Parallelization of the Shortest Path Graph Kernel on the GPU

Lifan Xu **Wei Wang**

Marco A. Alvarez **John Cavazos**

Department of Computer and Information Science

University of Delaware

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

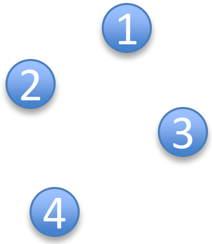
Graph

- A *graph* G is a set of vertices ***V*** and edges ***E*,** where ***E*** ⊂ ***V***^{2}
- A *graph* G may have labels on vertices and/or edges
- The *adjacency matrix* ***A*** of G is defined as

$$[\boldsymbol{A}_{ij}] = \begin{cases} 1 & if\ (v_i, v_j) \in \boldsymbol{E} \\ 0 & otherwise \end{cases}$$

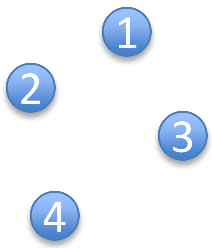
Labelled Undirected Graphs

vertices

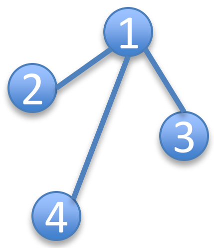


Labelled Undirected Graphs

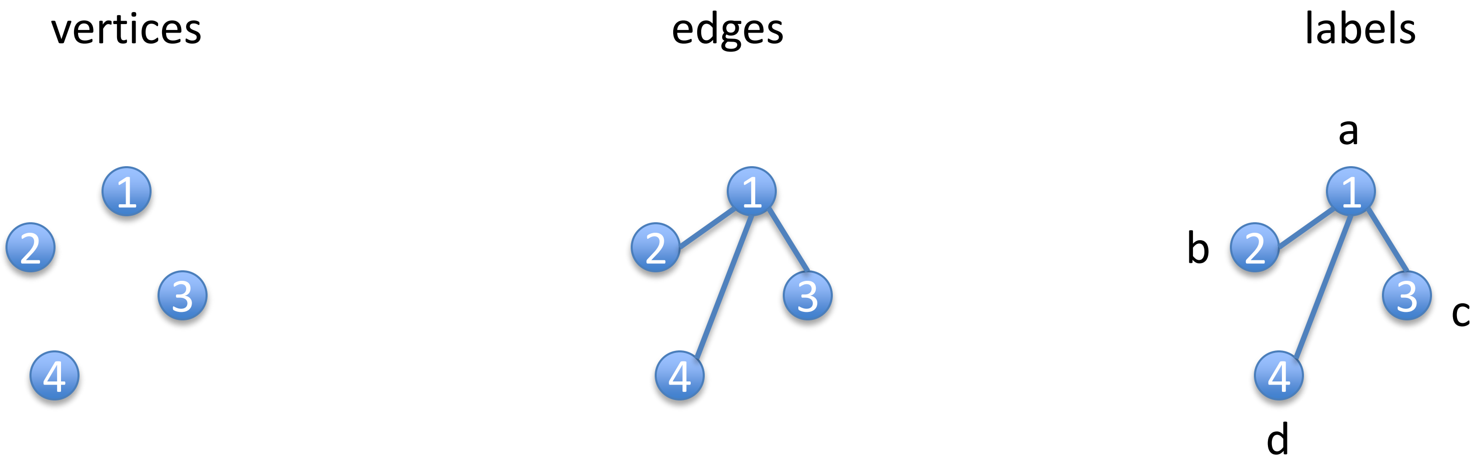
vertices



edges

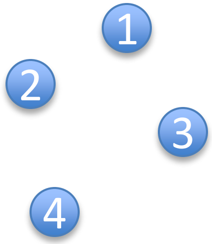


Labelled Undirected Graphs

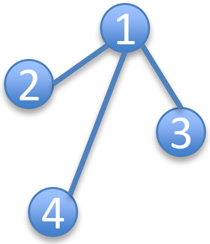


Labelled Undirected Graphs

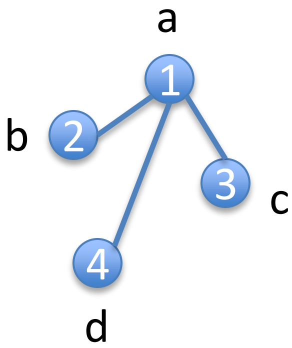
vertices



edges



labels



$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

Graph Similarity

- How similar are two graphs?
  - For Machine Learning problems like clustering and classification on graphs, graph similarity is crucial.
- Applications
  - Protein function prediction
  - Drug screening
  - Documents classification (Junk mail?)
  - Image classification
  - Cyber security
- Challenges
  - Graph isomorphism is NP-complete
  - Graph comparison via isomorphism is prohibitively expensive

Graph Kernel

- To Calculate the similarities between two graphs in polynomial time
  - Random Walk Kernel
    - Compare all walks in two graphs ***G*** and ***G'***
  - Shortest Path Kernel
    - Compare all pairs shortest paths for ***G*** and ***G'*** via Floyd-Warshall
  - Subtree Kernel
    - Compare subtree-like patterns in two graphs ***G*** and ***G'***
  - Cyclic Pattern Kernel
    - Compare simple cycles in two graphs ***G*** and ***G'***
  - Graphlet Kernel
    - Count subgraphs of limited size *K* in ***G*** and ***G'***

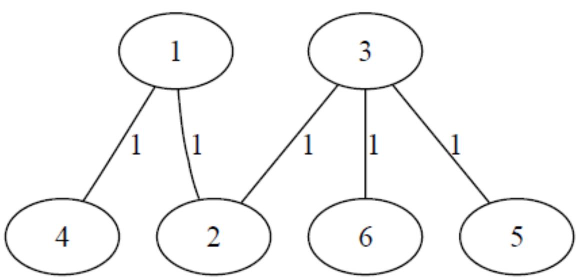
Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

