All Pairs Shortest Paths on Multiple GPUs

Terence Henriod

Today we'll be talking about...

- All Pairs Shortest Paths (APSP)
    - Computing the lengths/weights of the paths (and maybe storing those paths)
- Various algorithms and implementations
- Implementation on the GPU
- Implementation on multiple GPUs

Background

“The origin of the present methods provides an interesting illustration of the value of basic research on puzzles and games. Although such research is often frowned upon as being frivolous, it seems plausible that these algorithms might eventually lead to savings of very large sums of money by permitting more efficient use of congested transportation or communication systems. The actual problems in communication and transportation are so much complicated by timetables, safety requirements, signal-to-noise ratios, and economic requirements that in the past those seeking to solve them have not seen the basic simplicity of the problem, and have continued to use trial-and-error procedures which do not always give the true shortest path…

… However, in the case of a simple geometric maze, the absence of these confusing factors permitted algorithms A, B, and C to be obtained, and from them a large number of extensions, elaborations, and modifications are obvious. The problem was first solved in connection with Claude Shannon’s maze-solving machine. When this machine was used with a maze which had more than one solution, a visitor asked why it had not been built to always find the shortest path. Shannon and I each attempted to find economical methods of doing this by machine. He found several methods suitable for analog computation, and I obtained these algorithms. Months later the applicability of these ideas to practical problems in communication and transportation systems was suggested.”

- Edward Moore

What is Shortest Paths Computation

- Graphs are defined as G = (V, E), where V is the set of vertices and E is the set of edges between the vertices V. An edge between any two vertices is indicative of a binary relationship between the two, and can have a weight or cost associated.

- Sometimes we consider *directed* graphs, where the binary relationship is not mutual

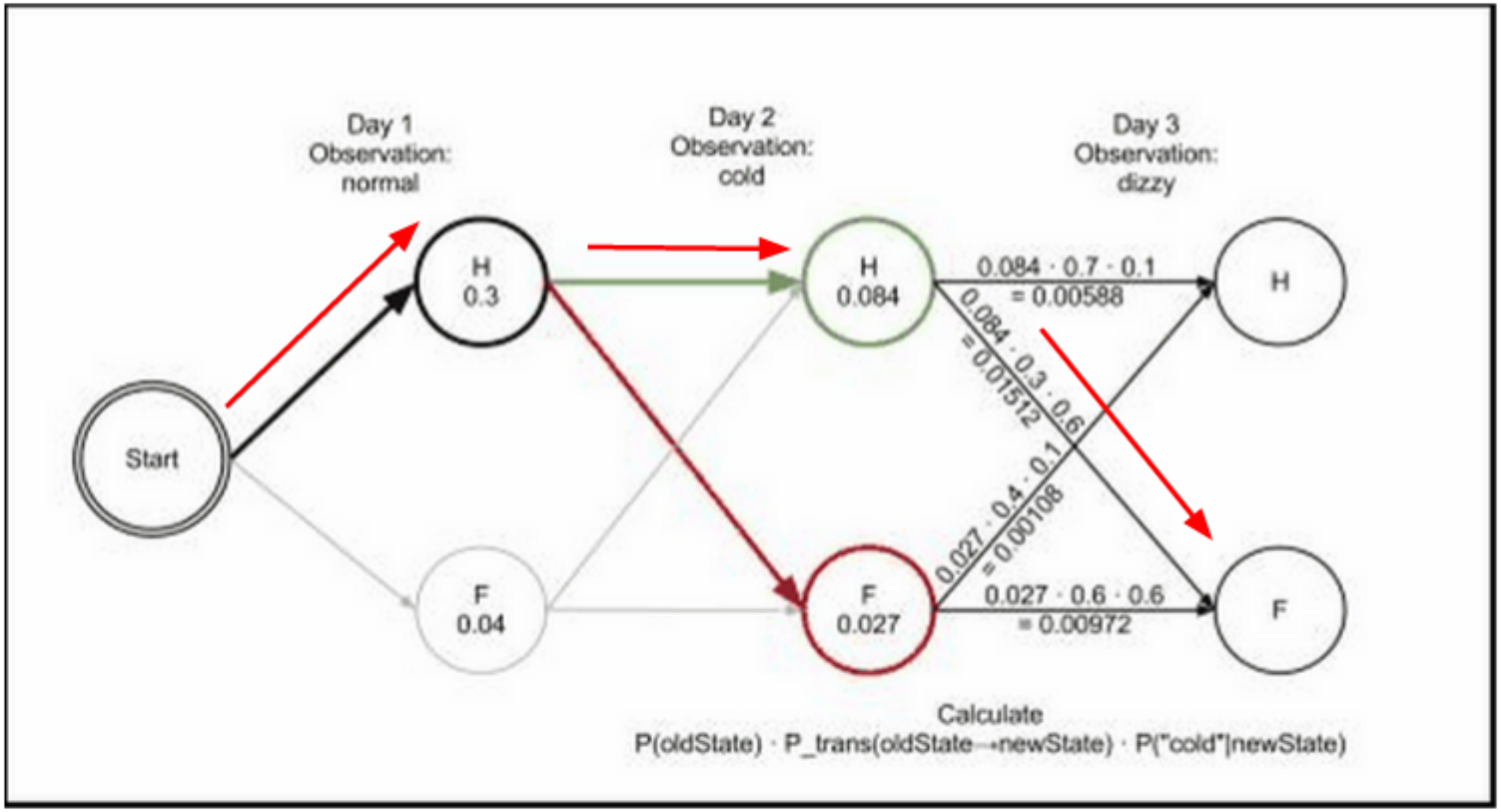
- We perform the computation to find the cost of the shortest path between a pair of vertices in a graph

