Εθνικό Μετσόβιο Πολυτεχνείο
ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση

ONASSIS FOUNDATION

Accelerating SIVIA (Set Inversion via Interval Analysis). An Interval Set Membership Technique to explain Neural Classifier Decisions.

Nasiotis Konstantinos

Contents

Contents

Introduction

Interval Analysis

What is Interval Analysis?

- Interval Analysis is a field of mathematics which was created by R. E. Moore in 1962 with the purpose of integrating errors in engineering computational problems

What are some common applications of IA?

- Basically, any engineering problem can benefit from Interval Analysis.
- Applications range from path planning and global optimization to neural networks

What is SIVIA?

Set Inversion Via Interval Analysis is a technique utilizing interval analysis that can estimate the input space of non-linear functions

❖ R. E. Moore, *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis, Department of Mathematics, Stanford University, 1962.
❖ R. E. Moore, *Interval analysis*, vol. 4. Prentice-Hall Englewood Cliffs, 1966.
❖ L. Jaulin and E. Walter, *Set inversion via interval analysis for nonlinear bounded-error estimation*, *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993.

SIVIA Applications

- Parameter Estimation
  - I. Braems, F. Berthier, L. Jaulin, M. Kieffer, and E. Walter, *Guaranteed estimation of electrochemical parameters by set inversion using interval analysis*, *Journal of Electroanalytical Chemistry*, vol. 495, no. 1, pp. 1–9, 2000.
- Combinatorial & Global Optimization problems
  - E. Hansen and G. W. Walster, *Global optimization using interval analysis: revised and expanded*, vol. 264. CRC Press, 2003.
- Path Planning
  - A. Pruski and S. Rohmer, *Robust path planning for non-holonomic robots*, *Journal of Intelligent and Robotic Systems*, vol. 18, pp. 329–350, 1997.
  - L. Jaulin, “Path planning using intervals and graphs,” Reliable computing, vol. 7, no. 1, pp. 1–15, 2001.
- Neural Networks
  - Vladik Kreinovich and Andrew Bernat. *Parallel algorithms for interval computations: An introduction*. Interval Computations, 1994, 01 1994.
  - S. P. Adam, D. A. Karras, G. D. Magoulas, and M. N. Vrahatis, *Reliable estimation of a neural network’s domain of validity through interval analysis-based inversion*, in *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2015.
  - S. P. Adam, A. C. Likas, and M. N. Vrahatis, *Evaluating generalization through interval-based neural network inversion*, Neural Computing and Applications, vol. 31, no. 12, pp. 9241–9260, 2019.

Interval Analysis

What are the challenges of SIVIA?

- SIVIA works by exhaustively expanding the search space, increasing computational demand exponentially

Why acceleration?

- Parallelization can reduce computational intensity by distributing work to other processing units, in this case, GPU devices

❖ R. E. Moore, *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis, Department of Mathematics, Stanford University, 1962.
❖ R. E. Moore, *Interval analysis*, vol. 4. Prentice-Hall Englewood Cliffs, 1966.
❖ L. Jaulin and E. Walter, *Set inversion via interval analysis for nonlinear bounded-error estimation*, *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993.

Background

Interval Definitions

Interval $[x] \subset \mathbb{R}$

- Lower Bound l$b([x]) = \underline{x}$
- Upper Bound $ub([x]) = \bar{x}$
- Diameter $Width([x]) = |\bar{x} - \underline{x}|$
- Midpoint (or Center) M$id([x]) = \frac{\underline{x}+\bar{x}}{2}$

Example:

- $[x] = [\underline{x}, \bar{x}] = [3,5]$
- $Width([x]) = 5 - 3 = 2$
- $Mid([x]) = \frac{3+5}{2} = 4$

Interval Basic Operations

- Addition

$[x]+[y]=[\underline{x}+\underline{y}\,,\bar{x}+\bar{y}]$ e.g. $[3,5]+[2,5]\;=\;[5,10]$

- Subtraction

$[x]-[y]=[\underline{x}-\bar{y}\,,\bar{x}-\underline{y}]$ e.g. $[3,5]-[2,5]\;=\;[-2,3]$

- Multiplication

$[x]*[y]=[\min\{\underline{xy},\underline{x}\bar{y},\bar{x}\underline{y},\overline{xy}\},\max\{\underline{xy},\underline{x}\bar{y},\bar{x}\underline{y},\overline{xy}\}]$ e.g. $[3,5]*[2,5]\;=\;[6,25]$

- Division

$\frac{[x]}{[y]}=[\underline{x},\bar{x}]*\frac{1}{[\underline{y},\bar{y}]}$ where $\frac{1}{[\underline{y},\bar{y}]}=\left[\frac{1}{\bar{y}},\frac{1}{\underline{y}}\right]$ if $0\notin[\underline{y},\bar{y}]$ e.g. $\frac{[3,5]}{[2,5]}=[0.6, 2.5]$

Interval Set Operations

Set-Membership Operations

- isSubset

$$[x] \subseteq [y] = \underline{y} \leq \underline{x} \;\; AND \;\; \bar{y} \geq \bar{x}$$

- Intersects

$$[x] \cap [y] = \;\; (\underline{x} \leq \bar{y} \; AND \; \bar{x} \geq \underline{y} \,)$$

Interval Trigonometric Operations

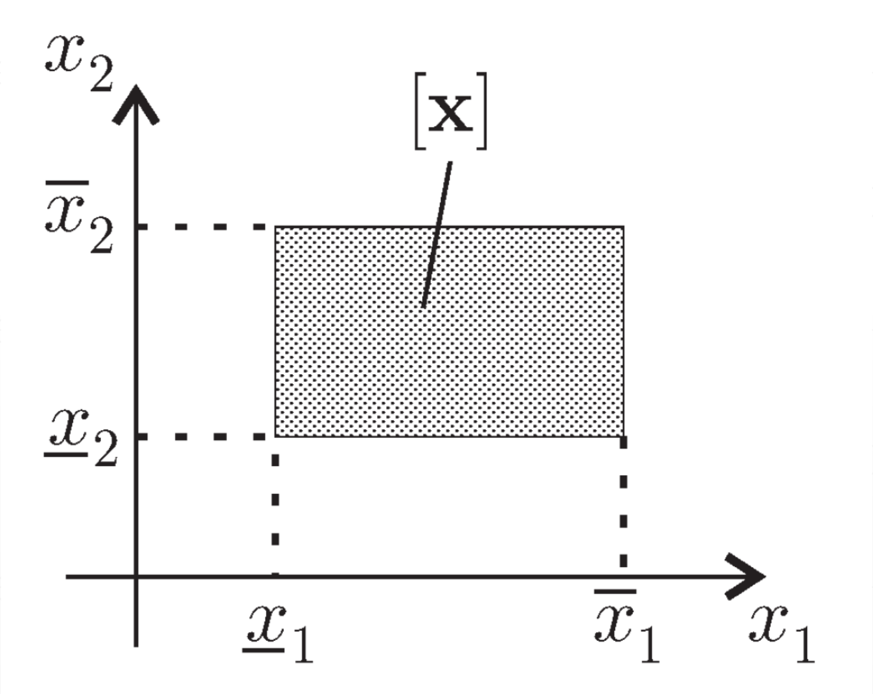
Non-linear Elementary & Trigonometric functions are also supported!

- $[x]^2$
- $\sqrt{[x]}$
- $e^{[x]}$
- $\log([x])$
- $sin([x])$
- $\cos([x])$
- $\tan([x])$

Interval Boxes

- Multidimensional Intervals or Interval Vectors

$$[X] = [[x_1], [x_2], \ldots, [x_n]]$$



Interval Matrices also supported!!

Interval Functions

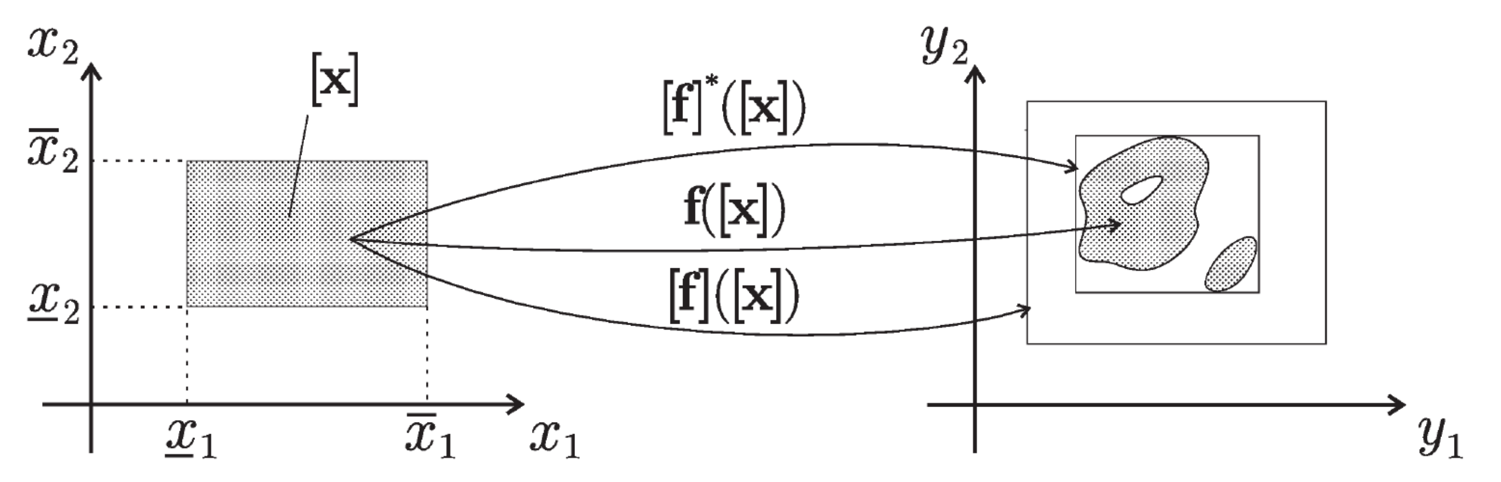
- Interval Inclusion Functions

Given f from $\mathbb{R}^n$ to $\mathbb{R}^m$, $[f]$ (from $\mathbb{IR}^n$ to $\mathbb{IR}^m$) is an inclusion function if

$$\forall [x] \in \mathbb{IR}^n \; f([x]) \subset [f]([x])$$

Input Set

Image Set



SIVIA Algorithm

SIVIA Algorithm

Input: $\boldsymbol{X_0}$ Box, incl. function $[\boldsymbol{f}](\boldsymbol{x})$, image **Y**

- Begin with an initial $\boldsymbol{X}_0$ box.

