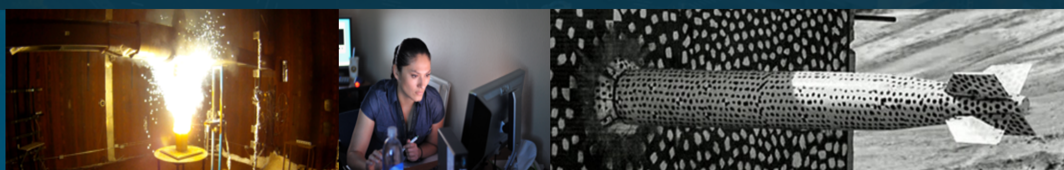
Chapel performance for some graph analytics



Richard Barrett

ICL 2020

September 11, 2020

SAND 2020-9258 PE

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Overview

My brief history of supercomputing

Overview of the Chapel programming language

Computing graph hitting time moments

Triangle enumeration

Goal: understand Chapel's performance capabilities.

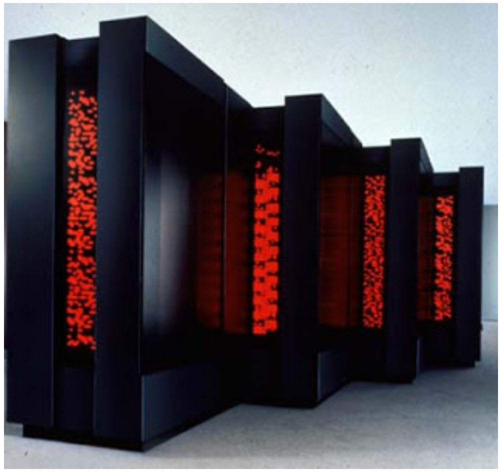
My brief history of supercomputing

We used to have things called supercomputers.

Integrated computing environment designed for modeling and simulation.

Cray 1, serial #1@LASL 1976
1 64-bit processor
160 MFLOPS, 8 MB
80 MHz, 115 kW
303 MB disk

Last was the TMC CM-5: OS, language/compiler (F90), runtime, tools (!)



TMC CM-5 @ LANL 1993
1024 x SuperSparc processors
597 GFLOPS, #1 Top500

My brief history of supercomputing

DOE 1995 ASCI program defined COTS HPC machines.

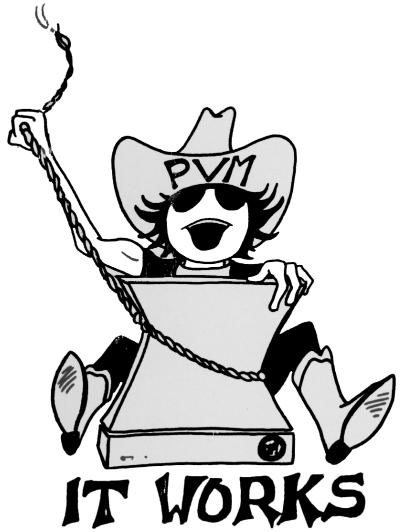
- I was hired at LANL 6/94 essentially to help them transition from Cray vector machines.

- Lots of cool research trying to make them supercomputers: Cellular IRIX, file systems, fault tolerance, etc.

"Anyone can build a fast CPU.

The trick is to build a fast system." Seymour Cray

- Hardware, OS, compilers, languages, libraries, tools, ...



1996 PVM UG@Santa Fe

The Cray 1 delivered to LASL as a slab of hardware. LASL/LANL wrote OS, scheduler, compiler, debugger, etc.

DARPA HPCS 2003
High Productivity Computing Systems

Productivity : time from idea to solution

Characteristics of a productive language

- Expressive
- Performance
- Portability
- Extensible

Importance of each as a function of your problem

Chapel Programming Language

Cray High Performance Language https://chapel-lang.org/

Informed by ZPL, HPF, MTA/XMT C and Fortran extensions, C++

Key construct : domain

- global view of a data structure and distribution
- first class index set

data parallel : forall i in Domain { }

task parallel : coforall { }, cobegin { }, begin { }

“The Rise and Fall of High Performance Fortran: an Historical Object Lesson”, Kennedy, Koelbel, and Zima, 2007.