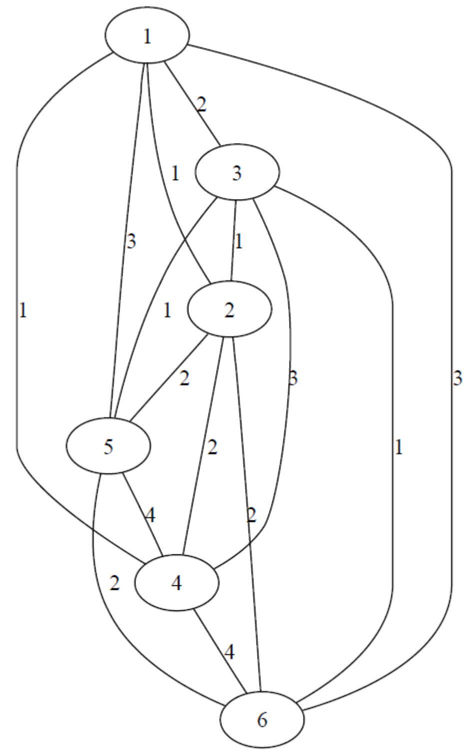
Shortest-Path Graph Kernel

- Convert graph to all pair shortest path graph
  - Floyd-Warshall Algorithm

Floyd-Warshall

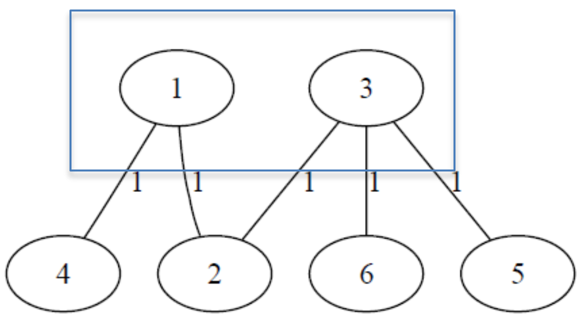


Original Graph

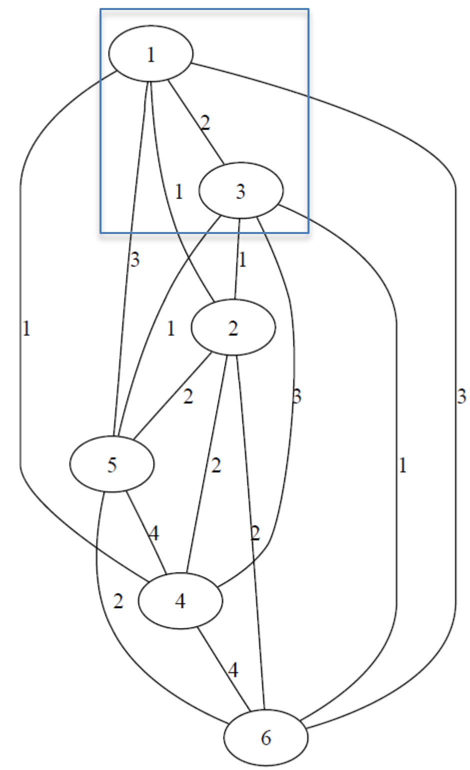


Shortest Path Graph

Floyd-Warshall



Original Graph



Shortest Path Graph

Shortest-Path Graph Kernel

- Apply shortest path kernel
  - $K_{sp}(G, G') = \sum_{e \in E} \sum_{e' \in E'} K_{walk}(e, e')$

Shortest-Path Graph Kernel

- Apply shortest path kernel
  - $K_{sp(G,G')} = \sum_{e \in E} \sum_{e' \in E'} K_{walk}(e, e')$
  - $K_{walk}(e, e') = K_{node}(u, u') \cdot K_{edge}(e, e') \cdot K_{node}(v, v')$

Shortest-Path Graph Kernel

- Apply shortest path kernel
  - $K_{sp(G,G')} = \sum_{e \in E} \sum_{e' \in E'} K_{walk}(e, e')$
  - $K_{walk}(e, e') = K_{node}(u, u') \cdot K_{edge}(e, e') \cdot K_{node}(v, v')$
  - K_{node} is a valid kernel function for comparing two vertices
  - K_{edge} is a valid kernel function for comparing two edges

Shortest Path Graph Kernel

- Lines 4-5 loop through all paths in G1

```
1: K ← 0
2: n1 ← num_node[g1]
3: n2 ← num_node[g2]
4: for i = 0 → n1, j = 0 → n1 do
5:     if i ≠ j AND sp_mat[g1][i][j] ≠ INF then