Repeat:

- Send Box to an Inclusion Function [f]([x])
- Evaluation phase - [f]([x]) $\subseteq Y$
- **Accept** - [f]([x]).isSubset(Y) (*Box is part of the solution*)
- **Reject** - ! [f]([x]).intersects(Y)
- **Bisect** - $X_0 \rightarrow X_1, X_2$
- Also <u>reject</u> width(box) $\leq \epsilon$ (*Box is too small to process*)

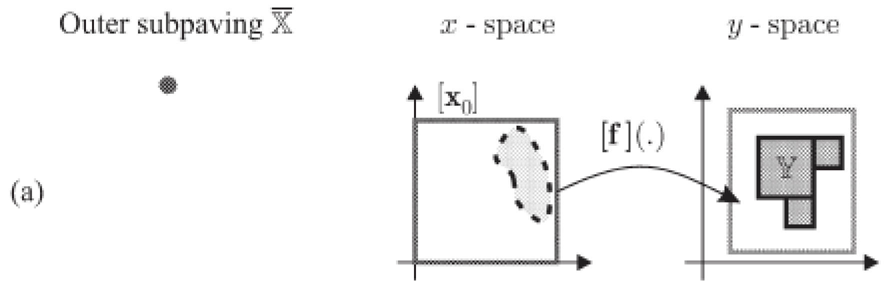
Until all boxes have been evaluated.

Output: $\mathbf{X_n}$ Boxes with their categorization labels (accepted, rejected, epsilon)

Branch and Bound Technique!!

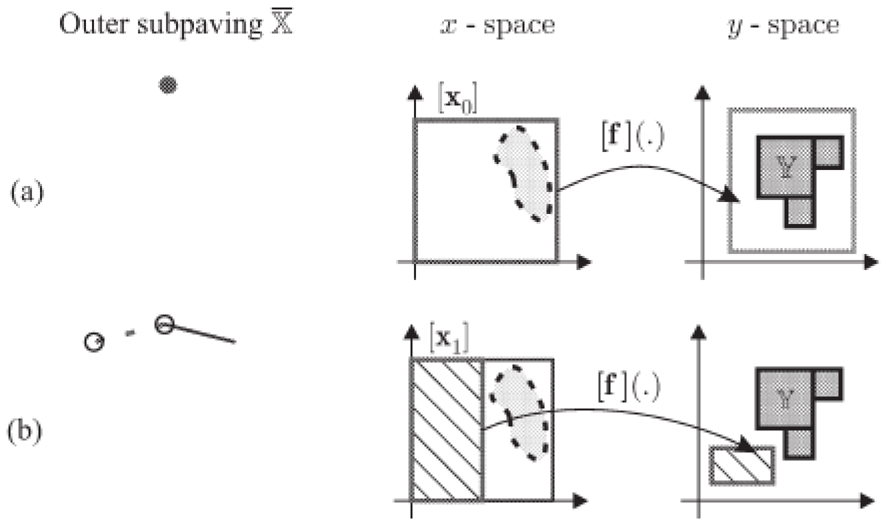
❖ *Jaulin L., Kieffer M., Didrit O., Walter E. (2001). Applied interval analysis. Springer.*
❖ L. Jaulin and E. Walter, *Set inversion via interval analysis for nonlinear bounded-error estimation*, *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993.

SIVIA Algorithm



❖ Jaulin L., Kieffer M., Didrit O., Walter E. (2001). *Applied interval analysis*. Springer.
❖ L. Jaulin and E. Walter, *Set inversion via interval analysis for nonlinear bounded-error estimation*, *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993.

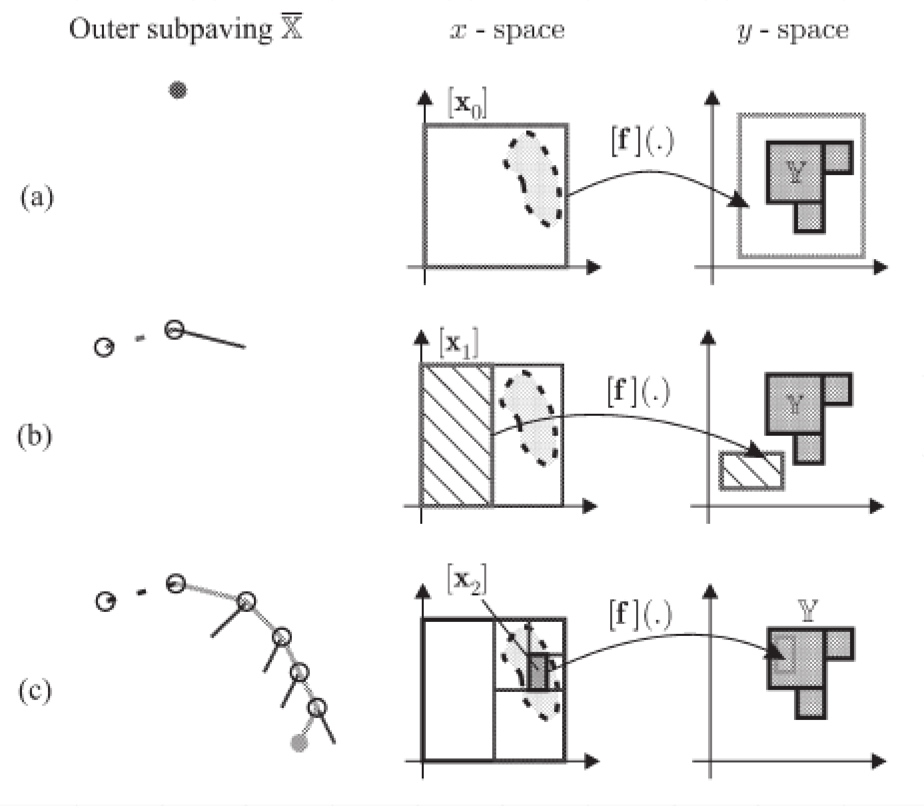
SIVIA Algorithm



❖ Jaulin L., Kieffer M., Didrit O., Walter E. (2001). *Applied interval analysis*. Springer.

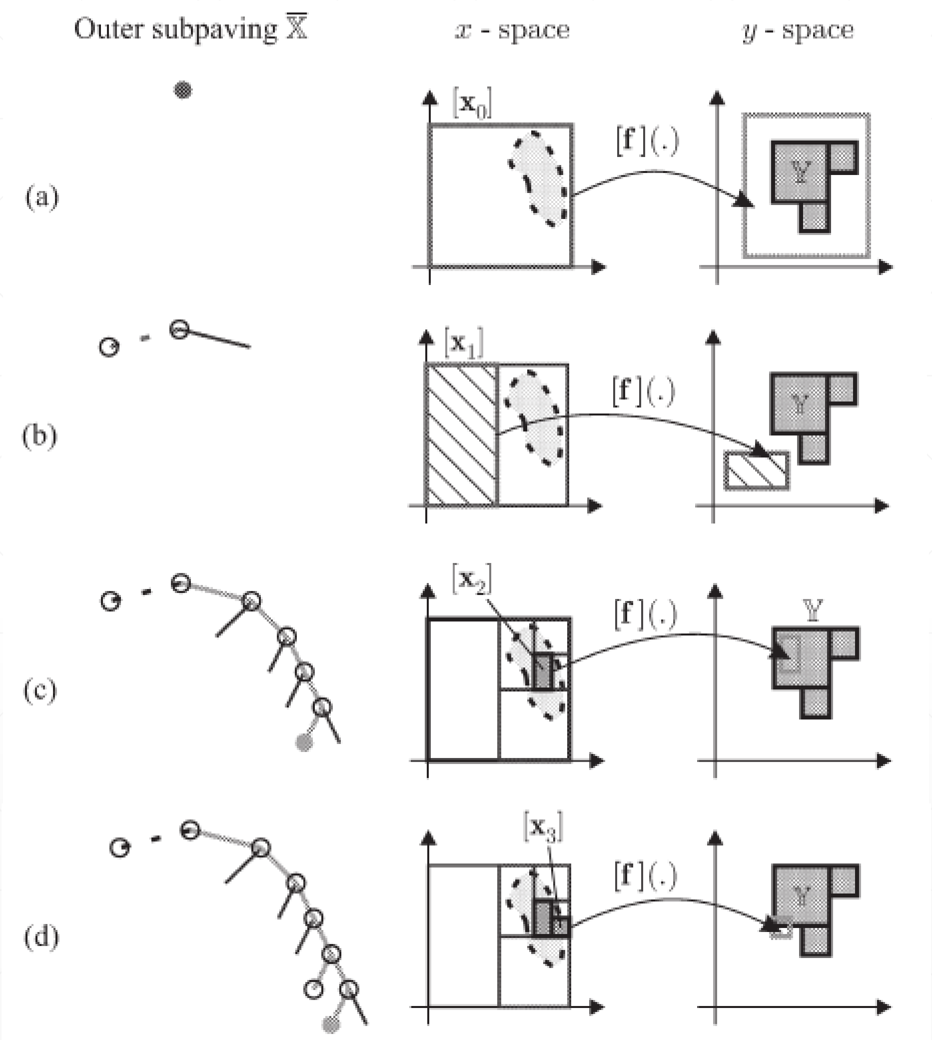
❖ L. Jaulin and E. Walter, *Set inversion via interval analysis for nonlinear bounded-error estimation*, *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993.

SIVIA Algorithm



❖ Jaulin L., Kieffer M., Didrit O., Walter E. (2001). *Applied interval analysis*. Springer.
❖ L. Jaulin and E. Walter, *Set inversion via interval analysis for nonlinear bounded-error estimation*, *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993.

SIVIA Algorithm

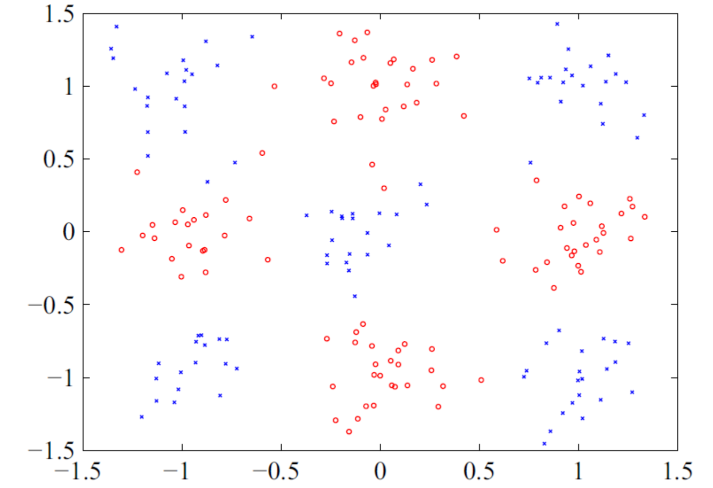


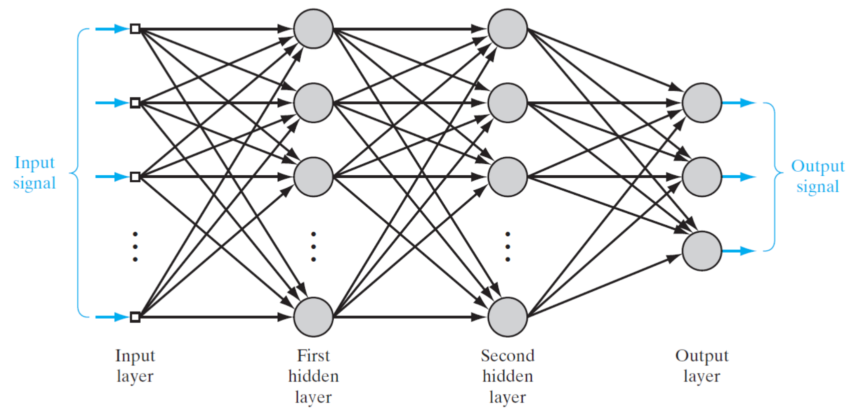
❖ Jaulin L., Kieffer M., Didrit O., Walter E. (2001). *Applied interval analysis*. Springer.
❖ L. Jaulin and E. Walter, *Set inversion via interval analysis for nonlinear bounded-error estimation*, *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993.

