloading addition and multiplication operations)

<table>
<thead>
<tr>
<th>Real field: ((R, +, \times))</th>
<th>Classical numerical linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean algebra: (({0, 1}, \mid, &amp;))</td>
<td>Graph traversal</td>
</tr>
<tr>
<td>Tropical semiring: ((R \cup {\infty}, \min, +))</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>((S, \text{select}, \text{select}))</td>
<td>Select subgraph, or contract nodes to form quotient graph</td>
</tr>
<tr>
<td>(edge/vertex attributes, vertex data aggregation, edge data processing)</td>
<td>Schema for user-specified computation at vertices and edges</td>
</tr>
<tr>
<td>((R, \max, +))</td>
<td>Graph matching &amp; network alignment</td>
</tr>
<tr>
<td>((R, \min, \text{times}))</td>
<td>Maximal independent set</td>
</tr>
</tbody>
</table>

- **Shortened semiring notation:** \((\text{Set, Add, Multiply})\). Both identities omitted.
- **Add:** Traverses edges, **Multiply:** Combines edges/paths at a vertex
Example: Exploring the next-level vertices via SpMSSpV

![Graph and Adjacency Matrix]

Overload (multiply, add) with (select2nd, min)

Current frontier

Next frontier

Adjacency matrix
Algorithmic coverage

Higher-level combinatorial and machine learning algorithms

- **Miscellaneous:** connectivity, traversal (BFS), independent sets (MIS), graph matching
- **Classification:** (support vector machines, Logistic regression)
- **Graph clustering:** (Markov cluster, peer pressure, spectral, local)
- **Centrality:** (PageRank, betweenness, closeness)
- **Dimensionality reduction:** (NMF, PCA)
- **Shortest paths:** (all-pairs, single-source, temporal)

GraphBLAS primitives in increasing arithmetic intensity

- Develop high-performance algorithms for 10-12 primitives.
- Use them in many algorithms (**boost productivity**).
Expectation: two-layer productivity

Graph algorithms

GraphBLAS operations

Chapel’s productivity features

use

use

use

user space

library

language
Part 2. Implementing a subset of GraphBLAS operations in Chapel
## For Chapel: A subset of GraphBLAS operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameters</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apply</strong></td>
<td>$x$: sparse matrix/vector</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>$f$: unary function</td>
<td></td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>$x$: sparse matrix/vector</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>$y$: sparse matrix/vector</td>
<td></td>
</tr>
<tr>
<td><strong>eWiseMult</strong></td>
<td>$x$: sparse matrix/vector</td>
<td>$z$: sparse</td>
</tr>
<tr>
<td></td>
<td>$y$: sparse matrix/vector</td>
<td>matrix/vector</td>
</tr>
<tr>
<td><strong>SpMSpV</strong></td>
<td>$A$: sparse matrix</td>
<td>$y$: sparse</td>
</tr>
<tr>
<td></td>
<td>$x$: sparse vector</td>
<td>vector</td>
</tr>
</tbody>
</table>
Chapel details
- Chapel 1.13.1 (the latest version before the IPDPS deadline)
- Chapel built from source
- CHPL_COMM: gasnet/gemini
- Job launcher: slurm-srun

Experiment platform: NERSC/Edison
- Intel Ivy Bridge processor
- 24 cores on 2 sockets
- 64 GB memory per node
- 30-MB L3 Cache
Sparse matrices in Chapel

- Block distributed sparse matrices. The dense container is block distributed.
- We used compressed sparse block (CSR) layout to store local matrices.

```chapel
var n = 6
const D = {0..n-1, 0..n-1}
    dmapped Block(1..3,1..3);
var spD: sparse subdomain(D);
var A = [spD] real;
```

In this example: #locales = 9

In our results, we did not include time to construct arrays.
The simplest GraphBLAS operation: **Apply** \( (x[i] = f(x[i])) \)

---

// Implementing apply() using forall loop

```chapel
proc Apply1(spArr, unaryOp)
{
    forall a in spArr do
        a = unaryOp(a);
}
```

**Apply1**: high-level (Chapel style)

---

// Implementing apply() with local arrays

```chapel
proc Apply2(spArr, unaryOp)
{
    var locArrs = spArr._value.locArr;
    coforall locArr in locArrs do
        on locArr {
            forall a in locArr.myElems do
                a = unaryOp(a);
        }
}
```

**Apply2**: manipulating internal arrays (MPI style)
Example, simple case: \textbf{Apply} \ (x[i] = f(x[i]))

\textbf{Apply1}: high-level (Chapel style)
\textbf{Apply2}: manipulating internal arrays (C++ style)

x: 10M nonzeros
Platform: NERSC/Edison

Data parallel loops perform well in shared memory
But do not perform well in distributed memory
Performance on distributed-memory