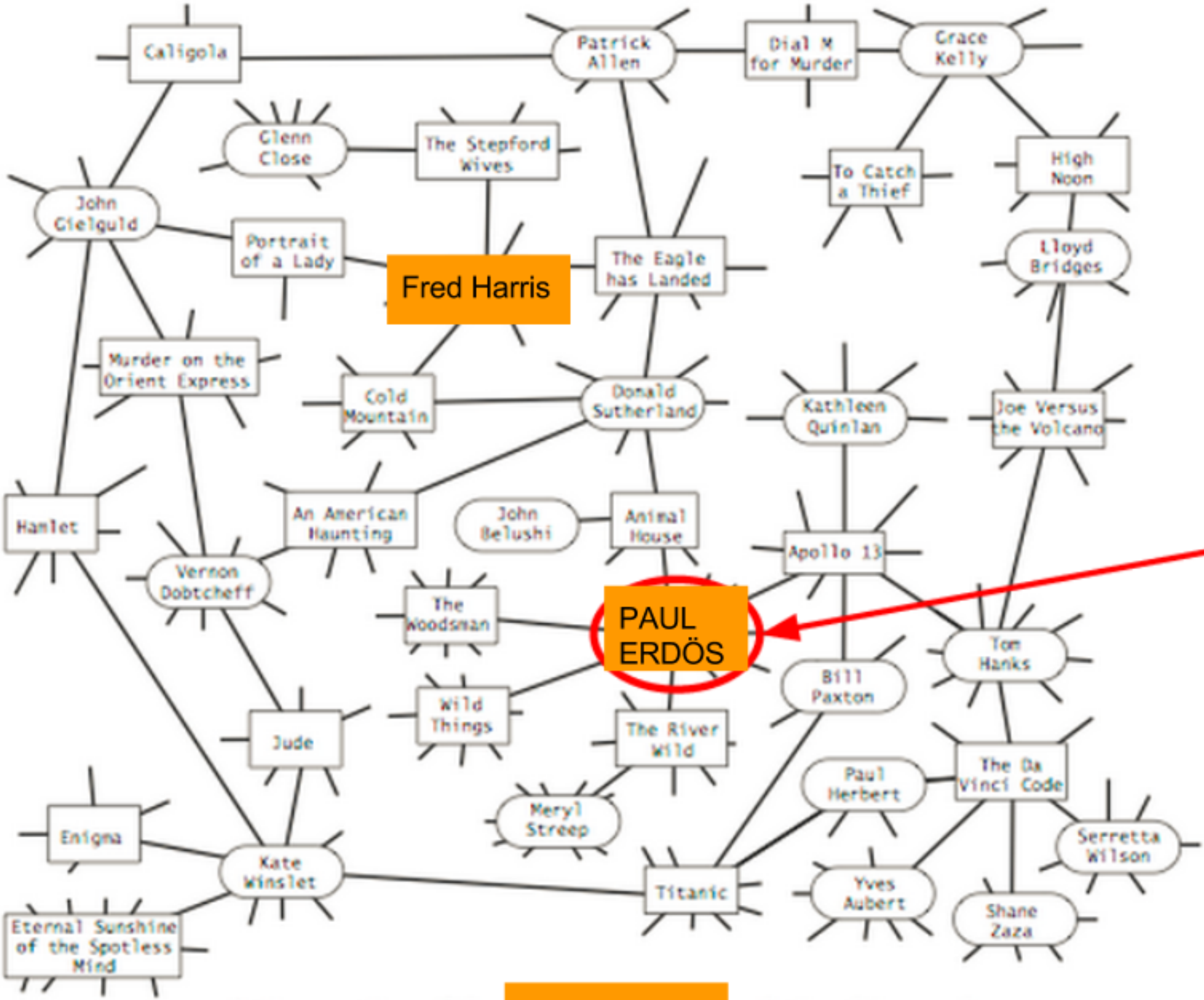
- Has a history riddled with “multiple invention”