Chapel domain

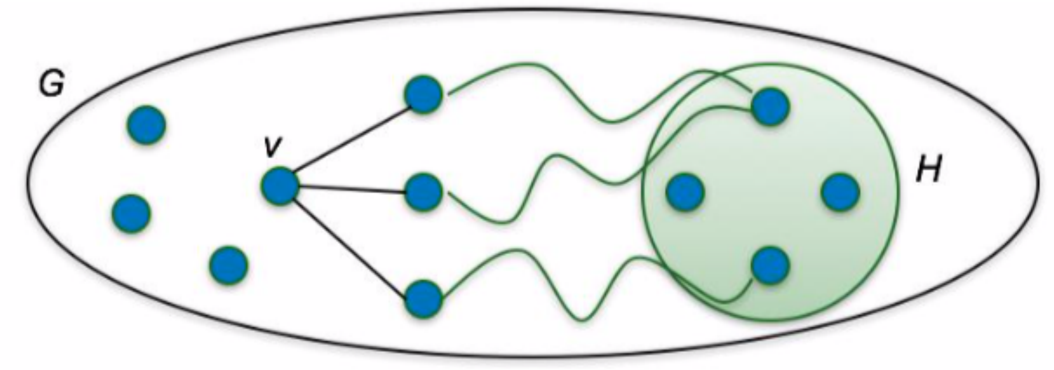
```
var MyDom = {1..n} ; dmapped Block ( boundingBox={1..n} );

var x, y : [MyDom] real;
x += alpha * y;
resid = + reduce x**2;

forall i in A.dom_nnz {          // Matrix-vector product
   y[A.rowidx[i]] += x[A.colidx[i]];
}
forall i in SomeDomain {               // Stencil is also a domain
   y[i] = ( + reduce [ k in Stencil ] coeff ( i + k ) * x ( i + k );
 }
```

Graph hitting time

- A random variable for the number of (Markov chain) steps to reach a set of hitting set vertices *H* of a graph *G*

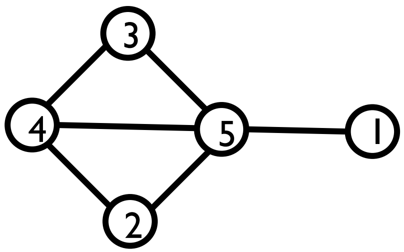


- Compute random variable distribution, i.e., the hitting time moments : mean, standard deviation, skew, and kurtosis.

"Advantages to modeling relational data using hypergraphs versus graphs", Wolf, Klinvexm, and Dunlavy, IEEE HPEC, 2016.

Setting up linear system

G :



Simple undirected graph

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Adjacency matrix

Configured as linear system* : $(D - A)\, x_k = f(D, A, x_{k-1})$

for D = diagonal matrix of vertex degrees, x = moments

where x_1 mean, x_2 standard deviation, x_3 skew, x_4 kurtosis

Solved using the *Conjugate Gradient* algorithm

- Key kernel: *matrix-vector product*

* Rich Lehoucq, report in progress.