6:         for m = 0 → n2, n = 0 → n2 do
7:             if m ≠ n AND sp_mat[g2][m][n] ≠ INF then
8:                 k_edge ← EdgeKernel(sp_mat[g1][i][j], sp_mat[g2][m][n])
9:                 if K_edge > 0 then
10:                    k_node1 ← NodeKernel(g1, g2, i, m)
11:                    k_node2 ← NodeKernel(g1, g2, j, n)
12:                    K+ = k_node1 * k_edge * k_node2
13:                end if
14:            end if
15:        end for
16:    end if
17: end for
18: return K
19:
```

Shortest Path Graph Kernel

- Lines 4-5 loop through all paths in G1
- Lines 6-7 loop through all paths in G2

```
1:  K ← 0
2:  n1 ← num_node[g1]
3:  n2 ← num_node[g2]
4:  for i = 0 → n1, j = 0 → n1 do
5:      if i ≠ j AND sp_mat[g1][i][j] ≠ INF then
6:          for m = 0 → n2, n = 0 → n2 do
7:              if m ≠ n AND sp_mat[g2][m][n] ≠ INF then
8:                  k_edge ← EdgeKernel(sp_mat[g1][i][j], sp_mat[g2][m][n])
9:                  if K_edge > 0 then
10:                     k_node1 ← NodeKernel(g1, g2, i, m)
11:                     k_node2 ← NodeKernel(g1, g2, j, n)
12:                     K+ = k_node1 ∗ k_edge ∗ k_node2
13:                 end if
14:             end if
15:         end for
16:     end if
17: end for
18: return K
19:
```