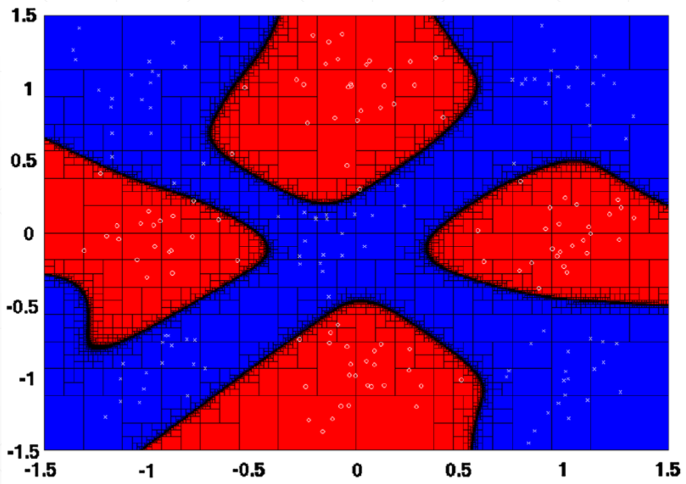
SIVIA and Neural Networks

SIVIA & Neural Networks

- SIVIA enables the estimation of non-linear spaces
- Consequently, combining it with a Neural Network enables the estimation of its input space even without knowledge of training data
- Allows the extraction of useful information such as the volume of a recognized input area







- McCulloch, Warren S., and Walter Pitts. *A logical calculus of the ideas immanent in nervous activity*. *The bulletin of mathematical biophysics* 5 (1943): 115-133.
- S. Haykin, *Neural Networks and Learning Machines.* Pearson India, 2008.
- S. P. Adam, A. C. Likas, and M. N. Vrahatis, "*Evaluating generalization through interval-based neural network inversion,*" *Neural Computing and Applications*, vol. 31, no. 12, pp. 9241–9260, 2019.

GPU Parallelism

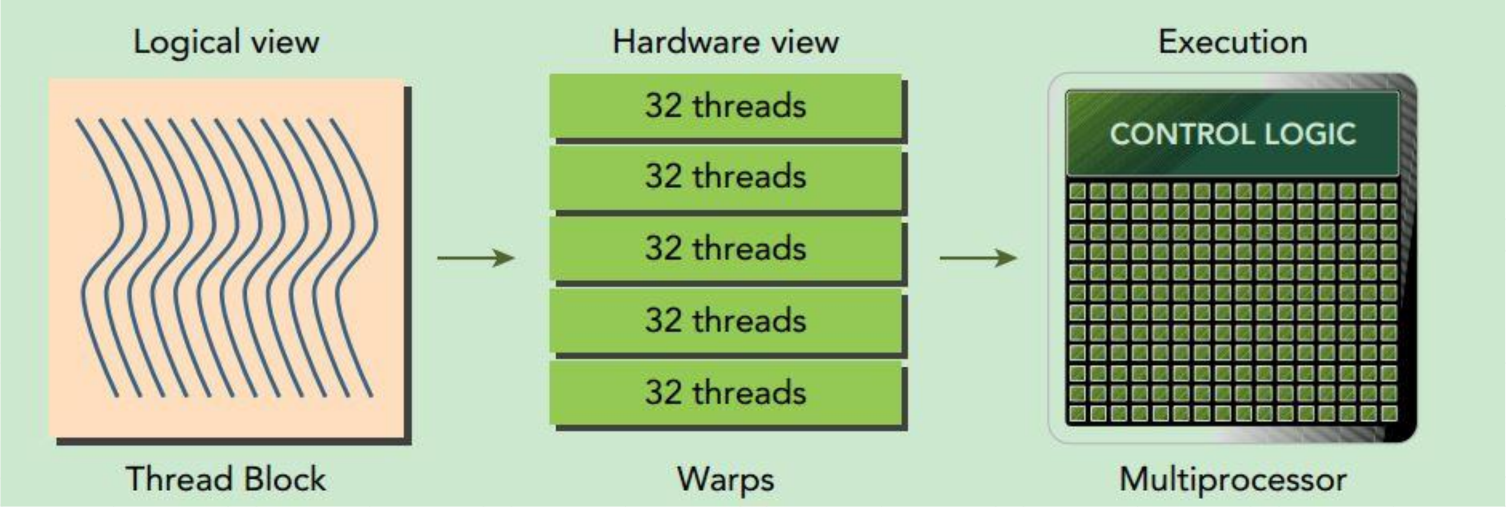
GPU Parallelism

- SIMD/SIMT: many low performance cores solving the same task in parallel
- Originally developed for computer graphics
- Primary focus on NVIDIA GPUs due to high market share

- Lorenz Gillner, Ekaterina Auer., *Interval Methods for the GPU, SWIM, 2023.*
- P. Pacheco and M. Malensek, *An introduction to parallel programming*. Morgan Kaufmann, 2021
- https://docs.nvidia.com/cuda/cuda-c-programming-guide/

CUDA Programming Model

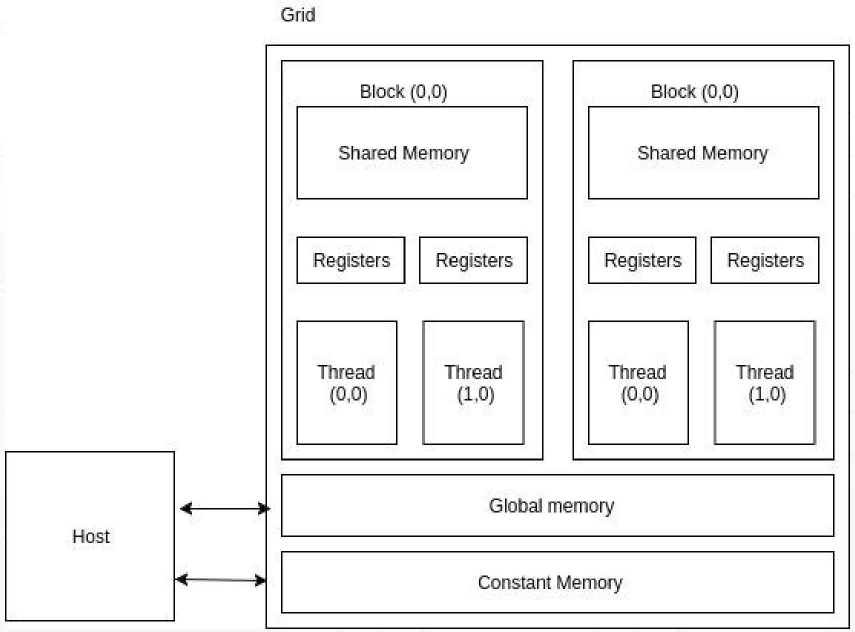
- ❖ Defined by Blocks, Threads, Warps
- ❖ Blocks are programmable
- ❖ Warps are groups of 32 Threads
- ❖ A thread is a processing (CUDA) core (max 1024 per block)



❖ Lorenz Gillner, Ekaterina Auer., *Interval Methods for the GPU, SWIM, 2023.*
❖ P. Pacheco and M. Malensek, *An introduction to parallel programming*. Morgan Kaufmann, 2021
❖ https://docs.nvidia.com/cuda/cuda-c-programming-guide/

CUDA Memory Hierarchy

- Constant, Global, Shared, Registers
- Global Memory is the slowest but largest in size – also known as VRAM
- Registers are the fastest but very small in size
- Shared Memory is on-chip slower than registers but of larger size
- Cache reduces the cost of accessing the Global Memory



- Lorenz Gillner, Ekaterina Auer., *Interval Methods for the GPU, SWIM, 2023.*
- P. Pacheco and M. Malensek, *An introduction to parallel programming*. Morgan Kaufmann, 2021
- https://docs.nvidia.com/cuda/cuda-c-programming-guide/

Implementation

CUDA Design Principles

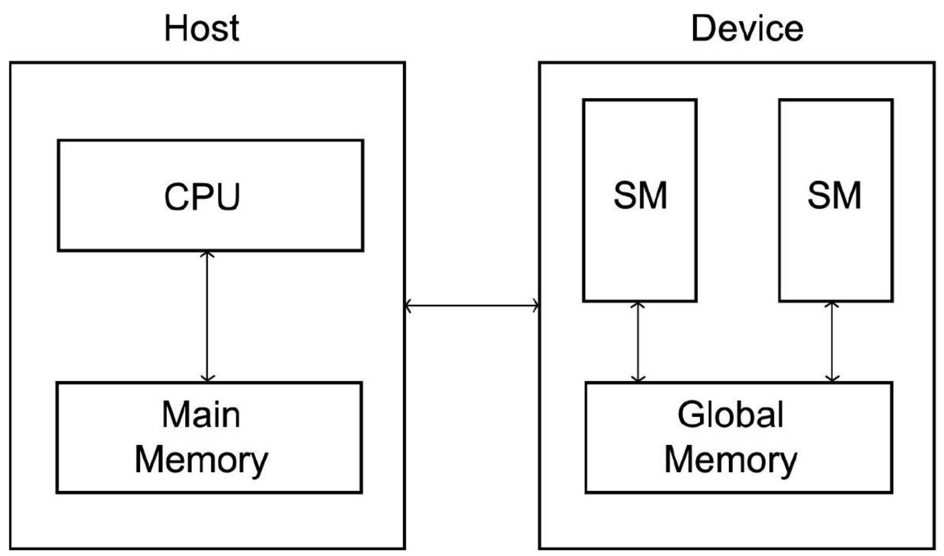
- Find ways to parallelize sequential code.
- Minimize data transfers between the host and the device.
- Adjust kernel launch configuration to maximize device utilization.
- Ensure global memory accesses are coalesced.
- Minimize redundant accesses to global memory whenever possible.
- Avoid long sequences of diverged execution by threads within the same warp.