Interesting Problems

- Examples
    - **Road Travel**
    - Network Packet Routing
    - VLSI
        - Very-Large-Scale Integration (of circuits)
    - Image Processing
    - **Neural Networks**
    - **Compiling/Code Optimization**
    - **Speech Recognition (Vitterbi)**

Interesting Solutions





A tiny portion of the researcher *relationship graph*

A Timeline of Shortest Paths

- 263: Gaussian Elimination (Liu Hui)
- 1946-55: Matrix methods for representing unit weight edges
  - Motivation: Finding transitive closure
- 1953-55: Shimbel Distance Matrices
  - Motivation: Informations transmission through neural nets
- 1956-7: Ford, Bellman-Ford
  - No negative CYCLES
- 1957/59: Case Institute of Technology / Dijkstra
  - Faster than Bellman-Ford, no negative EDGES.

- 1958: Dantzig
    - No negative EDGES, similar but slower than Dijkstra's
- 1959: Moore
    - A refinement of Bellman-Ford, slower than Dijkstra's
    - Suitable for **parallel implementation**
- 1967: Vitterbi (HMMs)
    - Finding probabilistic paths through networks
- 1972: Dijkstra's with Heaps
- 1980: A Unified Approach to Path Problems
    - Path problems can be cast as Gaussian Elimination
- 1877: Dijkstra's with Fibonnacci Heaps

- 1980s: A lot of people start casting APSP as a matrix solving problem
- 1990s: Some people start looking for integral weight path algorithms, others start working on dynamic APSP maintenance algorithms

The Textbook Algorithms

- Check Wikipedia

Algorithm	Use	Complexity
Djikstra’s	SSSP	O(V^2) ~ O(E + V log V)*
Bellman-Ford	SSSP with negative edges	O(VE)
A*	SSSP with heuristics	better than Dijkstra - O (b^d) worst case
Floyd-Warshall	APSP	O(V^3)
Johnson-Dijkstra	APSP for sparse graphs	O(EV + V^2 log V)
Viterbi	Shortest stochastic path with node weight (~SSSP)	O(L * |Q|^3)
BFS	SSSP for directed, unweighted graphs	O(|V| + |E|)

Various Methods for Shortest Paths

- Single Source Shortest Paths (SSSP)
  - Some can handle negative edges some can’t

- All Pairs Shortest Paths (APSP)
  - Some are SSSPs recycled for all vertices
  - Some are stand-alone algorithms
  - The situation may dictate which type to use

- Integer-weighted Graphs
  - Non-weighted edge problems
  - Integer weights less than some value *b*