Shortest Path Graph Kernel

- Lines 4-5 loop through all paths in G1
- Lines 6-7 loop through all paths in G2
- Line 8 calculates $K_{edge}(e, e')$

```
1: K ← 0
2: n1 ← num_node[g1]
3: n2 ← num_node[g2]
4: for i = 0 → n1, j = 0 → n1 do
5:     if i ≠ j AND sp_mat[g1][i][j] ≠ INF then
6:         for m = 0 → n2, n = 0 → n2 do
7:             if m ≠ n AND sp_mat[g2][m][n] ≠ INF then
8:                 k_edge ← EdgeKernel(sp_mat[g1][i][j], sp_mat[g2][m][n])
9:                 if K_edge > 0 then
10:                    k_node1 ← NodeKernel(g1, g2, i, m)
11:                    k_node2 ← NodeKernel(g1, g2, j, n)
12:                    K+ = k_node1 * k_edge * k_node2
13:                end if
14:            end if
15:        end for
16:    end if
17: end for
18: return K
19:
```

Shortest Path Graph Kernel

- Lines 4-5 loop through all paths in G1
- Lines 6-7 loop through all paths in G2
- Line 8 calculates $K_{edge}(e, e')$
- Lines 10-11 calculate $K_{node}(v, v')$

```
1: K ← 0
2: n1 ← num_node[g1]
3: n2 ← num_node[g2]
4: for i = 0 → n1, j = 0 → n1 do
5:     if i ≠ j AND sp_mat[g1][i][j] ≠ INF then
6:         for m = 0 → n2, n = 0 → n2 do
7:             if m ≠ n AND sp_mat[g2][m][n] ≠ INF then
8:                 k_edge ← EdgeKernel(sp_mat[g1][i][j], sp_mat[g2][m][n])
9:                 if K_edge > 0 then
10:                    k_node1 ← NodeKernel(g1, g2, i, m)
11:                    k_node2 ← NodeKernel(g1, g2, j, n)
12:                    K+ = k_node1 ∗ k_edge ∗ k_node2
13:                end if
14:            end if
15:        end for
16:    end if
17: end for
18: return K
19:
```

Shortest Path Graph Kernel

- Lines 4-5 loop through all paths in G1
- Lines 6-7 loop through all paths in G2
- Line 8 calculates $K_{edge}(e, e')$
- Lines 10-11 calculate $K_{node}(v, v')$
- Line 12 calculates $K_{walk}(e, e')$

```
1: K ← 0
2: n1 ← num_node[g1]
3: n2 ← num_node[g2]
4: for i = 0 → n1, j = 0 → n1 do
5:     if i ≠ j AND sp_mat[g1][i][j] ≠ INF then
6:         for m = 0 → n2, n = 0 → n2 do
7:             if m ≠ n AND sp_mat[g2][m][n] ≠ INF then
8:                 k_edge ← EdgeKernel(sp_mat[g1][i][j], sp_mat[g2][m][n])
9:                 if K_edge > 0 then
10:                    k_node1 ← NodeKernel(g1, g2, i, m)
11:                    k_node2 ← NodeKernel(g1, g2, j, n)
12:                    K+ = k_node1 * k_edge * k_node2
13:                end if
14:            end if
15:        end for
16:    end if
17: end for
18: return K
19:
```

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

Problem to be solved

- Given a set of graphs $\{g_1, g_2, \ldots, g_n\}$
- Calculate the kernel matrix $\boldsymbol{K_{nxn}}$
- $\boldsymbol{K_{(i,j)}}$ is the similarity between g_i and g_j

