Newton Energy Group

gpuCA

a GPU-based Contingency Analysis Tool

Presented to FERC's 2019 Technical Conference

John Goldis – Newton Energy Group

Andrei Kharchenko – Newton Energy Group

Aleksandr Rudkevich – Newton Energy Group

Yonghong Chen – MISO Energy

Fengyu Wang – MISO Energy

This work was partially funded by MISO R&D grant

Agenda

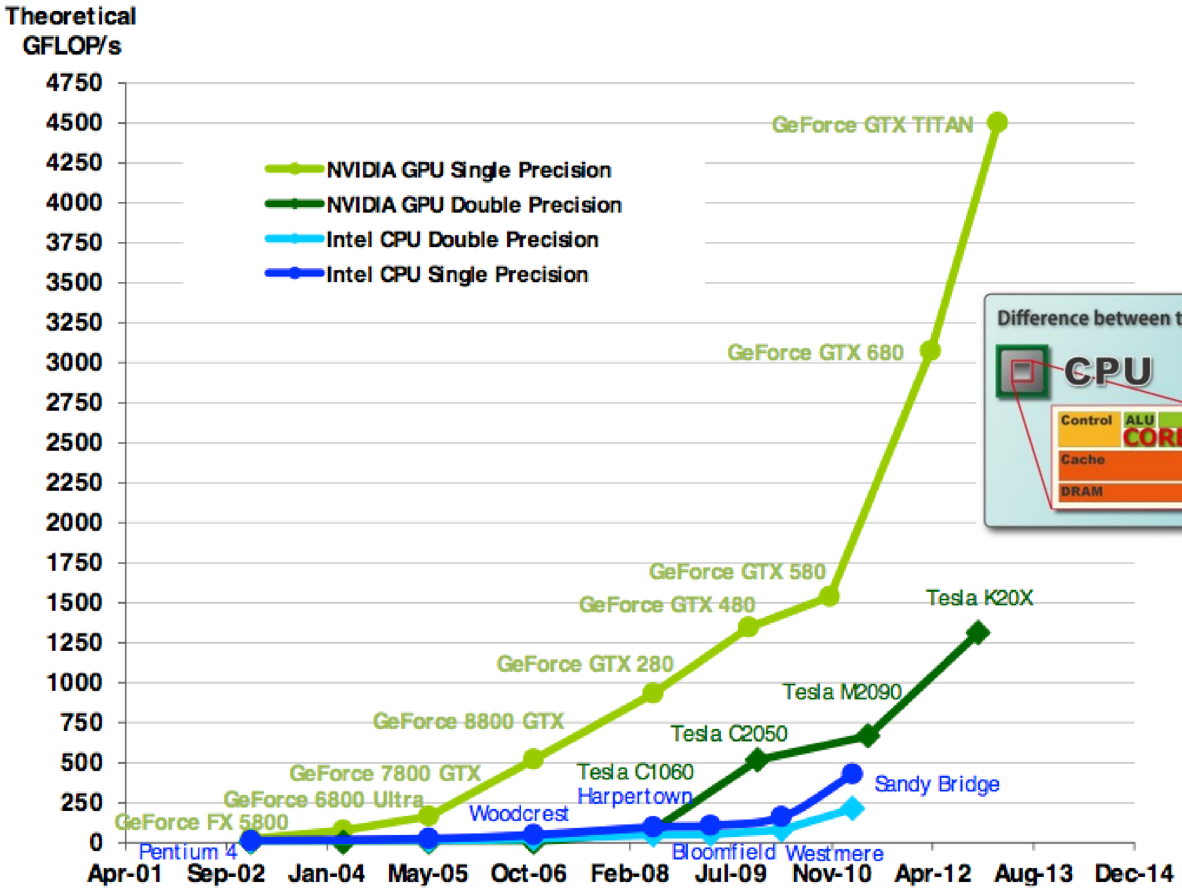
- Motivation
- GPU vs CPU
- GPU and Contingency Analysis
- MISO Case Study
- Algorithmic and Architectural Approach
- Initial Results
- Conclusions

