Exploring Chapel Productivity Using Some Graph Algorithms

Richard Barrett
Chapel Implementors and Users Workshop 2020
May 22, 2020
SAND 2020 – 5317 C
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

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)

Coordinate storage (COO)

Row compressed (CSR)

(i,j) = a(i,j)

All values = 1
Balance the load (COO)
Example: banded matrix $A$, in COO format

$G : 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10$

$A$

$A = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 \ 2 & 4 & 6 & 8 & 10 \ \end{bmatrix}$

$x = y$

Interlocale data movement
Strong Scaling

Matrix-vector product in Hitting Time Moments CG
Cray XC40 Intel Haswell processors

Lower is better

Time (seconds)

Number of nodes

10k (B)
10k
50k (B)
50k
100k (B)
User vs API runtime

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>User</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>221.4s</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6163.1s</td>
<td>6163.1s</td>
</tr>
<tr>
<td>4</td>
<td>5952.6s</td>
<td>6005.2s</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>6005.2s</td>
</tr>
</tbody>
</table>
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

\[
\begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
\end{array}
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{array}
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\{4,2,5\} & \{4,3,5\} & 0 & 0 \\
\{5,3,4\} & \{5,2,4\} & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\{2,4,5\} & \{3,4,5\} & 0 & 0 \\
\{5,3,4\} & \{5,2,4\} & 0 & 0 \\
\end{array}
\]
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 $A\mathbf{x}=b$

For symmetric positive definite matrix $A$ in $\mathbb{R}^{n \times n}$, $\mathbf{x}$ and $b$ in $\mathbb{R}^{n \times 1}$

```
for iter = 1:max_it
    z = M \ r;  \ \ \ \ \text{Preconditioning. \ \ } A\mathbf{x}=b \Rightarrow M^{-1}A\mathbf{x} = M^{-1}b; \text{Jacobi: } M = \text{diag}(A)
    rho = (r'*z);
    if ( iter > 1 ),
        beta = rho / rho_1;
        p = z + beta*p; \text{ vector update (daxpy)}
    else
        p = z;
    end
    q = A*p; \text{ Matrix-vector product }
    alpha = rho / (p'*q ); \text{ inner product }
    x = x + alpha * p; \text{ vector update (daxpy)}
    r = r - alpha*q; \text{ vector update (daxpy)}
    error = norm( r ) / bnorm2;
    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)

```python
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, ...)

```python
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)