- Real-weighted Graphs
  - Any numbers (depending on the algorithm, negatives weights may be allowed)
  - Generally we don’t consider graphs with negative cycles

Literature Review

Proofs of Correctness for the use of Matrix algebras

- The use of semi-ring matrix algebra is key to any matrix based algorithms
- Uses of Regular Expressions are very common for matrix based algorithms too
- The non-matrix algorithms tend to be much more verbose

Casting of Problems

- A Unified Approach to Path Problems, Robert Endre Tarjan 1981
  - Early work on casting seemingly unrelated problems as Gaussian Elimination (GE)
  - Uses Regular expressions extensively to show how shortest paths, systems of linear equations over real numbers, and even global [program] flow can be solved with the same strategy

Many Recent Works

- Improvements using Integral Edge Weights
- Dynamic APSP Maintenance
  - Preserve the APSP solution (once computed) across updates to the graph
  - For many of these, their worst case was about as bad as recomputing APSP from scratch

Integral Edge Weights

- These allow us to use special multiplications for integers
  - Example of such an improvement: we can use a sub-cubic matrix multiplication model like Strassen’s to solve our APSP problems
  - Too bad it’s only for integers

APSP From Scratch

- A new approach to all-pairs shortest paths on real-weighted graphs, Seth Pettie 2003
  - Presented a O(mn + n^2 log log n) APSP algorithm
  - An All-S-SSSP variant
  - No implementation though…

Dynamic APSP Maintenance

- A New Approach to Dynamic All Pairs Shortest Paths, Demetrescu and Italiano 2004
  - Maintains a solution to APSP through updates to a graph
  - This is the first dynamic one to “do it all”: real weights, negative weights, increases, decreases, etc.
  - This one improves over other dynamic ones significantly; older ones were still within reach of APSP from scratch
  - Claimed fast implementation, but none presented

Improvements in Matrix Algebra Routines

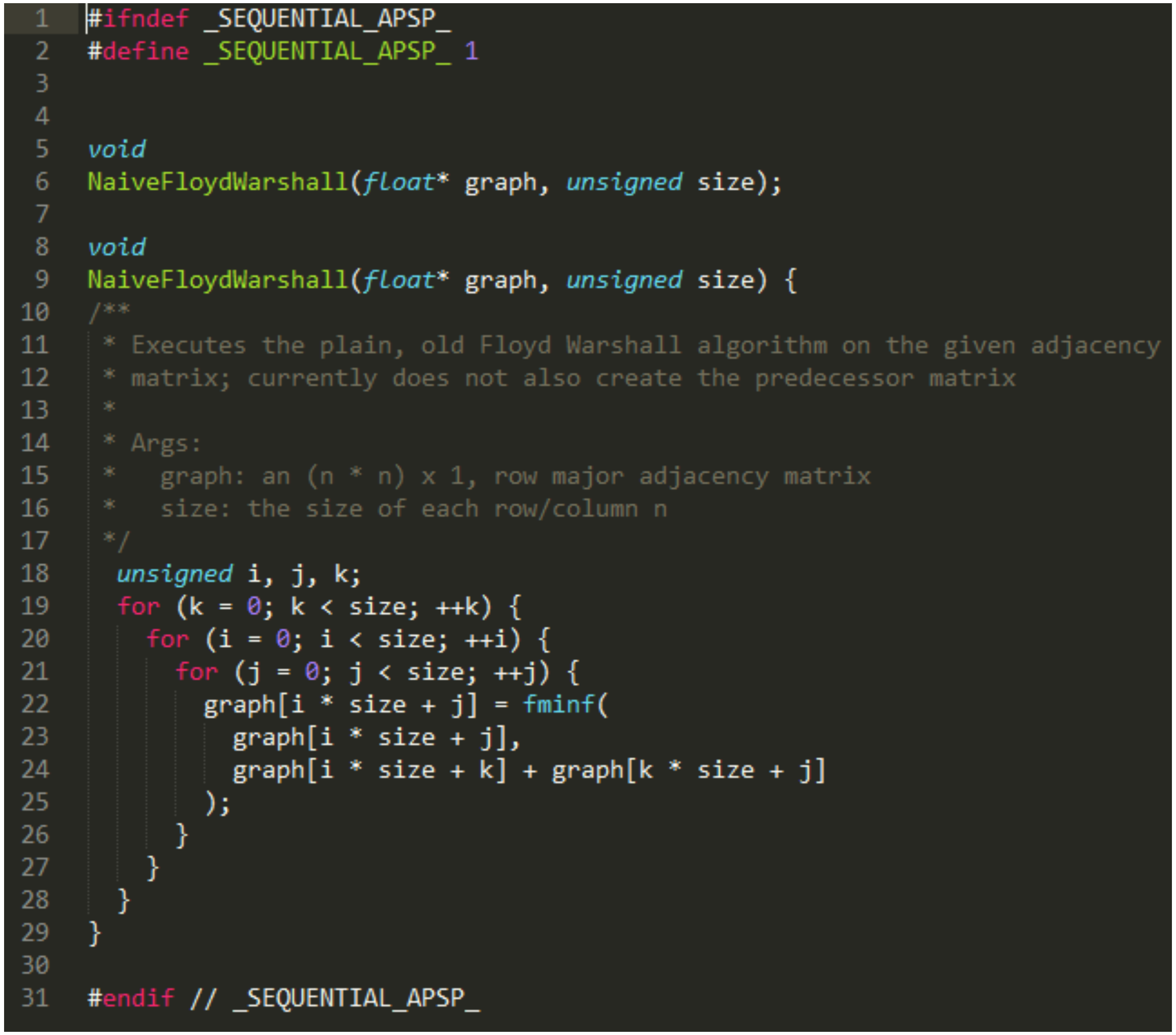
- Vasily Volkov, 2008
    - Implemented an SGEMM kernel for the GPU that outperformed the cuBLAS routine it would replace
    - Contributions like these enable improvements in many other areas

