Sandia National Laboratories

Exploring Chapel Productivity Using Some Graph Algorithms

Richard Barrett

Chapel Implementors and Users Workshop 2020

May 22, 2020

SAND 2020 – 5317 C

ENERGY NNSA

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.

Team

Omar Aaziz : node performance analysis

Richard Barrett : application development

Jeanine Cook : node performance analysis

Chris Jenkins : architecture

Stephen Oliver : runtime systems

Courtenay Vaughan : distributed memory performance analysis

Overview

Investigating Chapel performance for some linear algebra based graph analytics

Compute hitting time moments and triangle enumeration.

Sparse matrix-vector and matrix-matrix multiplication.

Compare with existing implementations

- Grafiki hitting time : C++/Kokkos/MPI
  - "Advantages to modeling relational data using hypergraphs versus graphs", Wolf, Klinvexm, and Dunlavy, IEEE HPEC, 2016.
- miniTri : C++/OpenMP/MPI
  - "A Task-Based Linear Algebra Building Blocks Approach for Scalable Graph Analytics,", Wolf, Stark, and Berry, IEEE HPEC 2015.

Outline

Graph hitting time

Key computation

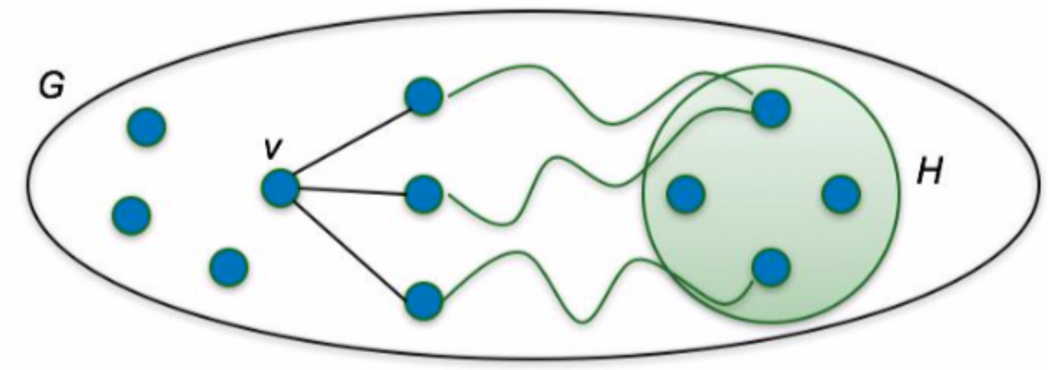
Performance

Preview of triangle enumeration

Summary

Graph hitting time

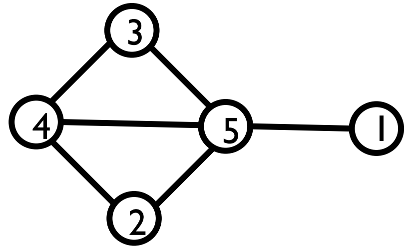
- A random variable for the number of (Markov chain) steps to reach a set of hitting set vertices *H* of a graph *G*