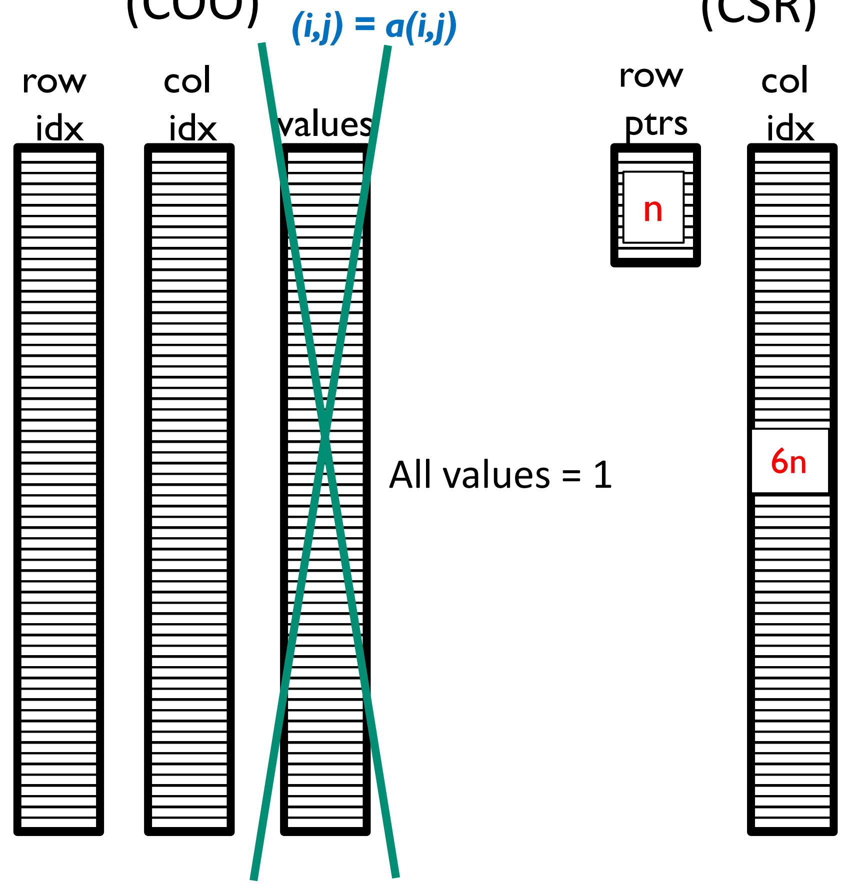
Storing the sparse matrix

Chapel sparse domain

- Define dense domain
- Define subset of it: sparse domain
- Not (yet) performant
- Using for triangle enumeration in unique way

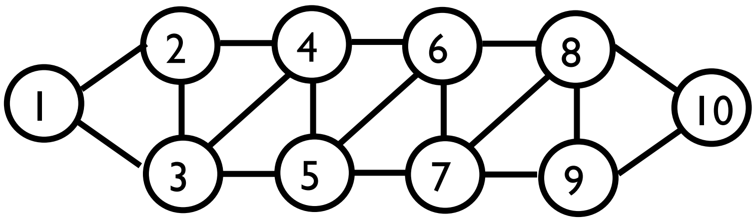
Coordinate storage (COO)

Row compressed (CSR)