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

GPU Naive1

- Calculate the whole kernel matrix $\boldsymbol{K_{nxn}}$ in GPU at once
  - One GPU thread calculate one element in kernel matrix $\boldsymbol{K_{nxn}}$
  - # of GPU threads is n^2

Drawbacks of Naive1

- May not have enough GPU memory for large data set
  - 3GB global memory on Nvidia Tesla C2050
- GPU threads may have different workload due to different graph sizes
  - Load balancing
  - Branch divergence
- Works good if all graphs are small and have the same size

GPU Naive2

- Calculate similarity between one pair in GPU at a time
  - One GPU thread takes one entry in Shortest Path Adjacency Matrix of one input graph
  - # of GPU thread equals to the size of Shortest Path Adjacency Matrix of one input graph
  - If there is one edge, then loop through all entries in the other Shortest Path Adjacency Matrix

Drawbacks of Naive2

- Waste of GPU resources
  - May have idle threads because ***0*** and ***INF*** in Shortest Path Adjacency Matrix
- Slow Memory access
  - Random, non-coalesced memory access pattern

0	1	2
0	0	1
0	0	0

Shortest Path Adjacency Matrix

Data Transformation

- Transform Shortest Path Adjacency Matrix to three arrays with length equals to number of shortest paths
  - ***SP_W*** to store the weight of each path
  - ***SP_S*** to store the starting node of each path
  - ***SP_E*** to store the ending node of each path

0	1	2
0	0	1
0	0	0

Shortest Path Adjacency Matrix

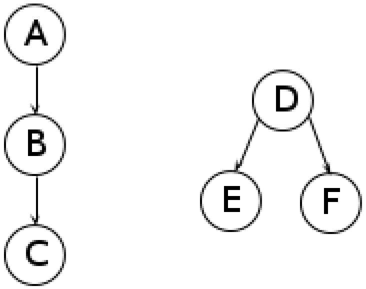
0	1	2
0	0	1
0	0	0

Shortest Path Adjacency Matrix

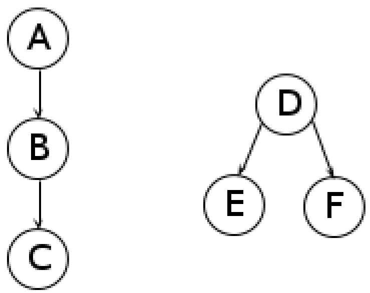
SP_W	1	2	1
SP_S	0	0	1
SP_E	1	2	2

Advanced GPU Implementation

- Pre-calculation of K_{node} using ***Vertex Kernel***
- Calculate K_{walk} uisng ***Edge Kernel***
- Apply ***Reduction Kernel*** to sum the results