Most Recent APSP Work

- Solving Path Problems on the GPU
  - To our knowledge, the only recent (last 10 years) work on computing APSP from scratch has been the work of Buluc, Gilbert, and Budak.
  - A Recursive GPU Implementation that acheived ~480x speedup over CPU implementations

Sequential Results

The Code



```c
#ifndef _SEQUENTIAL_APSP_
#define _SEQUENTIAL_APSP_ 1


void
NaiveFloydWarshall(float* graph, unsigned size);

void
NaiveFloydWarshall(float* graph, unsigned size) {
/**
 * Executes the plain, old Floyd Warshall algorithm on the given adjacency
 * matrix; currently does not also create the predecessor matrix
 *
 * Args:
 *   graph: an (n * n) x 1, row major adjacency matrix
 *   size: the size of each row/column n
 */
  unsigned i, j, k;
  for (k = 0; k < size; ++k) {
    for (i = 0; i < size; ++i) {
      for (j = 0; j < size; ++j) {
        graph[i * size + j] = fminf(
          graph[i * size + j],
          graph[i * size + k] + graph[k * size + j]
        );
      }
    }
  }
}

#endif // _SEQUENTIAL_APSP_
```

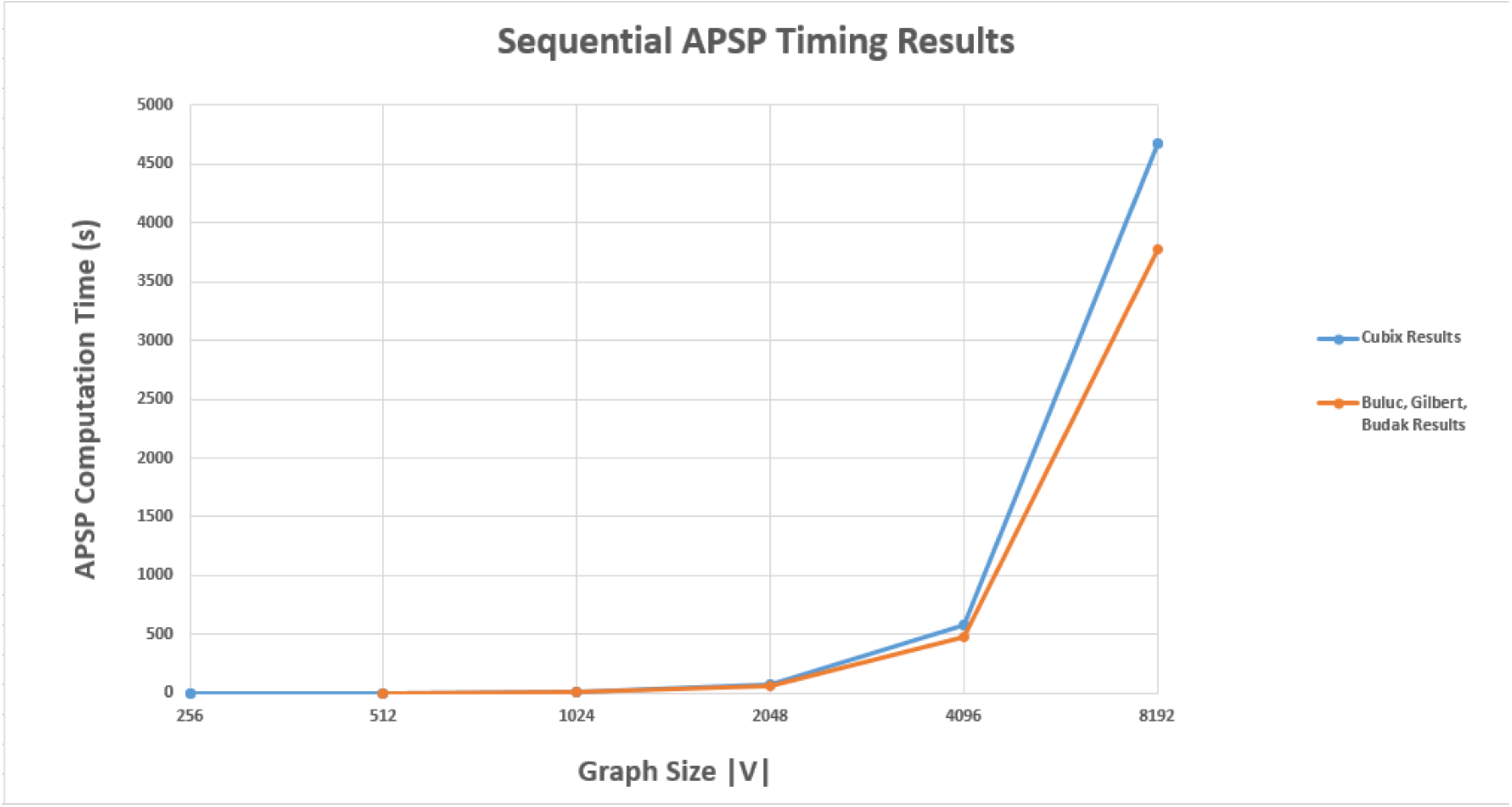
The Machine

- UNR Cubix
  - 24x:
    - Intel Xeon E5-2620 @ 2.00 GHz
    - 6 cores
    - Cache: 15360 KB