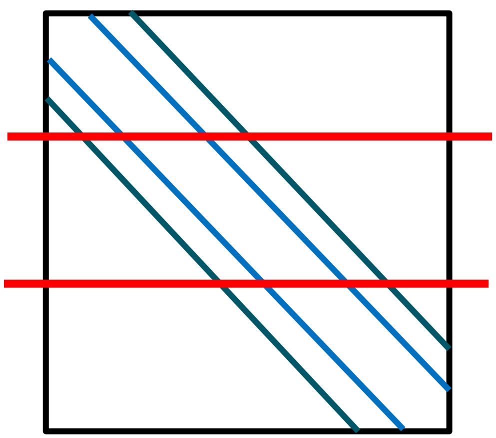


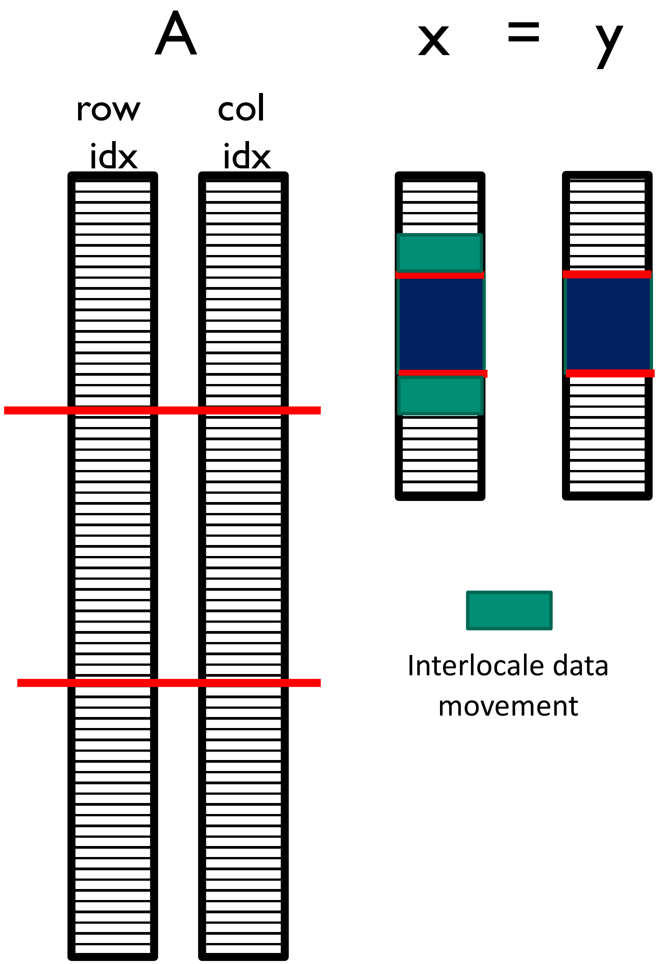
Example: banded matrix A, in COO format

G :