Motivation

- Contingency Analysis is often a bottleneck in planning and market clearing algorithms
- Due to the large number of constraints to be screened there are two typical approaches
  - Limiting the set of contingency constraints based on offline assessment
  - Using HPC servers to parallelize the process on the CPU
- The first approach misses relevant contingency events that may occur as the state of the system changed in real time
- The second approach can be expensive and difficult to manage

GPU vs CPU Performance

While the GPU can process significantly more floating point operations per second, the CPU processes significantly more instructions per second, i.e. the clock speed (not shown in diagram)



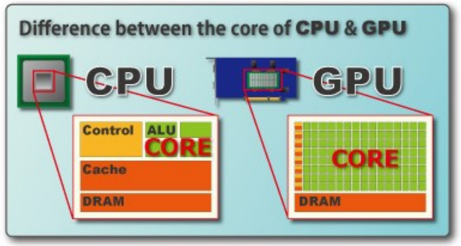


Figure 1 Floating-Point Operations per Second for the CPU and GPU

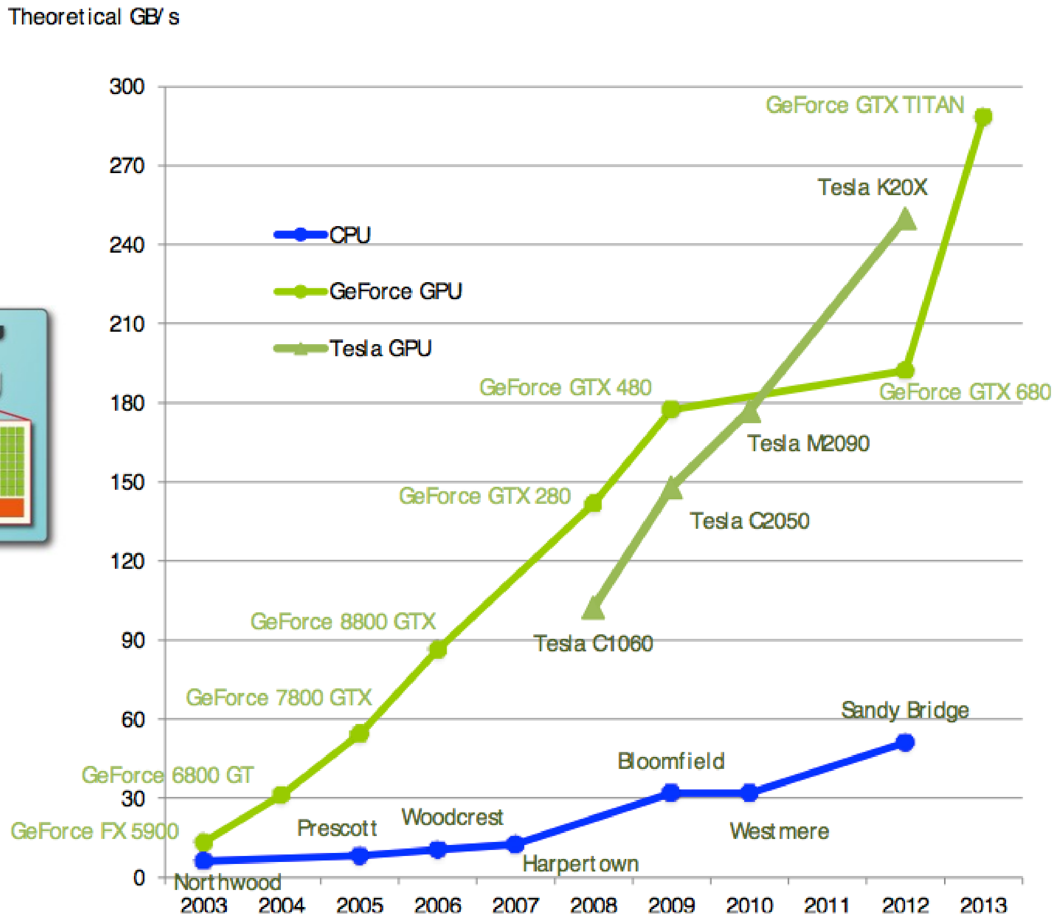


Figure 2 Memory Bandwidth for the CPU and GPU

GPU and Optimization

Can optimization problems benefit from parallel supercomputers compared to sequential machines?