- https://docs.nvidia.com/cuda/cuda-c-programming-guide/
- https://docs.nvidia.com/cuda/ampere-tuning-guide/index.html
- https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/

Implementation

Single GPU:

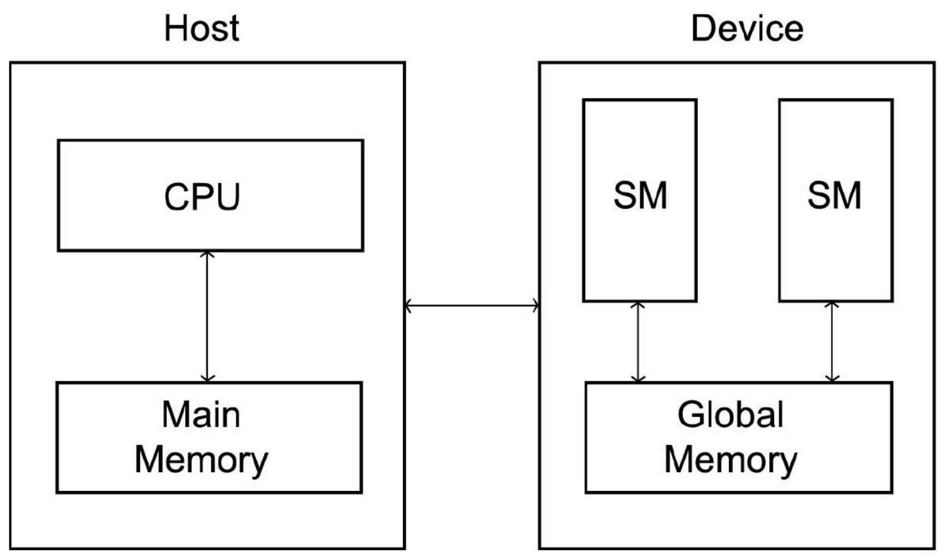
- X_0 Box initialization.
- Initialize work pool on HOST & DEVICE.
- If $\mathrm{calculateBoxes}(X_0) > \ Capacity_{device}$
  - Bisect X_0 on the HOST
- Move X_0 Box to the HOST pool.
- Copy pool from $HOST \rightarrow DEVICE$.
- Parallel Bisection. GPU
- Inclusion Function. GPU
- Set-Membership Operations. GPU
- Transfer the result $DEVICE \text{−>} HOST$ CPU



Implementation

Single GPU:

- X_0 Box initialization.
- Initialize work pool on HOST & DEVICE.
- If $\mathrm{calculateBoxes}(X_0) > \ Capacity_{device}$
  - Bisect X_0 on the HOST
- Move X_0 Box to the HOST pool.
- Copy pool from $HOST \rightarrow DEVICE$.
- Parallel Bisection. GPU
- Inclusion Function. GPU
- Set-Membership Operations. GPU
- Transfer the result $DEVICE -> HOST$ CPU



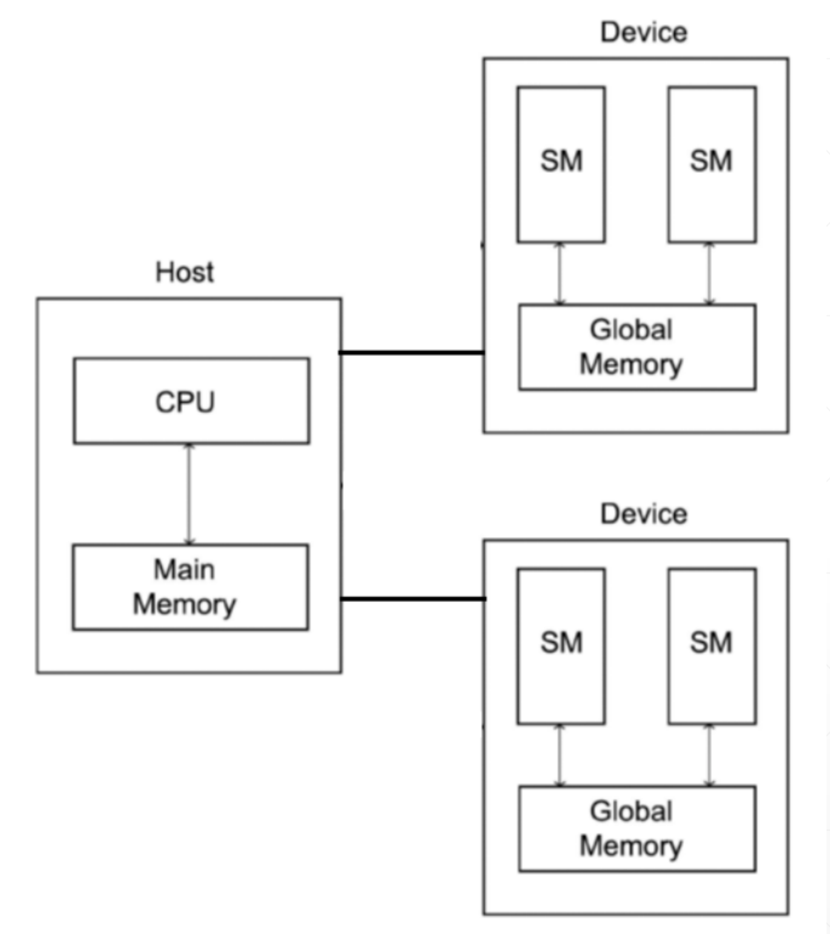
SSP Strategy!!

❖ B. Gendron and T. G. Crainic, *Parallel branch-and-branch algorithms: Survey and synthesis*, Operations research, vol. 42, no. 6, pp. 1042–1066, 1994.
❖ K. Nasiotis, D. López, S. Adam, and L. Casado, *Set inversion via interval analysis a study on parallel processing implementation*, SWIM 2019.

Implementation

Multiple GPUs:

- X_0 Box initialization.
- Initialize work pools on HOST & DEVICES.
- Bisect X_0 on the HOST
- Copy pools from $HOST \rightarrow DEVICES$.
- <u>Start threads</u> equal to number of DEVICES
- If $\text{calculateBoxes}(X_{thread}) > Capacity_{device}$ — $\boldsymbol{Thread}$
  - Bisect X_{thread} — $\boldsymbol{Thread}$
- Copy Buffers to each DEVICE Buffer — $\boldsymbol{Thread}$
- Parallel Bisection. — $\boldsymbol{GPU}$
- Inclusion Function. — $\boldsymbol{GPU}$
- Set-Membership Operations — $\boldsymbol{GPU}$
- Transfer the result. — $\boldsymbol{Thread}$



AMP/SSP Hybrid Strategy!!

❖ B. Gendron and T. G. Crainic, *Parallel branch-and-branch algorithms: Survey and synthesis*, Operations research, vol. 42, no. 6, pp. 1042–1066, 1994.
❖ K. Nasiotis, D. López, S. Adam, and L. Casado, *Set inversion via interval analysis a study on parallel processing implementation*, SWIM 2019.

Parallel Bisection

Input:

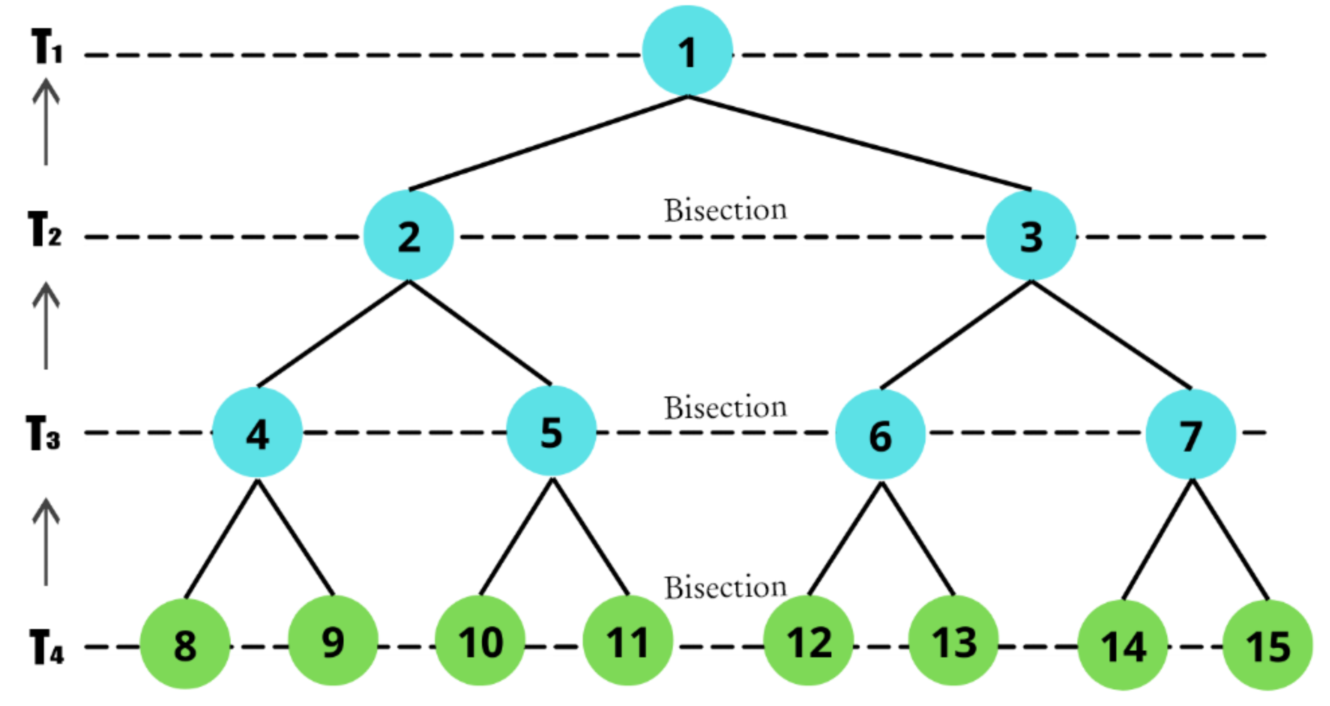
- Pool with X_0 Box at the 0th index

Algorithm:

- Host sends a number **bisect** commands to the DEVICE.
- Number of commands determined by $\mathbf{log_2(\mathit{numBoxes})}$

Output:

- Complete pool of N Boxes.