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

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

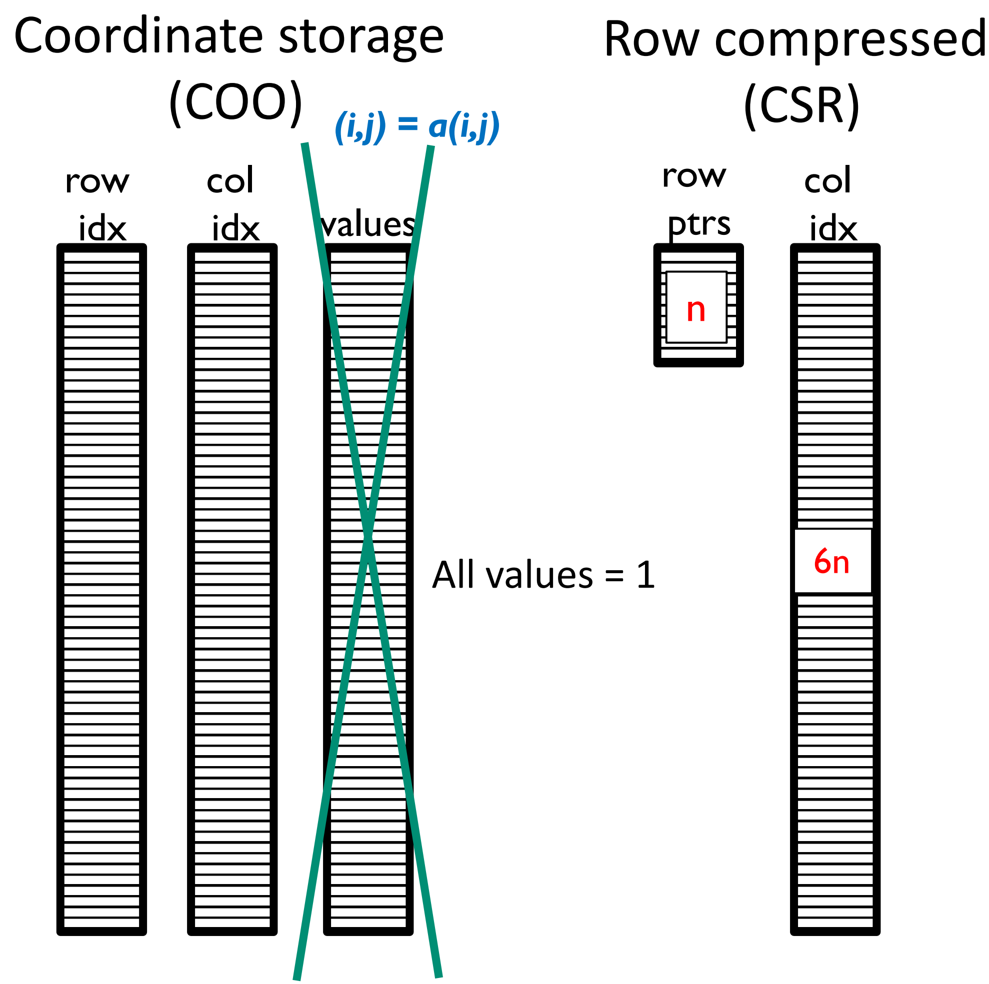
- Key kernel: *matrix-vector product*

Storing the sparse matrix