LP:

$$\min p^T = x$$
subject to:
$$Ax = b$$
$$x \geq 0$$

- Interior Point method for power market applications observers a 4x speed-up from parallel implementation

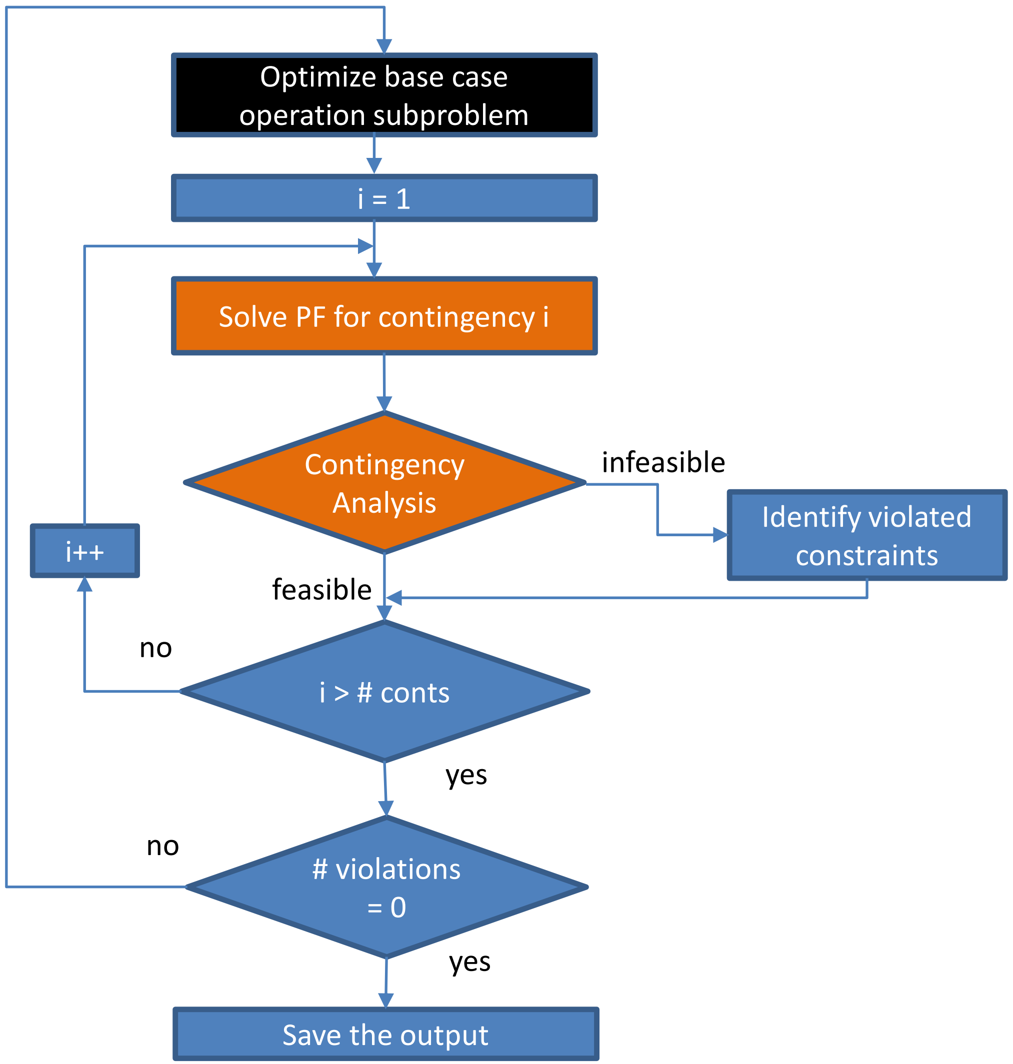
MIP:

$$\max p^T x$$
subject to:
$$Ax = b$$
$$x \geq 0$$
$$x \in \{0,1\}$$

- Complex algorithms, many logical operations
- Parallel implementations on par with sequential
- In general, problems are NP-hard (solution cannot be deterministically found within polynomial time)