Using chplvis on four locales
Red: data in, blue: data out

All work at locale 0

This issue with sparse arrays has been addressed about a week ago
Assign \( x[i] = y[i] \)

**Assign1**: high-level (Chapel style)

```plaintext
proc Assign1(A: [?DA], B: [?DB]) {
  
  //------ Assign domain ------
  DA.clear(); // destroy A
  DA += DB;
  //------ Assign array ------
  forall i in DA do
    A[i] = B[i];
}
```

**Assign2**: manipulating internal arrays (MPI style)

```plaintext
proc Assign2(A: [?DA], B: [?DB]) {
  DA.clear(); // destroy A
  if(DB.size == 0) then return;
  //------ Assign domain ------
  var locDAs = DA._value.locDoms;
  var locDBs = DB._value.locDoms;
  coforall (locDA, locDB) in zip(locDAs, locDBs) do
    on locDA {
      locDA.mySparseBlock += locDB.mySparseBlock
    }
  }
```
Shared-memory performance: Assign \((x[i] = y[i])\)

Assign1: high-level (Chapel style)
Assign2: manipulating internal arrays (C++ style)

x: 1M nonzeros
Platform: NERSC/Edison

Big performance gap
Even in shared memory

Why?
Indexing a sparse domain uses binary search. For assignment it can be avoided
distributed-memory performance: Assign \( (x[i] = y[i]) \)

**Assign1**: high-level (Chapel style)
**Assign2**: manipulating internal arrays (C++ style)

x: 1M nonzeros
Platform: NERSC/Edison

Big performance gap
Even in distributed memory
Example, complex case: SpMSpV \( (y = Ax) \)

Algorithm overview

\[ y = \text{gather} \left( \text{SPA} \right) \]

\[ A \text{ \* } \text{scatter/accumulate} \]

\[ A \quad x \]
Sparse matrix-sparse vector multiply (SpMSpV)

**Algorithm (MPI Style)**
1. Gather vertices in processor column
2. Local multiplication
3. Scatter results in processor row

**Algorithm (Chapel Style)**
Multiply (access remote data as needed). No collective communication

P processors are arranged in \( \sqrt{p} \times \sqrt{p} \) Processor grid
Distributed-memory performance of SpMSpV on Edison

A: random; 16M nonzeros  x: random; 2000 nonzeros

Remote atomics are expensive in Chapel
Requirements for achieving high performance

- Exploit available **spatial locality** in sparse manipulations
  - Efficient access of nonzeros of sparse matrices/vectors
  - Chapel is almost there, needs improved parallel iterators

- Use **bulk-synchronous communication** whenever possible
  - Avoid latency-bound communication
  - Team collectives are useful
## Our experience: productivity vs. performance

### Productivity (easy to develop a prototype)

<table>
<thead>
<tr>
<th>Task</th>
<th>Hardness</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data structure</td>
<td>medium</td>
<td>Manipulating domains and arrays</td>
</tr>
<tr>
<td>Functionality</td>
<td>easy</td>
<td>Fewer lines of code with built-in features</td>
</tr>
<tr>
<td>Parallelization</td>
<td>easy</td>
<td>No need to think about communication</td>
</tr>
</tbody>
</table>

### Performance (hard to achieve performance)

<table>
<thead>
<tr>
<th>Task</th>
<th>Hardness</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data structure</td>
<td>hard</td>
<td>Manipulating low level data structures</td>
</tr>
<tr>
<td>Shared-memory</td>
<td>medium</td>
<td>Data parallel iterators for sparse data</td>
</tr>
<tr>
<td>Distributed-memory</td>
<td>hard</td>
<td>Needs bulk synchronous communication, team collectives, etc.</td>
</tr>
</tbody>
</table>
We have implemented a prototype GraphBLAS library in Chapel
- Implemented breadth-first search as a representative algorithm using these primitives

Library development in Chapel is easy (relative to C++)

Chapel’s distributed-sparse matrix support is still under development. The distributed-memory performance is expected to improve over time.
Future direction

- Finish a complete GraphBLAS-compliant library in a PGAS language (including Chapel)
  - Achieving high performance is our focus
  - Benchmark our library against other programming models and languages

- Design complex graph algorithms using the library to demonstrate its utility
  - Understand the impact of programming models on graph analytics
Funded in part by DOD/ACS and in part by DOE/ASCR

Acknowledgement: Costin Iancu (LBNL), Brad Chamberlain (Cray), Michael Ferguson (Cray), Engin Kayraklioglu (George Washington University)

References:
- A. Azad and A. Buluç, IPDPS Workshops 2017, Towards a GraphBLAS library in Chapel.

Questions?