  - Although only of these really matters for the sequential implementation

The Numbers

Graph Size |V|	Cubix	Buluc, Gilbert, Budak
256	0.146	-
512	1.150	0.843
1024	9.031	7.400
2048	72.731	5.9400
4096	580.745	479.000
8192	4668.633	3770.00

The Graph



Sequential APSP Timing Results
APSP Computation Time (s)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
256
512
1024
2048
4096
8192
Graph Size |V|
Cubix Results
Buluc, Gilbert, Budak Results

GPU Implementation Approach

The General Algorithm

```
$A^* : \mathbb{R}^{N\times N} = \text{APSP}(A : \mathbb{R}^{N\times N})$
if $N < \beta$
    then $A \leftarrow \text{FW}(A)$                ▷ Base case: perform iterative FW serially
    else
        $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$
        $A_{11} \leftarrow \text{APSP}(A_{11})$
        $A_{12} \leftarrow A_{11}A_{12}$
        $A_{21} \leftarrow A_{21}A_{11}$
        $A_{22} \leftarrow A_{22} \oplus A_{21}A_{12}$
        $A_{22} \leftarrow \text{APSP}(A_{22})$
        $A_{21} \leftarrow A_{22}A_{21}$
        $A_{12} \leftarrow A_{12}A_{22}$
        $A_{11} \leftarrow A_{11} \oplus A_{12}A_{21}$
```

Fig. 3. Pseudocode for recursive in-place APSP.