Except for LP implementations, MIP problems show minimal benefit from parallelization

GPU and the SCOPF Algorithm



Hard to parallelize block

- Computational complexity: NP-hard
- Irregular data structure
- Complex algorithm relying on logical decisions
- Fits sequential (Intel) architectures

Naturally parallelizable block

- Computational complexity:
  O(# buses, #branches, #monitored elements, # contingencies)
- Performance bottleneck due to massive data
- Regular data structure
- Simple algorithm relying on floating point operations
- Fits GPU multiprocessor architectures

MISO Case Study

As part of a Phase I MISO Energy grant, NEG performed a simulation of contingency analysis for the MISO system for a single hour.

- 45,110 nodes
- 57,461 branches
- 9,364 monitored branches
- 2,324 contingencies => 21,761,936 constraints to analyze
- Current hour nodal dispatch, load, PAR settings, flow limits and generator sensitivities for the base topology are provided from the UC or ED solution
- Run on Nvidia P100 GPU card (with workstation), provided by MISO

Algorithmic Implementation

- **The Contingency Analysis algorithm relies on Flow Cancelling Transactions (FCTs) developed as part of an ARPA-E project on topology optimization**
- **All contingencies are mutually independent and naturally parallelizable**

$$\Lambda_{M,S\delta} = \Phi_{M\delta}\left(I - \Phi_{S\delta}\right)^{-1}$$

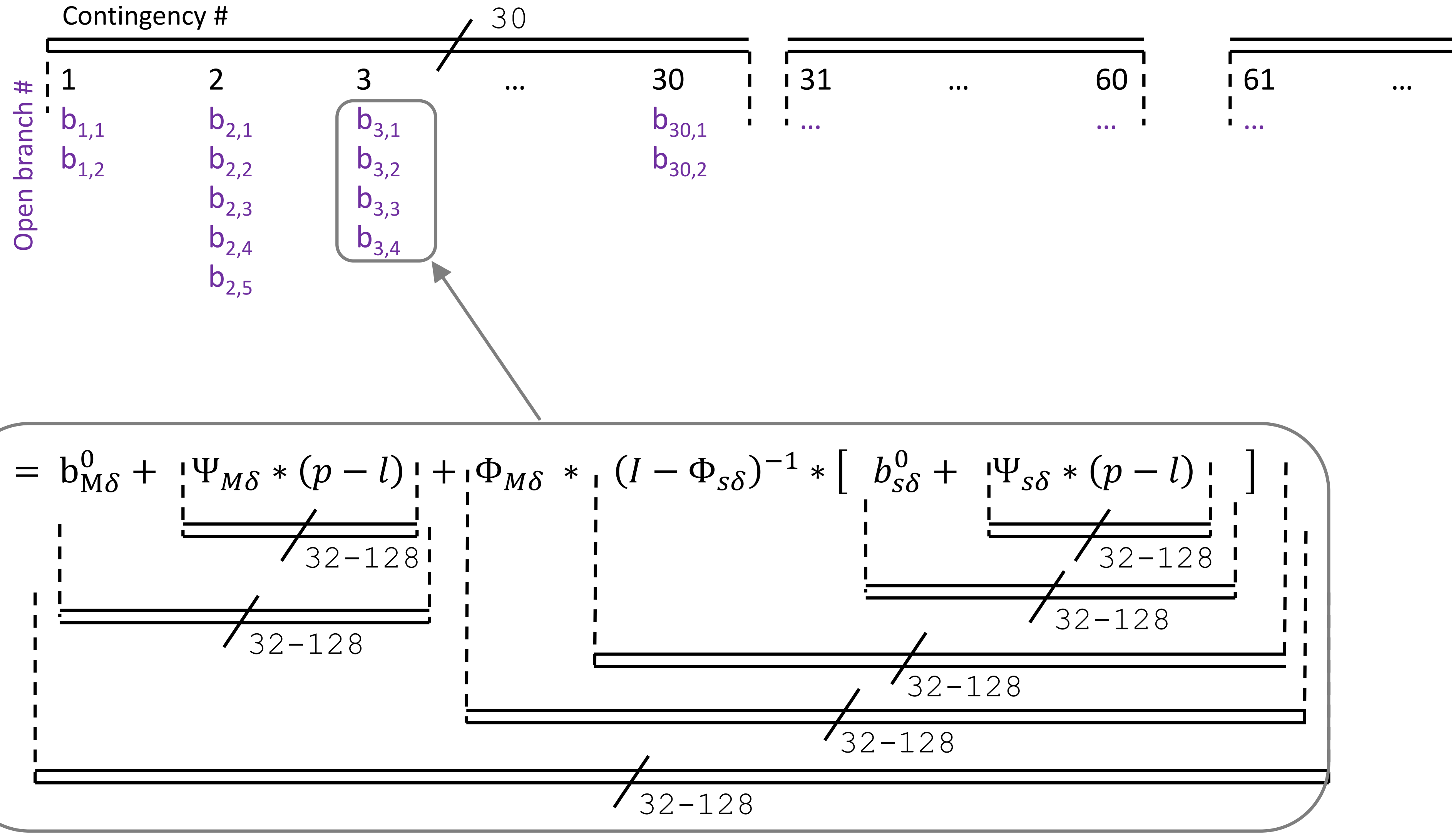
$$b_{M\delta} = b^{\circ}_{M\delta} + \Lambda_{M,S\delta} b^{\circ}_{S\delta}$$