Input graphs

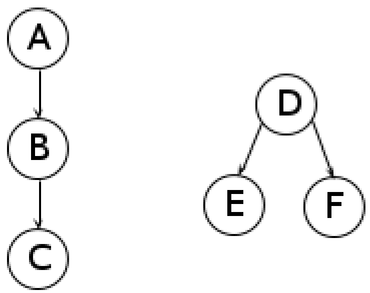


Input graphs

	A	B	C
A	0	1	0
B	0	0	1
C	0	0	0

	D	E	F
D	0	1	1
E	0	0	0
F	0	0	0

Adjacency matrix



Input graphs

	A	B	C
A	0	1	0
B	0	0	1
C	0	0	0

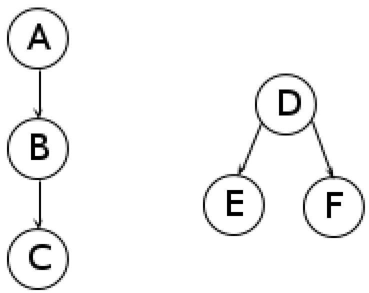
	D	E	F
D	0	1	1
E	0	0	0
F	0	0	0

Adjacency matrix

	A	B	C
A	0	1	2
B	0	0	1
C	0	0	0

	D	E	F
D	0	1	1
E	0	0	0
F	0	0	0

Shortest Path Adjacency matrix



Input graphs

	A	B	C
A	0	1	0
B	0	0	1
C	0	0	0

	D	E	F
D	0	1	1
E	0	0	0
F	0	0	0

Adjacency matrix

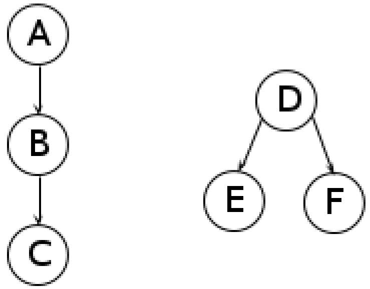
	A	B	C
A	0	1	2
B	0	0	1
C	0	0	0

	D	E	F
D	0	1	1
E	0	0	0
F	0	0	0

Shortest Path Adjacency matrix

↯$_0$ A D	↯$_1$ A E	↯$_2$ A F
↯$_3$ B D	↯$_4$ B E	↯$_5$ B F
↯$_6$ C D	↯$_7$ C E	↯$_8$ C F

Vertex Kernel



Input graphs

	A	B	C
A	0	1	0
B	0	0	1
C	0	0	0

	D	E	F
D	0	1	1
E	0	0	0
F	0	0	0

Adjacency matrix

	A	B	C
A	0	1	2
B	0	0	1
C	0	0	0

	D	E	F
D	0	1	1
E	0	0	0
F	0	0	0

Shortest Path Adjacency matrix

↯0 A D	↯1 A E	↯2 A F
↯3 B D	↯4 B E	↯5 B F
↯6 C D	↯7 C E	↯8 C F

Vertex Kernel

↯0	A→B D→E	A→B D→F
↯1	A→C D→E	A→C D→F
↯2	B→C D→E	B→C D→F

Edge Kernel

Advantage and Disadvantage of Adv. Implementation

- Advantage
  - No branch divergence
  - Coalesced memory access
- Disadvantage
  - Waste of GPU resource when graphs are small

Hybrid Implementation

- Combine ***Naive1*** and ***Advanced***
- Sort the input graphs according to their sizes
- Set a threshold for the graph size
  - For graphs with sizes smaller than threshold, use ***Naive1***
  - Otherwise, use ***Advanced***

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

OpenMP Implementation

- Convert the top triangle of the kernel matrix to a 1D array
- Create as many OpenMP threads as number of CPU cores
- Each OpenMP thread calculates one entry in the 1D array in order, goes to next iteration until all entries are computed

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

Execution Environment

- **CPU** - Intel 5530 Quad core @ 2.4 GHz with 8MB cache (8 OpenMP threads)
- **GPU** - NVIDIA C2050 (448 Cores @ 1.15GHz) with 3GB GDDR5 1.5 GHZ ECC RAM