Parallel Bisection

Pros:

- Faster Execution
- Trivial memory transfer costs
- Scalable

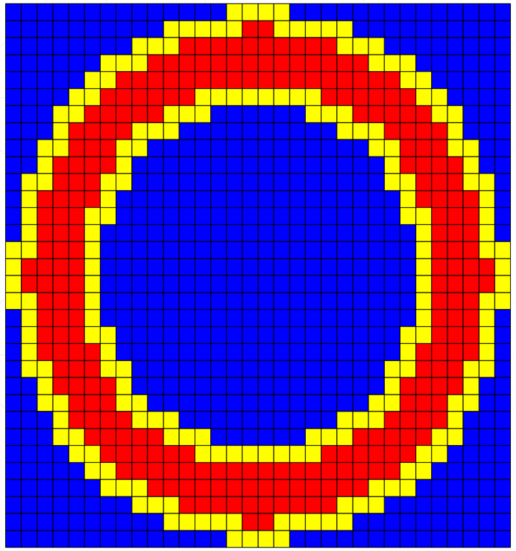
Cons:

- Wasted GPU Resources in initial executions

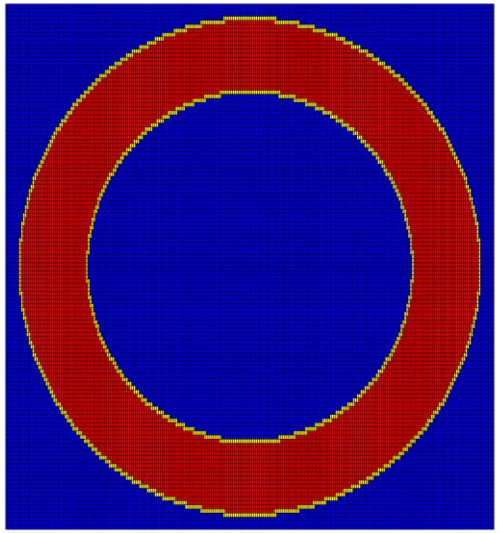
```
> GPU Device has Compute Capabilities SM 8.6

Bisection Benchmark
Initial Box ([-1.500000,1.500000],[-1.500000,1.500000])
Epsilon: 0.001 Dimensions: 2
Problem size: 16777216 boxes
Boxes: 1 | CPU Duration: 0us GPU Duration: 230us
Boxes: 2 | CPU Duration: 0us GPU Duration: 40us
Boxes: 4 | CPU Duration: 0us GPU Duration: 43us
Boxes: 8 | CPU Duration: 0us GPU Duration: 49us
Boxes: 16 | CPU Duration: 0us GPU Duration: 44us
Boxes: 32 | CPU Duration: 1us GPU Duration: 43us
Boxes: 64 | CPU Duration: 2us GPU Duration: 54us
Boxes: 128 | CPU Duration: 5us GPU Duration: 47us
Boxes: 256 | CPU Duration: 9us GPU Duration: 75us
Boxes: 512 | CPU Duration: 19us GPU Duration: 48us
Boxes: 1024 | CPU Duration: 39us GPU Duration: 43us
Boxes: 2048 | CPU Duration: 77us GPU Duration: 40us
Boxes: 4096 | CPU Duration: 216us GPU Duration: 40us
Boxes: 8192 | CPU Duration: 451us GPU Duration: 127us
Boxes: 16384 | CPU Duration: 916us GPU Duration: 43us
Boxes: 32768 | CPU Duration: 1693us GPU Duration: 43us
Boxes: 65536 | CPU Duration: 3398us GPU Duration: 46us
Boxes: 131072 | CPU Duration: 6843us GPU Duration: 56us
Boxes: 262144 | CPU Duration: 13761us GPU Duration: 201us
Boxes: 524288 | CPU Duration: 28239us GPU Duration: 236us
Boxes: 1048576 | CPU Duration: 55536us GPU Duration: 307us
Boxes: 2097152 | CPU Duration: 111522us GPU Duration: 433us
Boxes: 4194304 | CPU Duration: 225368us GPU Duration: 688us
```

Parallel Bisection

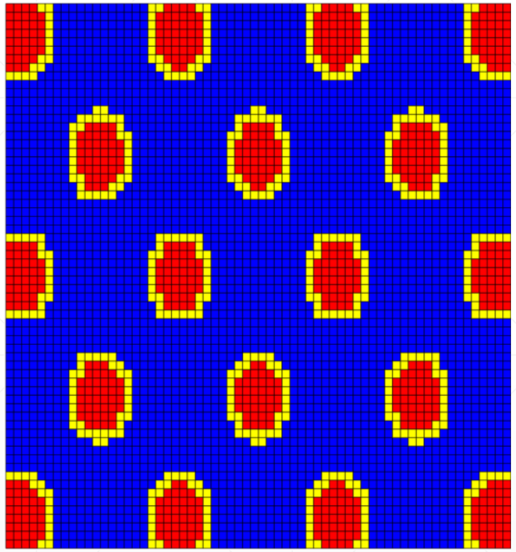
Equal Bisection —> Full expansion of the problem tree



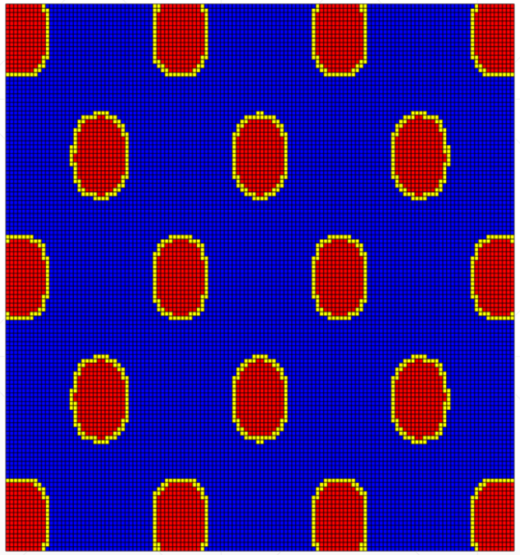
$e = 0.1$



$e = 0.01$



$e = 0.4$



$e = 0.2$

Inference & Set Estimation

Input:

- Vector of Boxes – Work Pool
- Empty Label Vector - Problems 1&2

Algorithm:

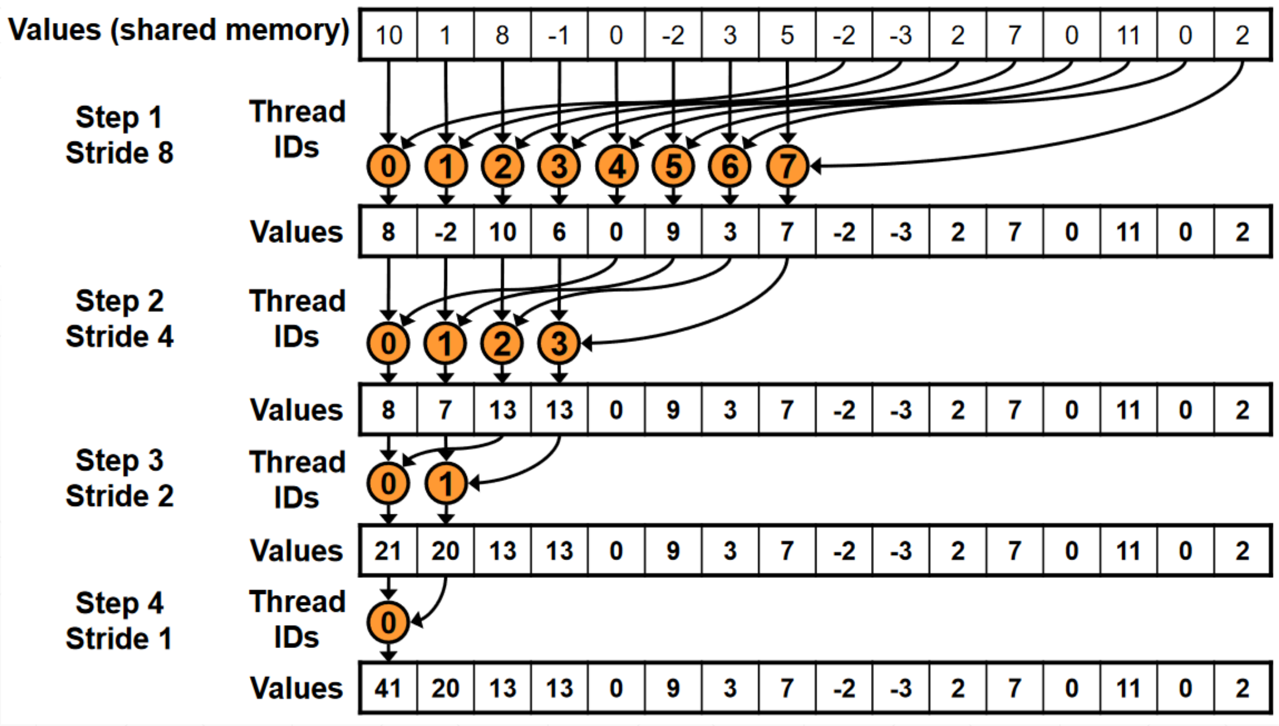
- Coarse-grained approach
- Each Box processed independently. ([f]([x]) and set-membership operations)
- Optimized to avoid Warp Divergence. (replaces "if" clauses with Boolean operations)
- Parallel Reduction – *Problem 3*

Output:

- Label Buffer filled with evaluation labels (included, discarded, epsilon) – *Problems 1&2*
- Volume of domain of validity – *Problem 3*

Parallel Reduction

- Without Warp Divergence
- Sequential Addressing (cache friendly)



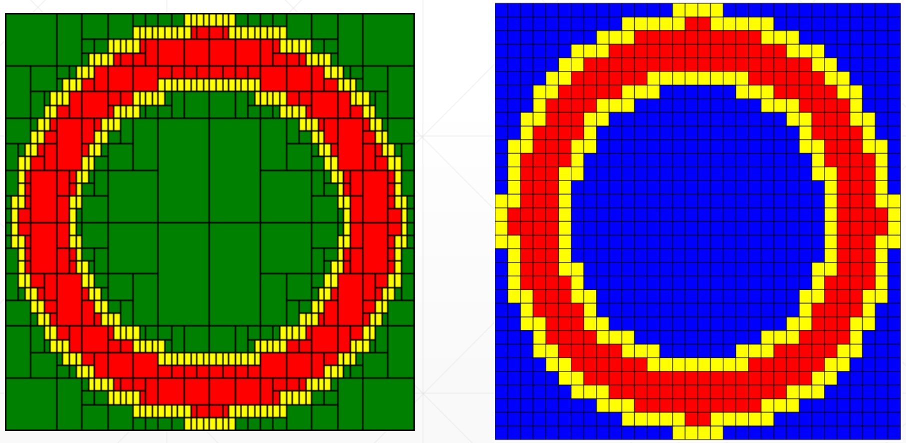
- M. Harris et al., *Optimizing parallel reduction in cuda,* Nvidia developer technology, vol. 2, no. 4, p. 70, 2007.
- https://github.com/NVIDIA/cuda-samples - *Reduction #3*

Results

Problems 1 & 2

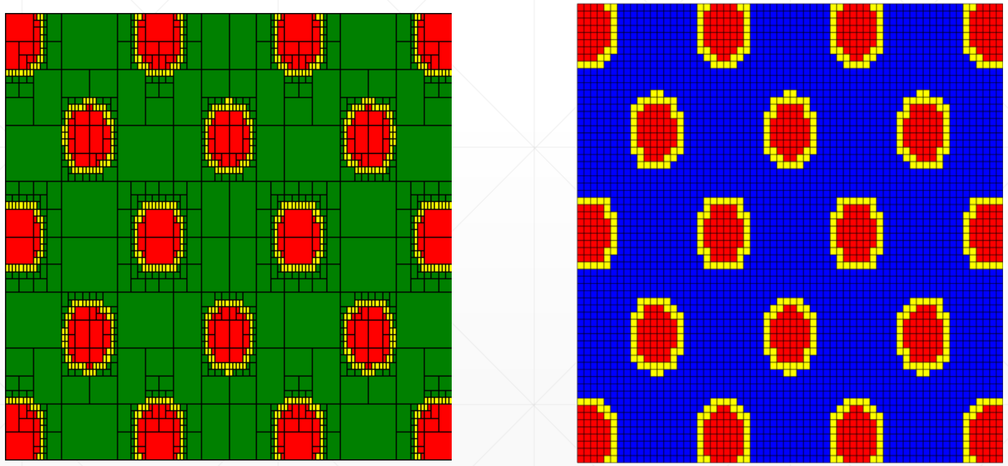
❖**Problem 1**: 2D Torus

$$[f]([x]) = [x]^2 + [y]^2$$
$$[x]_0 = [-1.5,1.5],[-1.5,1.5]$$
$$[y] = [1,2]$$



❖ **Problem 2**: 2D Griewank

$$[\mathrm{x}_0] = [-10,10]^2$$
$$[\mathrm{y}] = [1.5,3]$$
$$[f]([x]) = \Sigma_{i=1}^{2} \frac{[x]_i^2}{4000} - \Pi_{i=1}^{2} \cos\left(\frac{[x]_i}{\sqrt{i}}\right) + 1$$