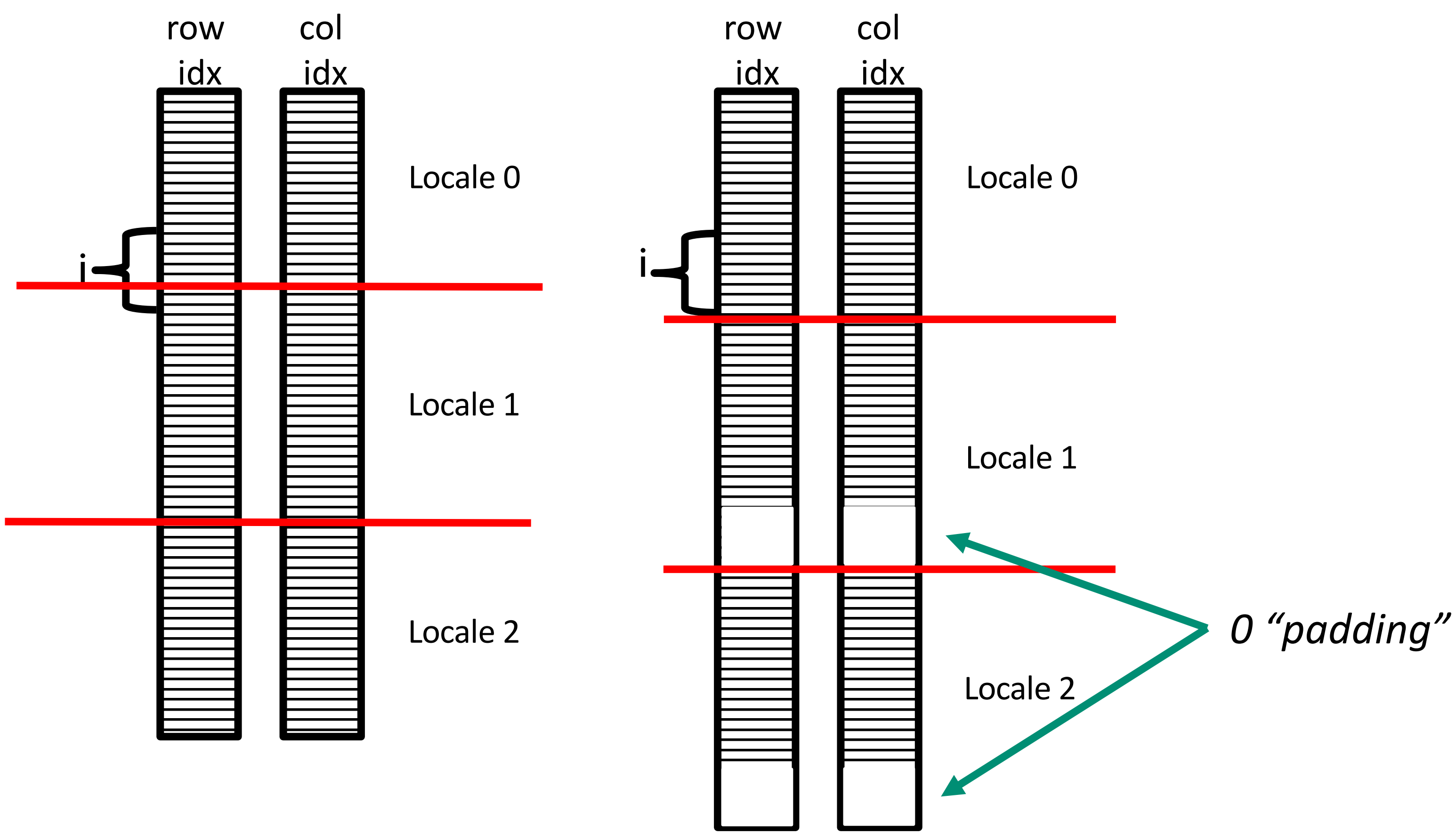
Chapel sparse domain

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

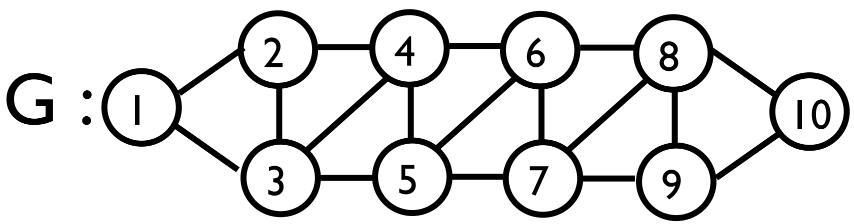
  (not allocating anything
  using the sparse domain)