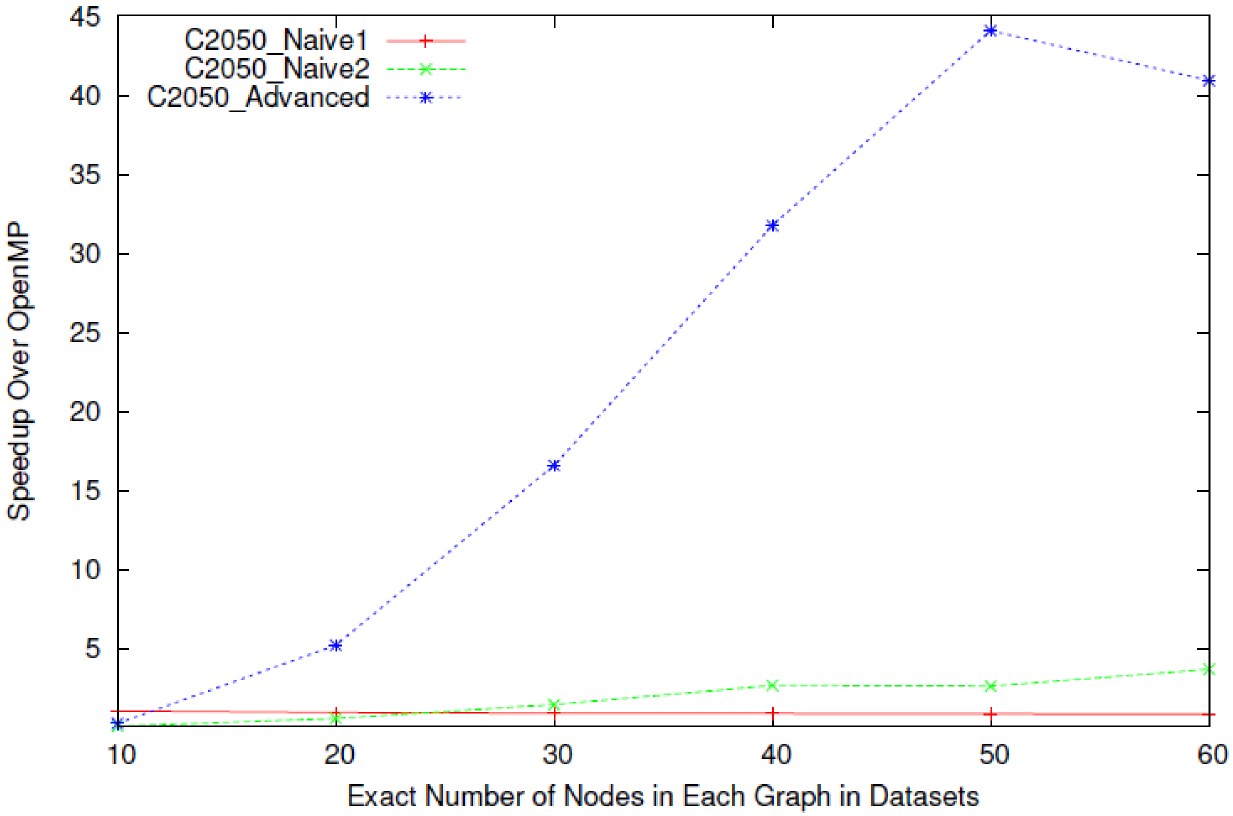
Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