- *Problem Requirement: **The whole input space***

Problem 3

❖ Evaluating Generalization Performance of Neural Classifiers

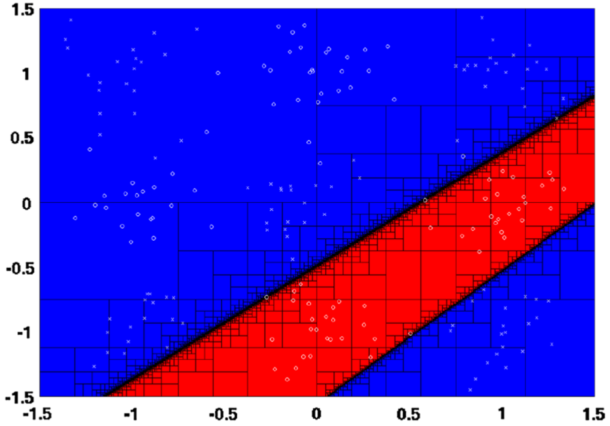
- $V_{net} = \sum_{i=1}^{M} V_i$
- $G_{net} = \frac{V_{net}}{V_{input}} - \frac{l}{P}$
- $V_{input} = \prod_{i}^{N} |x_i^{max} - x_i^{min}|$
- *Problem Requirement:* ***A single value***

Consequently:

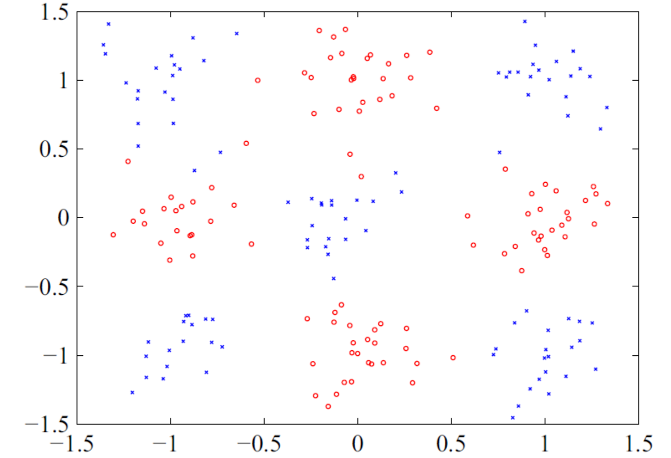
❖ $[f]([x]): 6-30-2\ MLP$ Weights provided by [1], Vertebral Column Dataset, example with early stopping

❖ $[x]_0 = [-1,1]^6$

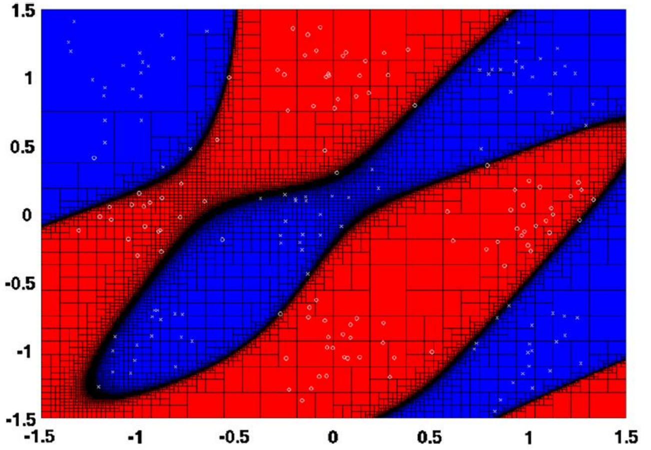
❖ $[y] = [0.8,1]$



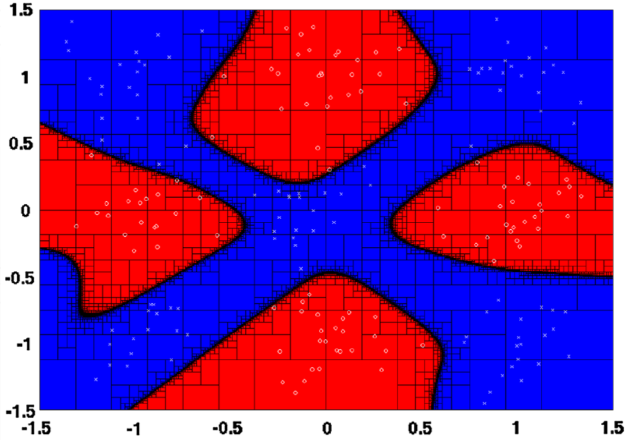
Undertraining



Sample data



“Proper” fit



Overtraining (overfitting)

❖ [1] S. P. Adam, A. C. Likas, and M. N. Vrahatis, “*Evaluating generalization through interval-based neural network inversion,” Neural Computing and Applications*, vol. 31, no. 12, pp. 9241–9260, 2019.

❖ S. P. Adam, D. A. Karras, G. D. Magoulas, and M. N. Vrahatis, *Reliable estimation of a neural network’s domain of validity through interval analysis-based inversion*, in *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2015.

Test Configurations

Home Setup

- GPU: Nvidia RTX 3060 Ti 8GB
- OS: WSL2 Ubuntu 22.04.2 LTS

Microlab DaVinci of NTUA:

- GPU: Nvidia Tesla V100 32GB
- OS: Ubuntu 20.04.2 LTS

Google Collab:

- GPU 1: Nvidia A100 40GB
- GPU 2: Nvidia Tesla V100 16GB

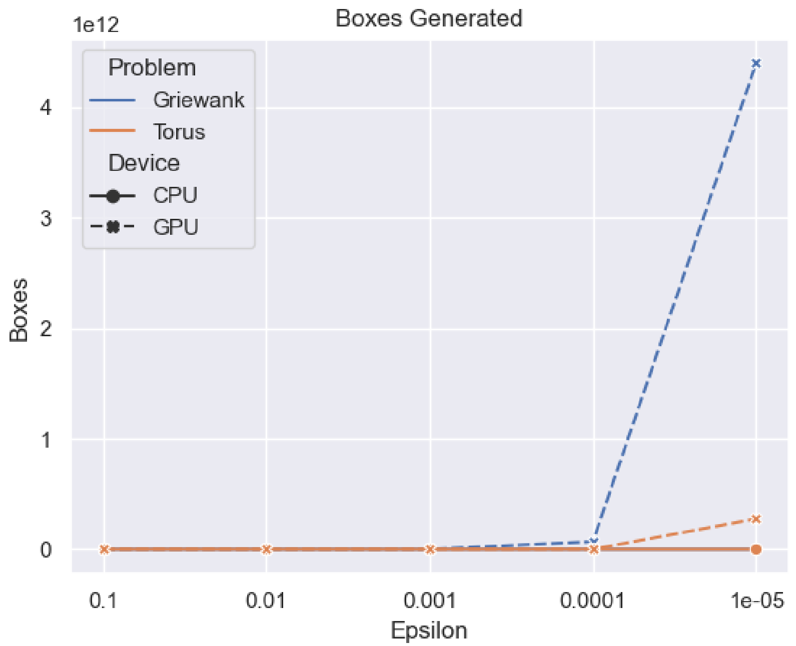
CEID of UPATRAS:

- GPU: 8x Nvidia A100 32GB
- OS: Ubuntu 22.04.3 LTS

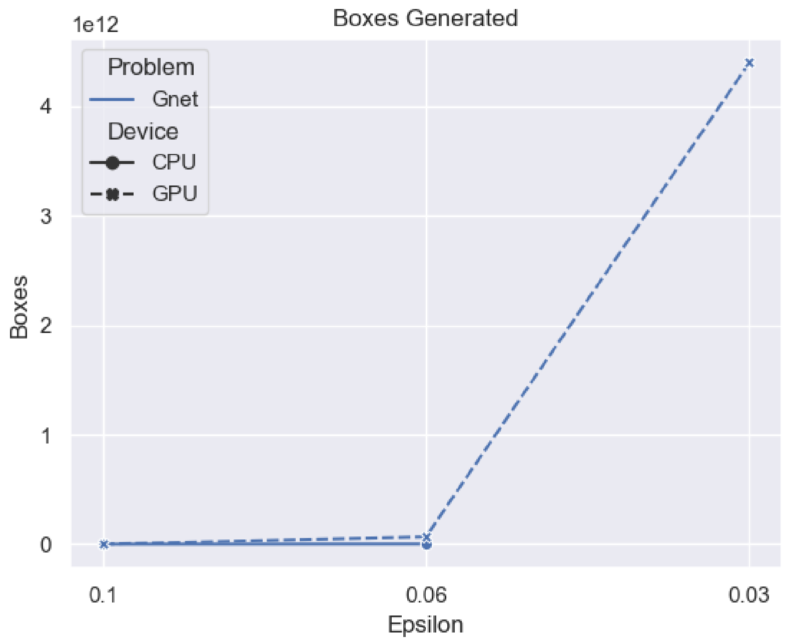
Problem Size

- Parallel implementation computes a larger number of boxes
- Missing value on problem 3 due to long execution time required to exit execution