$$f_{\delta} = b_{M\delta} + \left[\Psi_{M\delta} + \Lambda_{M,S\delta}\Psi_{S\delta}\right](p - l)$$

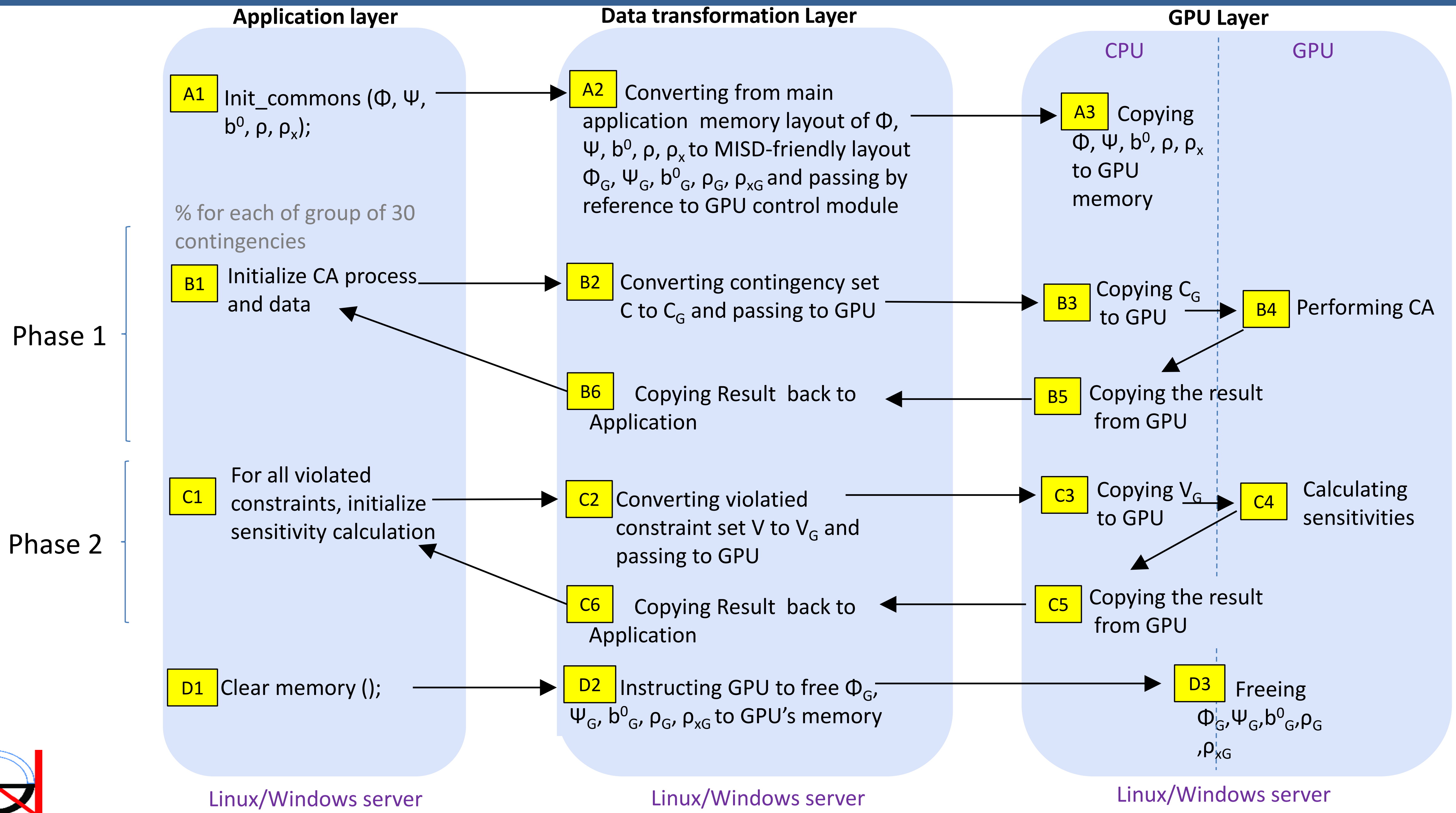
- **The algorithm solves for the flow on all monitored elements in the set M under contingency δ**
- **After identifying violations, sensitivities are calculated for violated constraints**

Contingency Analysis Parallelization on the GPU

Each contingency being run in parallel drives additional parallel matrix calculations, up to 128 threads



GPU CA Implementation in Two Phases



Performance Statistics

Out of the 21,761,936 constraints evaluated 3,266 were found to be violated

Category	Time in Seconds
Time to transfer non sensitivity data to GPU (A2 + A3)	0.77
Time to transfer the sensitivity matrices to GPU (A2 + A3)	1.19
Time to perform phase 1 on the GPU - contingency analysis (B2+B3+B4+B5)	21.05
Time to write results of phase 1: CA results and branch flows (B6*)	6.87
Time to perform phase 2 on the GPU (C2+C3+C4+C5)	6.16
Time to free memory (D1+D2)	0.17
Time to write sensitivities from phase 2 to CSV (C6*)	145.31
Total Time without writing results	**29.33**
Total Time	**180.75**

- phase 1, which calculates flows and determines violated constraints takes 21.05 seconds
- phase 2, which calculates sensitivities for each of the violated constraints takes 6.16 seconds
- Most of the time is taken up by file I/O operations, which could largely be eliminated if results were passed in memory from the gpCA application back to the core algorithm (see previous slide)
- Transfer of data from the CPU to the GPU took 1.96 seconds

Conclusions

The current Phase 1 implementation is slower than MISO's current development on HPC under the ARPA-E HIPPO project

- HPC implementation has large number of cores
- The volume of data requires GPU algorithm to solve contingencies in batches, limiting the level of parallelization.
- Need to investigate alternative methods to minimize data volume stored on GPU
- Add additional GPU card

The current Phase 2 implementation runs quickly

- Number of violated constraints generally small enough to be processed fully in parallel without batching

Questions

For Questions, Contact

John Goldis

Newton Energy Group

jgold@negll.com