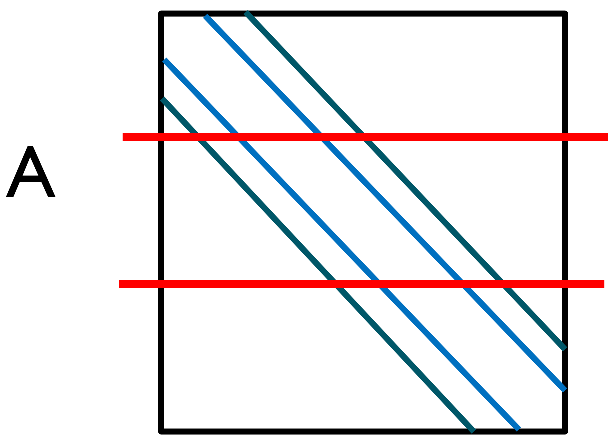


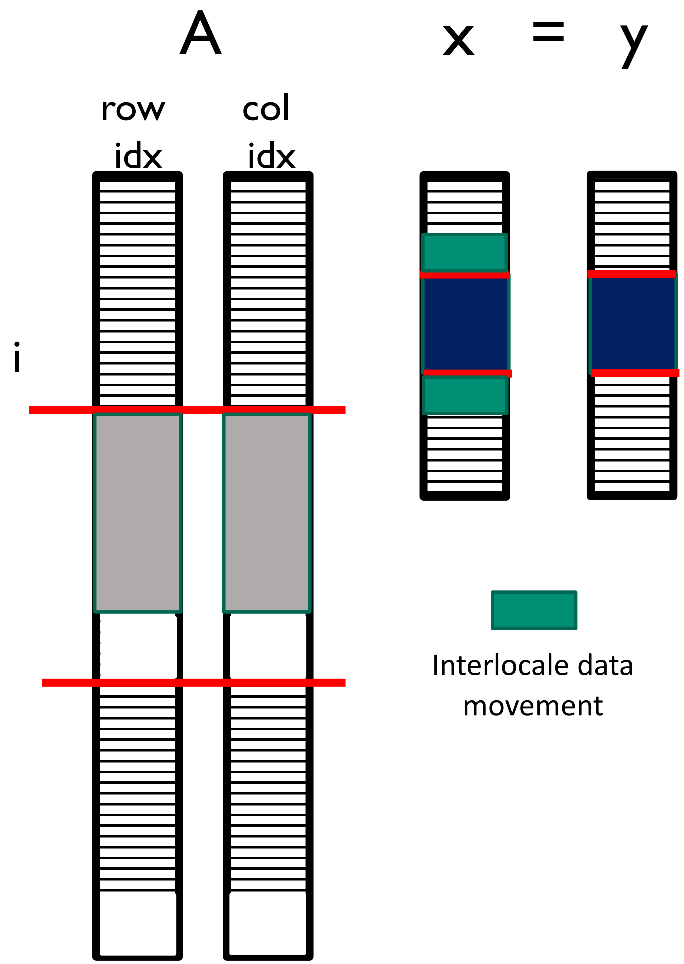
Balance the load (COO)