Problems 1&2



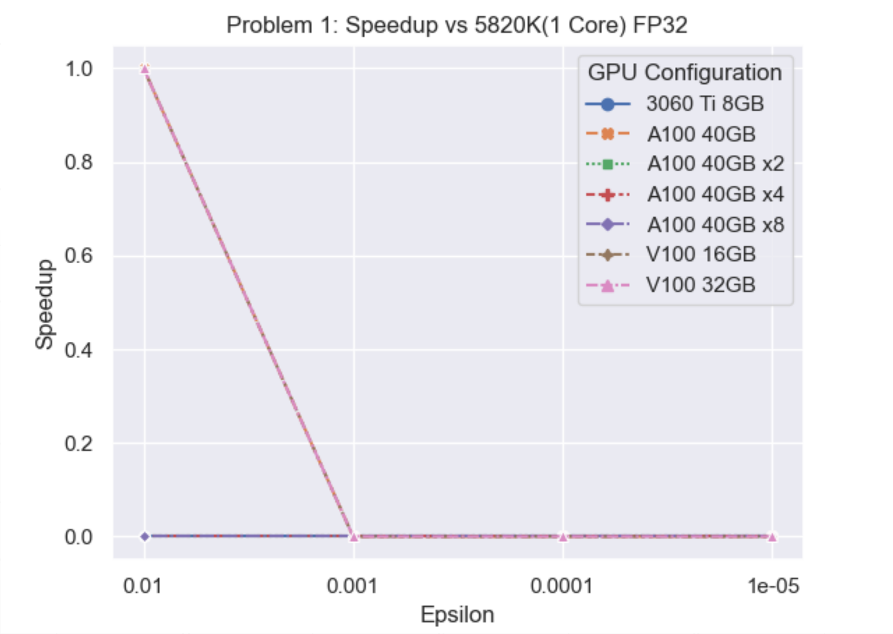
Problem 3

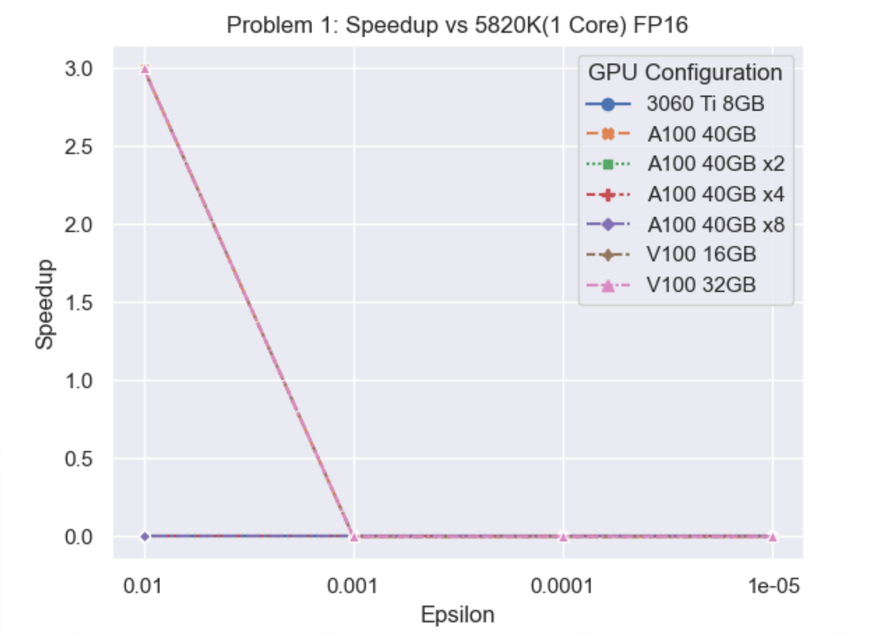


Results: Problem 1

Speedup

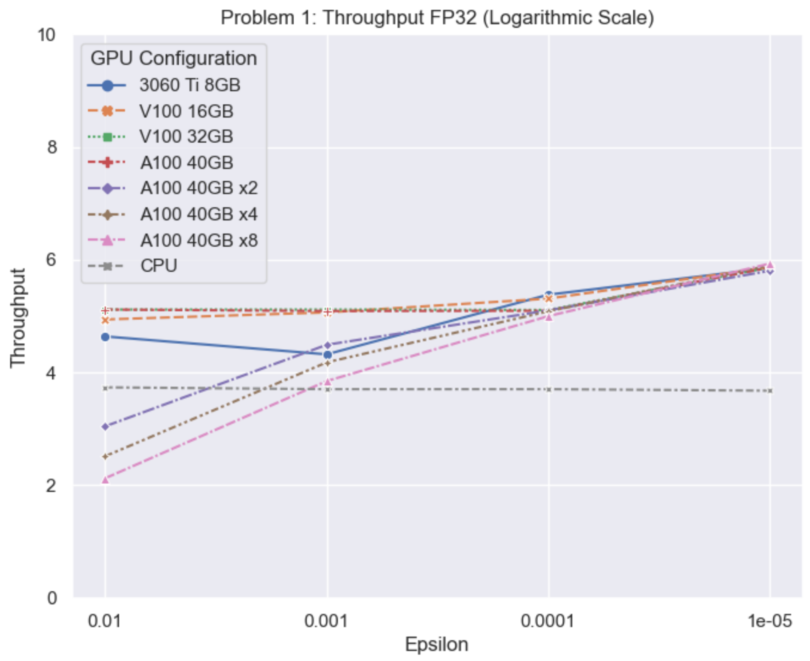
- Comparison to sequential SIVIA using an i7 5820k @ 4.3Ghz
  - Slowdowns due to intensive memory transfers between host and device

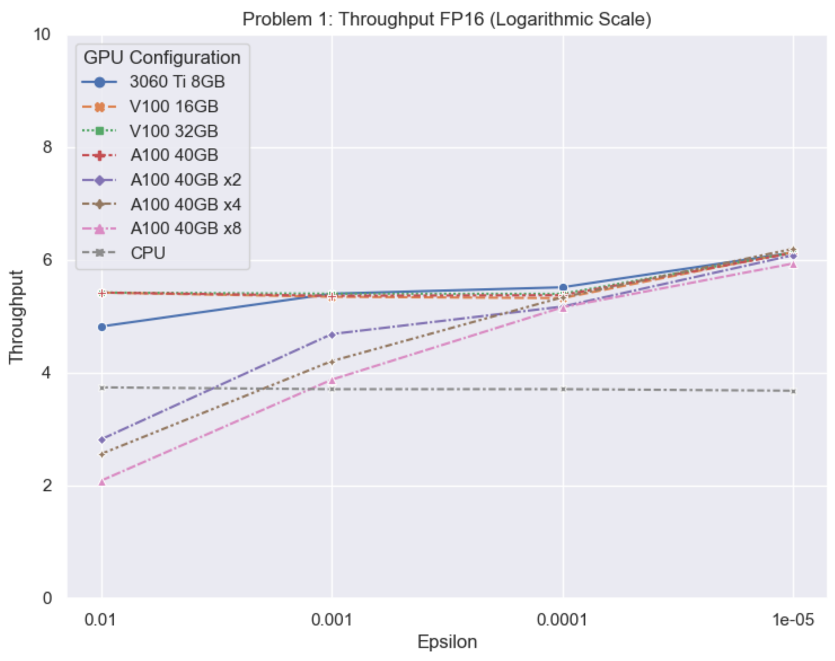




Throughput

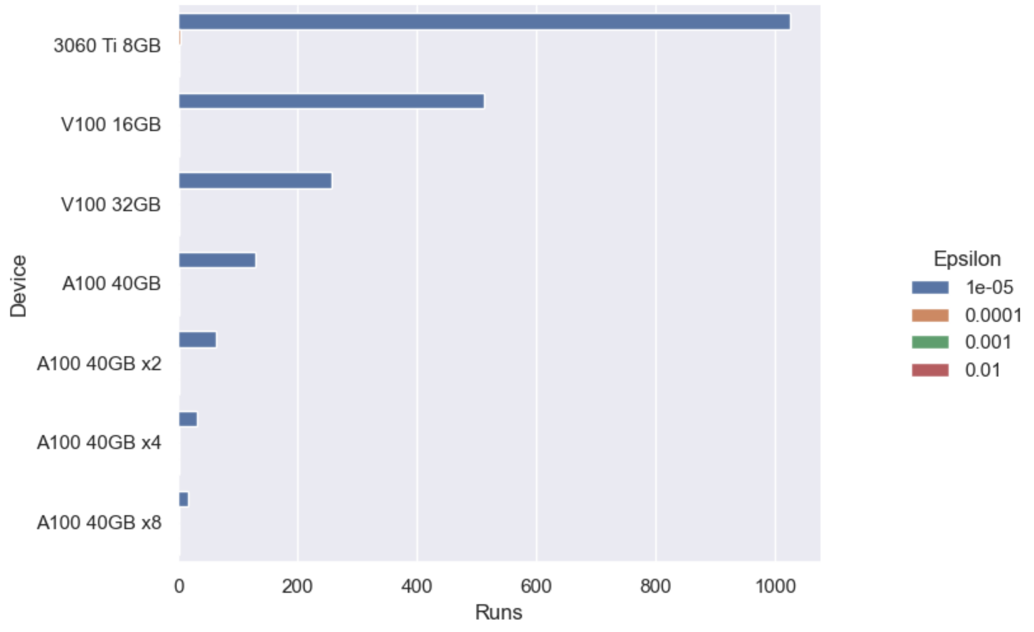
- CPU has way smaller throughput
- Half variables increase throughput
  - Slight depiction due to logarithmical scaling