Considerations for GPU Implementations

- Memory Locality
  - This makes or breaks a GPU implementation
- Memory Transfers
  - Kind of trivial with one GPU (still takes time, but there’s no way around it)
  - With multiple GPUs this may be a real problem depending on how many transfers need to be made
- Recursion is not strong on the GPU
  - (in my experience)
  - Have the CPU issue the kernel launches as in first paper

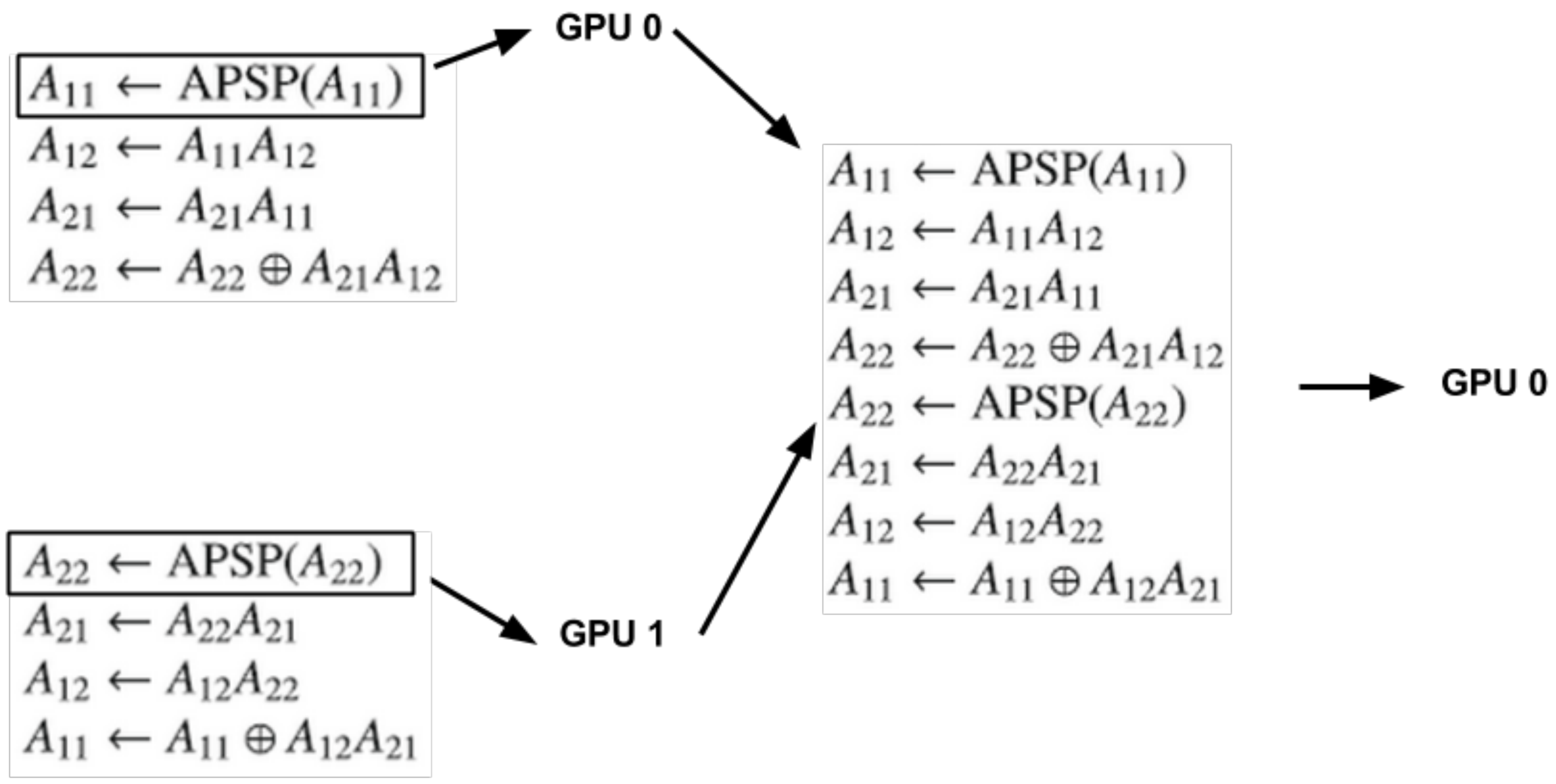
Possible Improvements

- Add a layer of parallelism by adding GPUs for *device parallelism*

- Increase the size of β (matrix size of “serial” FW solving)
    - Modern hardware can support more threads per block than the old hardware, possibly improving our ability to solve larger matrices outright

```
$A^* : \mathbb{R}^{N \times N} = \mathrm{APSP}(A : \mathbb{R}^{N \times N})$
if $N < \beta$
    then $A \leftarrow \mathrm{FW}(A)$          ▷ Base case: perform iterative FW serially
    else
        $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$
        $A_{11} \leftarrow \mathrm{APSP}(A_{11})$
        $A_{12} \leftarrow A_{11} A_{12}$
        $A_{21} \leftarrow A_{21} A_{11}$
        $A_{22} \leftarrow A_{22} \oplus A_{21} A_{12}$
        $A_{22} \leftarrow \mathrm{APSP}(A_{22})$
        $A_{21} \leftarrow A_{22} A_{21}$
        $A_{12} \leftarrow A_{12} A_{22}$
        $A_{11} \leftarrow A_{11} \oplus A_{12} A_{21}$
```

Fig. 3. Pseudocode for recursive in-place APSP.



```
$\boxed{A_{11} \leftarrow \mathrm{APSP}(A_{11})}$
$A_{12} \leftarrow A_{11}A_{12}$
$A_{21} \leftarrow A_{21}A_{11}$
$A_{22} \leftarrow A_{22} \oplus A_{21}A_{12}$
```

```