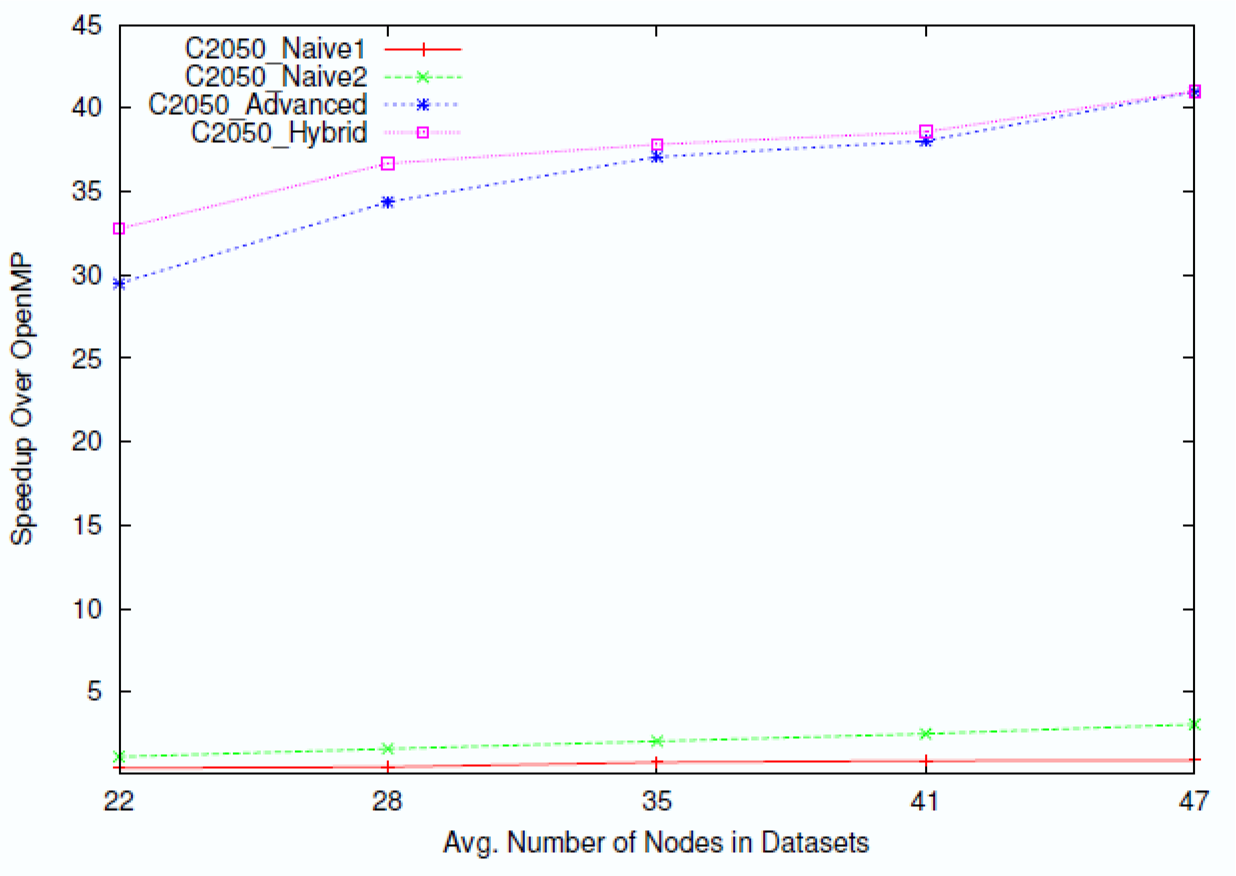
Synthetic Datasets

Dataset	Avg. Nodes	Avg. Edges	Avg. SP
10-nodes	10	19	61
20-nodes	20	76	367
30-nodes	30	175	867
40-nodes	40	310	1559
50-nodes	50	489	2449
60-nodes	60	706	3540
M1	22	191	930
M2	28	277	1365
M3	35	362	1800
M4	41	448	2235
M5	47	535	2670

Speedups on Uni-size Sets



Speedups on Mixed Sets



Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

Scientific Datasets

Dataset	Num. of Graphs	Min. Nodes	Max. Nodes	Avg. Nodes	Min. Edges	Max. Edges	Avg. Edges	Avg. SP
MUTAG	188	10	28	18	20	66	39	324
ENZYMES	600	2	126	33	2	298	124	1215
NCI1	4110	3	111	30	4	238	64	1005
NCI109	4127	4	111	30	6	238	64	995

Speedups on Scientific Datasets

Dataset	*Naive1*	*Advanced*	*Hybrid*
MUTAG	2.367	1.962	2.882
ENZYMES	1.320	10.823	10.895
NCI1	1.895	7.527	7.823
NCI109	1.992	7.751	8.037

Outline

- Introduction
  - Graph
  - Graph similarity and graph kernel
  - Shortest Path Graph Kernel
- Parallelization on GPU and CPU
  - Four GPU implementations
  - One OpenMP implementation
- Experiments results
  - Synthetic datasets
  - Scientific datasets
- Conclusion and Future Work

Conclusion and Future Work

- We present four different GPU parallelizations
- Achieve up to **44x speedup** on synthetic datasets
- Achieve up to **10x speedup** on scientific datasets
- We are going to accelerate other graph kernels in the future

Thanks!

Questions?