Example: banded matrix A, in COO format

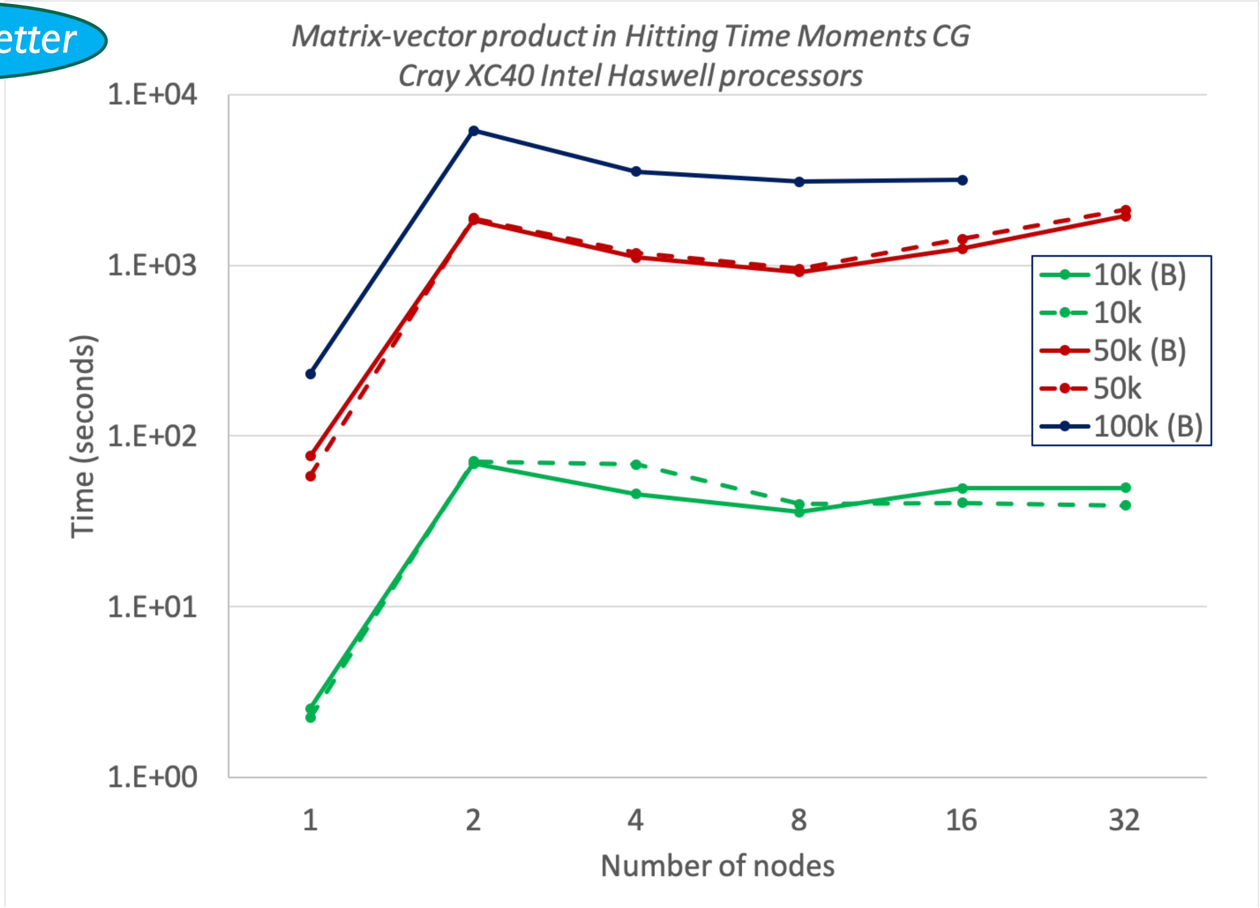




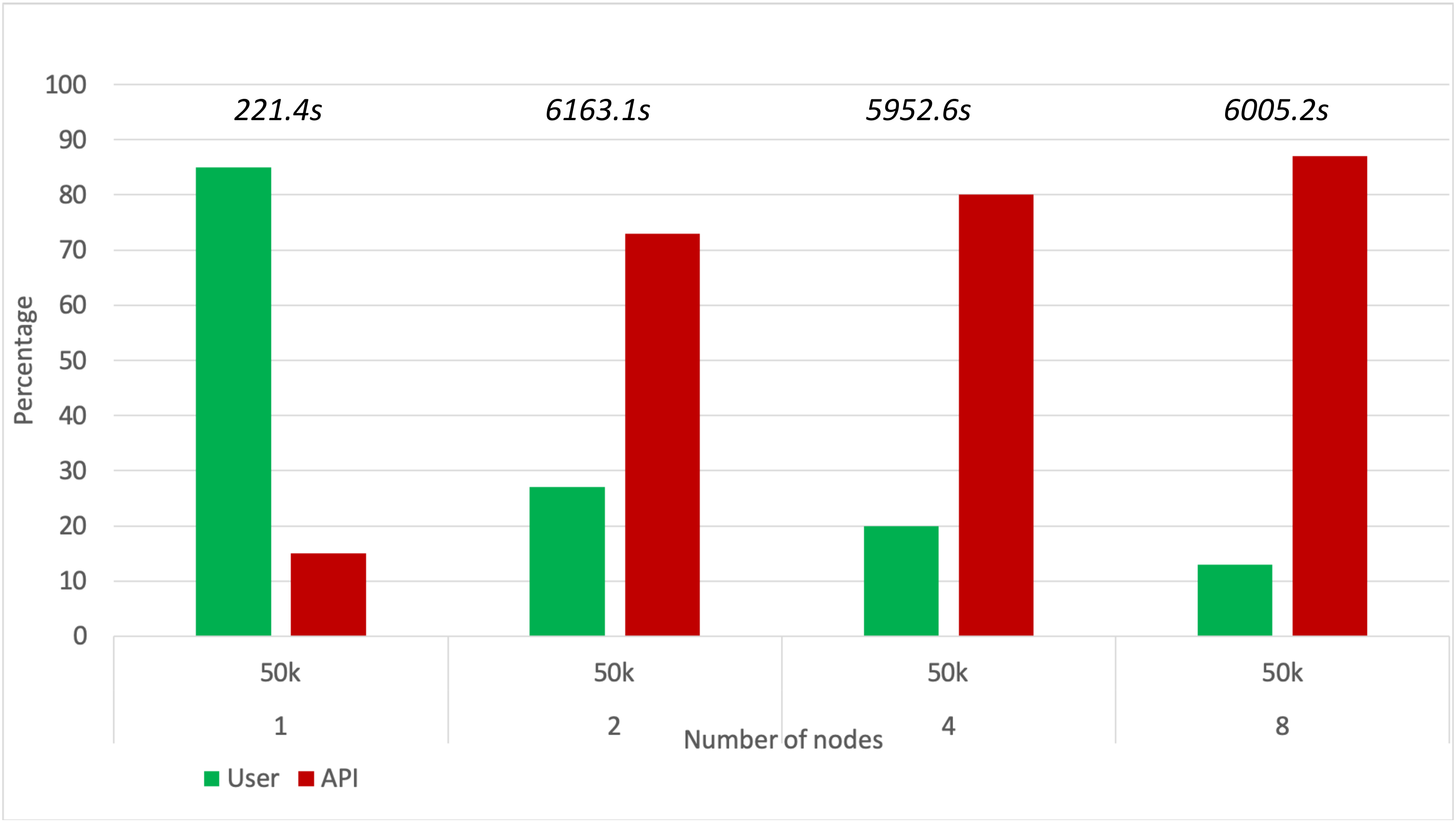


Strong Scaling

Lower is better



User vs API runtime



Performance Tools

CrayPat

- Results look like it's mostly monitoring runtime, not user code.
- No longer supports Chapel.

HPCToolKit

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

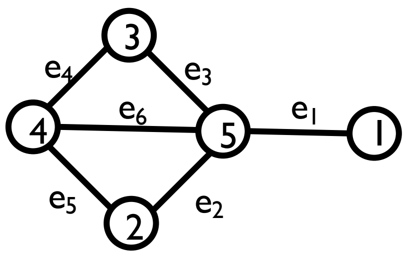
LDMS

- 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?

Triangle enumeration
Key computation: sparse MatMat



Adjacency matrix   Incidence matrix

$$\begin{array}{cc} & \text{vertex} \rightarrow \\ \text{vertex} \downarrow & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & \boxed{1} & 1 & \boxed{1} & 0 \end{pmatrix} \end{array} * \begin{array}{cc} & \text{edge} \rightarrow \\ \text{vertex} \downarrow & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \boxed{1} & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \boxed{1} & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array} = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{2,4,5\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{3,4,5\} \\ \emptyset & \{4,2,5\} & \{4,3,5\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{5,3,4\} & \boxed{\{5,2,4\}} & \emptyset \end{pmatrix}$$

Summary

Scaling performance currently poor.

We're assuming no known graph structure.

Exploring various matrix storage formats:

- COO, CSR, Chapel sparse domain

User supplied Chapel operator capability.