$\boxed{A_{22} \leftarrow \mathrm{APSP}(A_{22})}$
$A_{21} \leftarrow A_{22}A_{21}$
$A_{12} \leftarrow A_{12}A_{22}$
$A_{11} \leftarrow A_{11} \oplus A_{12}A_{21}$
```

```
$A_{11} \leftarrow \mathrm{APSP}(A_{11})$
$A_{12} \leftarrow A_{11}A_{12}$
$A_{21} \leftarrow A_{21}A_{11}$
$A_{22} \leftarrow A_{22} \oplus A_{21}A_{12}$
$A_{22} \leftarrow \mathrm{APSP}(A_{22})$
$A_{21} \leftarrow A_{22}A_{21}$
$A_{12} \leftarrow A_{12}A_{22}$
$A_{11} \leftarrow A_{11} \oplus A_{12}A_{21}$
```

Issues of Multiple GPUs

- Memory Coherency
  - Need to avoid multiple memory transfers between GPU ←→ CPU
    - I am concerned this limits us to 2 GPUs
  - Certainly results from GPUs must be rectified once computations are complete
- Load Balancing
  - I would think this won't be a big problem with 2 GPUs
  - Could easily increase with more than 2 depending on how it's implemented

Other Differences

- I will “jitter” the input sizes
  - I hope to see interesting effects of using non-power-of-two and odd inputs sizes that may have been ignored by the original paper
- Increased input sizes
  - Can’t compare with old timings because they couldn’t do sizes >8192 ($|V| = 8192^2 \rightarrow$ 268 MB)
  - Will the time requirements be prohibitive though?

    We could easily get to a range of computations that take days (for the sequential)...

- If there’s time: Add the maintenance of the predecessor graph to reconstruct the shortest paths
    - Should be simple to add the update line in the FW algorithm
    - Will the memory requirements be prohibitive?

      Probably not. Will they cause large slow downs? We’ll see.

Questions?

Thank you!