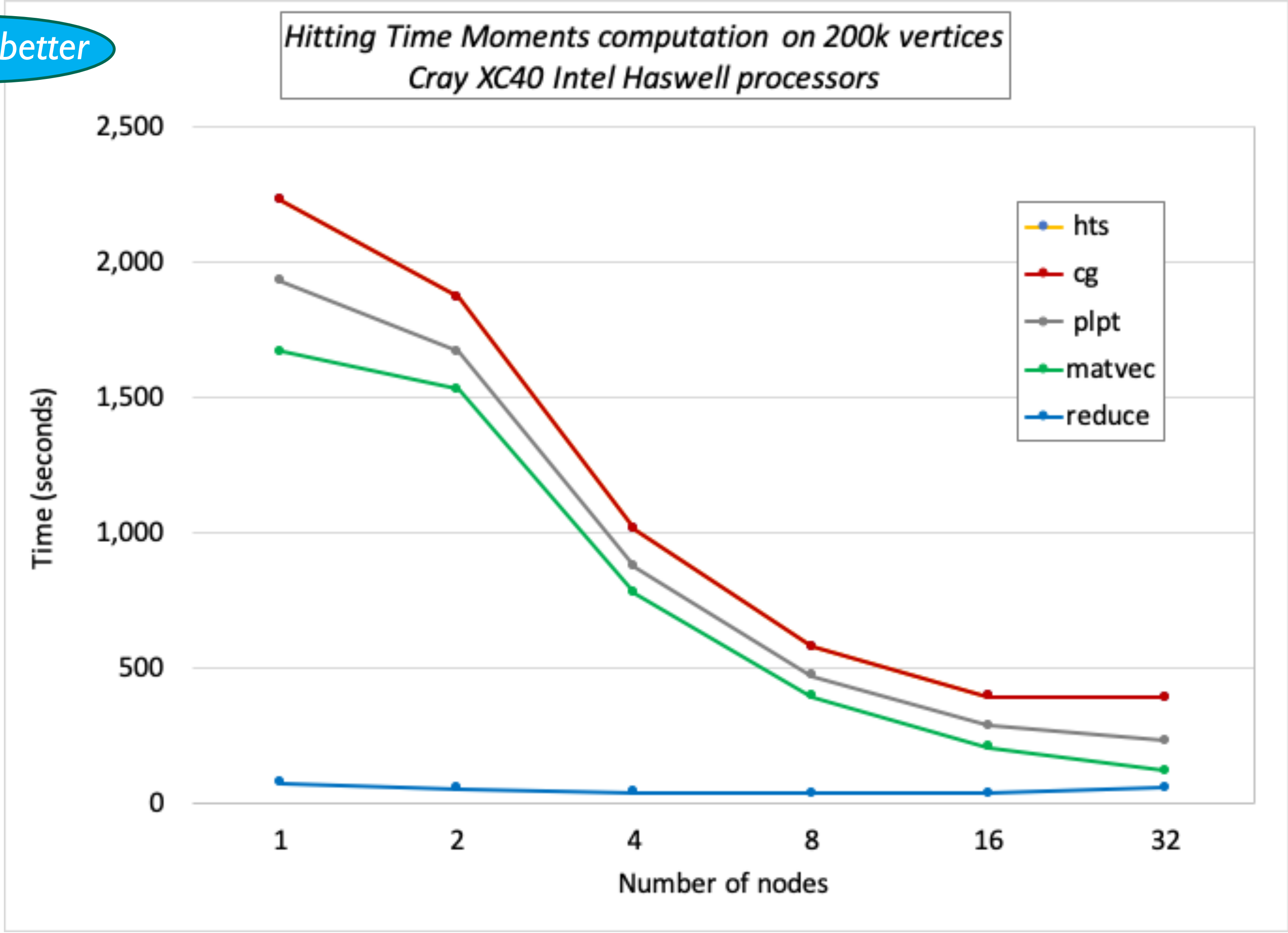
A





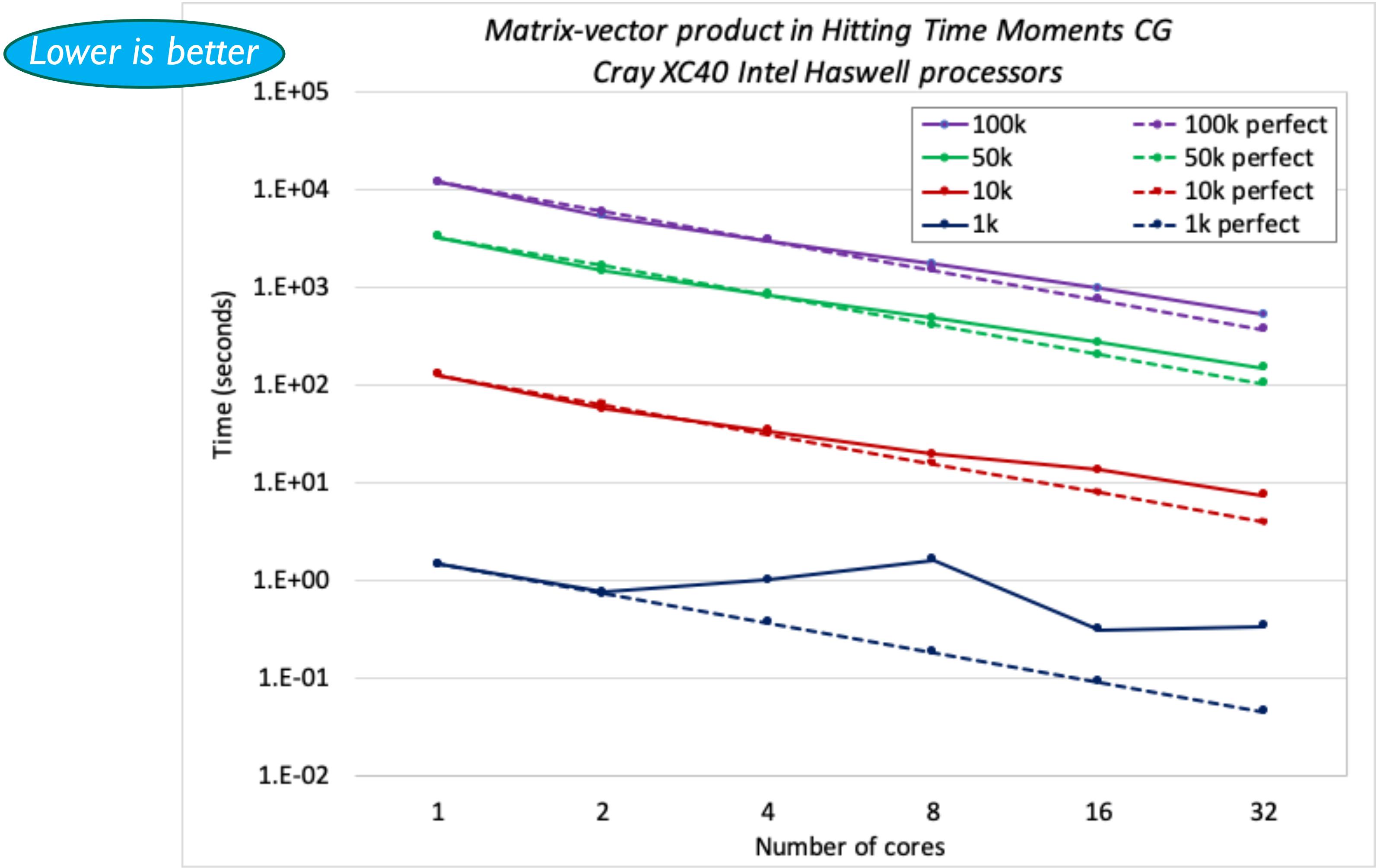
Strong scaling*

Lower is better



* qthreads + ugni

MatVec: single node* strong scaling



* qthreads runtime

Arkouda

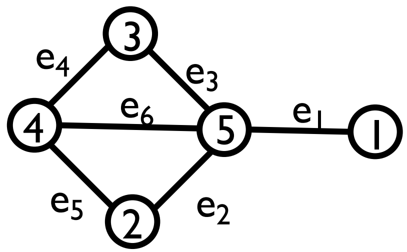
Jupyter-like notebook, launching jobs onto HPC using Chapel.

Based on numpy-like arrays

Implemented a gather/scatter capability, called “aggregation"

https://github.com/mhmerrill/arkouda

Triangle enumeration
Key computation: sparse MatMat



Adjacency matrix — Edge Incidence matrix

vertex — vertex — edge — vertex

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & \boxed{1} & 1 & \boxed{1} & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \boxed{1} & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \boxed{1} & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{2,4,5\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{3,4,5\} \\ \emptyset & \{4,2,5\} & \{4,3,5\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{5,3,4\} & \boxed{\{5,2,4\}} & \emptyset \end{pmatrix}$$

“A Task-Based Linear Algebra Building Blocks Approach for Scalable Graph Analytics,”, Wolf, Stark, and Berry, IEEE HPEC 2015.

Performance Tools

CrayPat

- No longer supports Chapel.

Tau

- Single node support

HPCToolKit

- Returns profile with missing function names, even when compiling with -g

LDMS

- PAPI sampler runs with Chapel code, but gives ‘0’ for all data collected.
- Network samplers should work to show communication (TBD).

ChplBlamer

- Academic tool from University of Maryland (Jeff Hollingsworth); supported?

Summary

Scaling performance currently pretty good. Arkouda "aggregation" better?

We're assuming no known graph structure.

Exploring various matrix storage formats:

- COO, CSR, Chapel sparse domain

User supplied Chapel operator capability.

Need better tools!

Future work

- Matrix "in place" implementation, to support full application.
- Additional processors, eg ARM, GPU and interconnects.