Need tools!

Future work

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

Additional slides

Productivity

Time from idea to solution   (DARPA HPCS motivator)

- Expressiveness
- Performance
- Portability
- Robustness
- Code development tools

Conjugate gradient method solving A*x=b

For symmetric positive definite matrix A in R^{nxn}, x and b in R^{nx1}

```
r = b - A*x;
error = norm( r ) / bnrm2;
if ( error < tol ) return, end

for iter = 1:max_it

   z  = M \ r;  Preconditioning.    Ax=b => M^-1 Ax = M^-1 b; Jacobi: M = diag(A)
   rho = (r'*z);  inner product

   if ( iter > 1 ),
      beta = rho / rho_1;
      p = z + beta*p;  vector update (daxpy)
   else
      p = z;
   end

   q = A*p;  Matrix-vector product
   alpha = rho / (p'*q );  inner product
   x = x + alpha * p;   vector update (daxpy)

   r = r - alpha*q;    vector update (daxpy)
   error = norm( r ) / bnrm2;
   if ( error <= tol ), break, end

   rho_1 = rho;

end
```

Matrix-vector multiplication: COO and CSR matrix storage

COO: Arrays for row indices, column indices (values: n/a for us)

```
for i in y.dom {     // For nnz nonzero coefficients
  y[rowidx[i]] += x[colidx[i]] * A[rowidx[i]];
}
```

CSR: rowptr[i+1] – rowptr[i] – 1 = number of nonzeros in row i.

(For a 6 banded matrix, rowptr = 1, 7, 13, 19, …)

```
for i in y.dom{    // For n matrix rows
  for j in rowptr[i]..rowptr[i+1]-1 {
     y[i] += x[colidx[j]] * A[i];
}}
Analogous for Compressed Column (CSC)
```