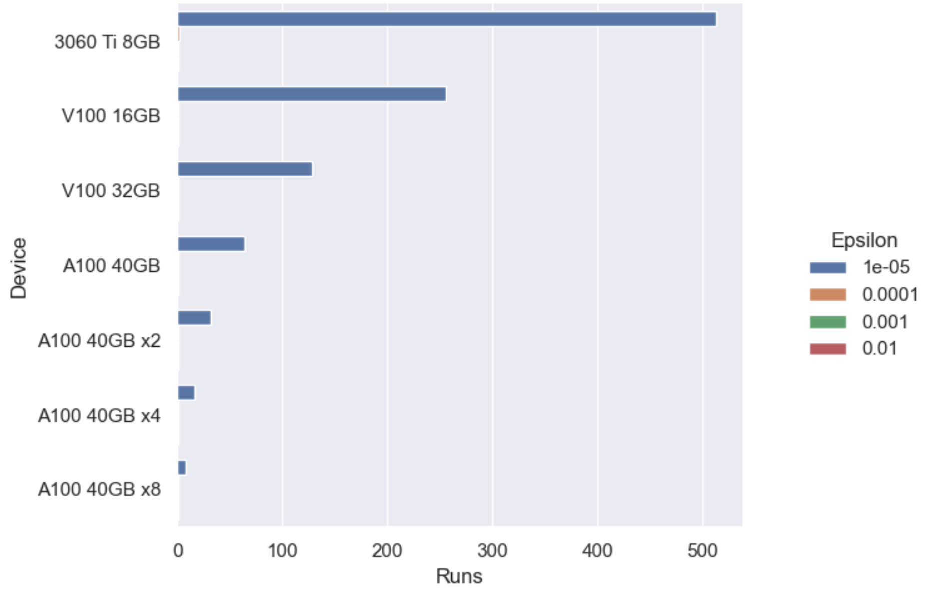


Kernel Runs

- Problem 1
  - More VRAM -> More Boxes fit -> less GPU executions



- FP32

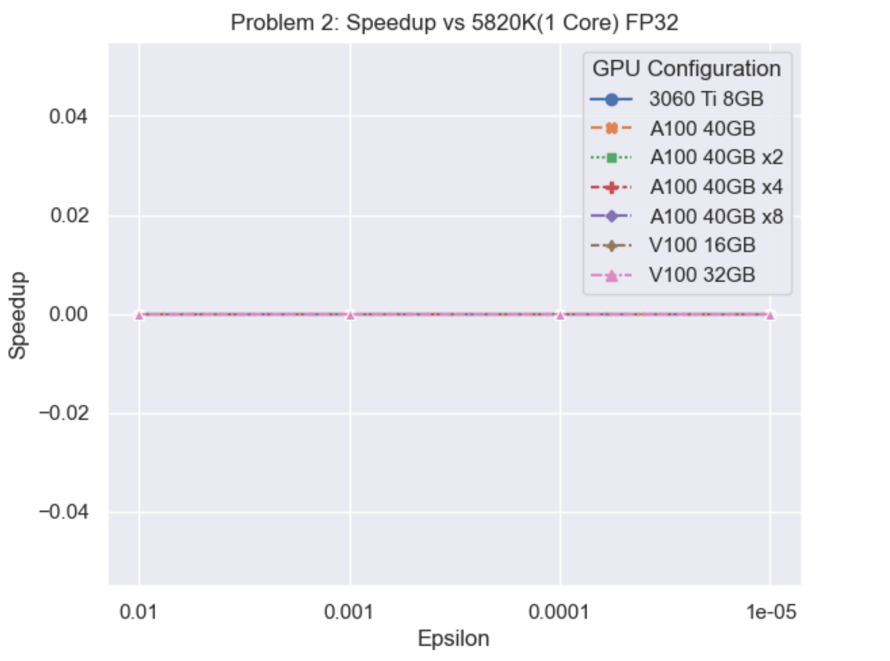


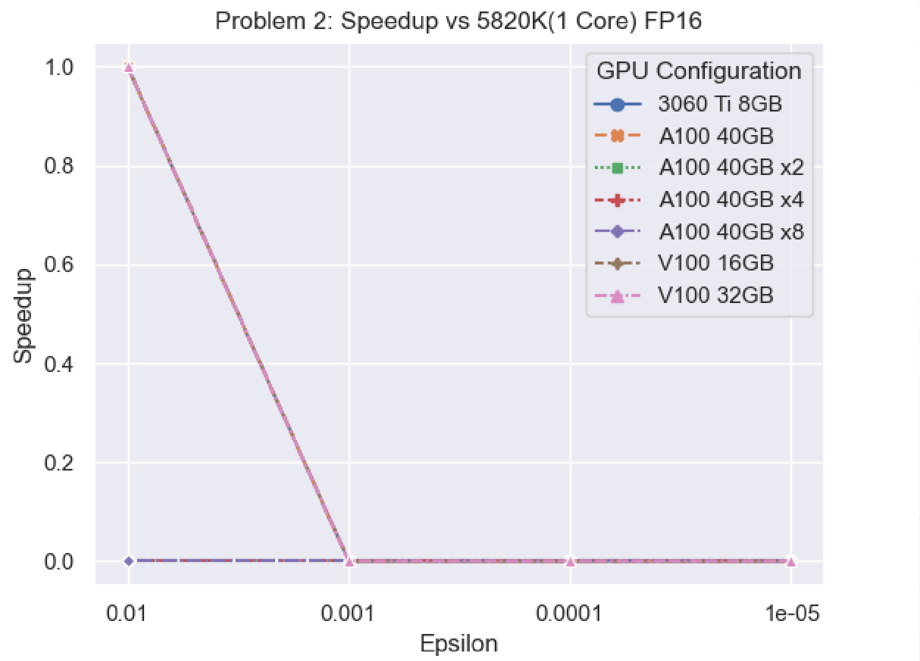
- FP16

Results: Problem 2

Speedup

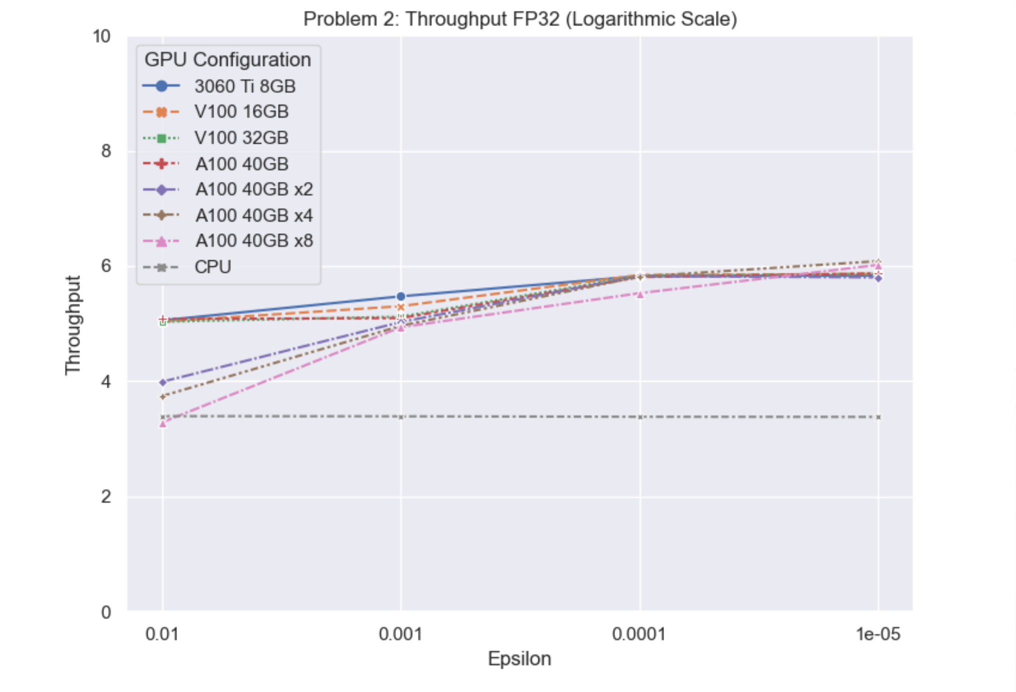
- Similar findings as with Problem 1

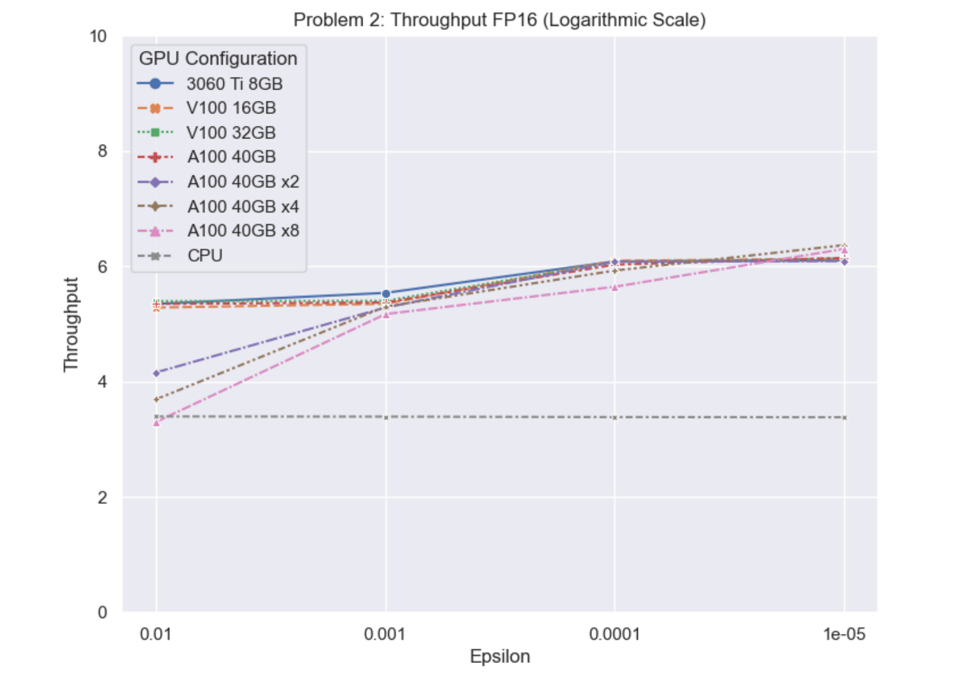




Throughput

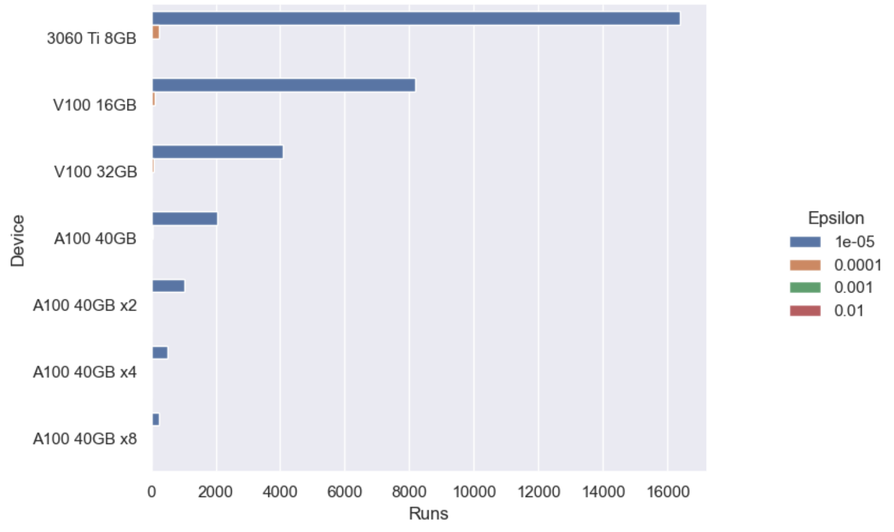
- Throughput findings of Problem 1 also extend to Problem 2
- Half variables increase throughput

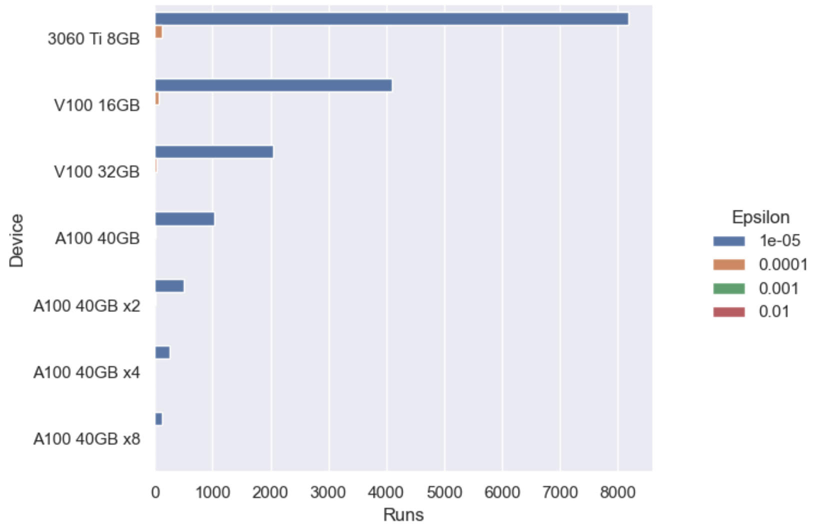




Kernel Runs

- Problem 2
  - Similar behavior to Problem 1, more VRAM mean less GPU executions



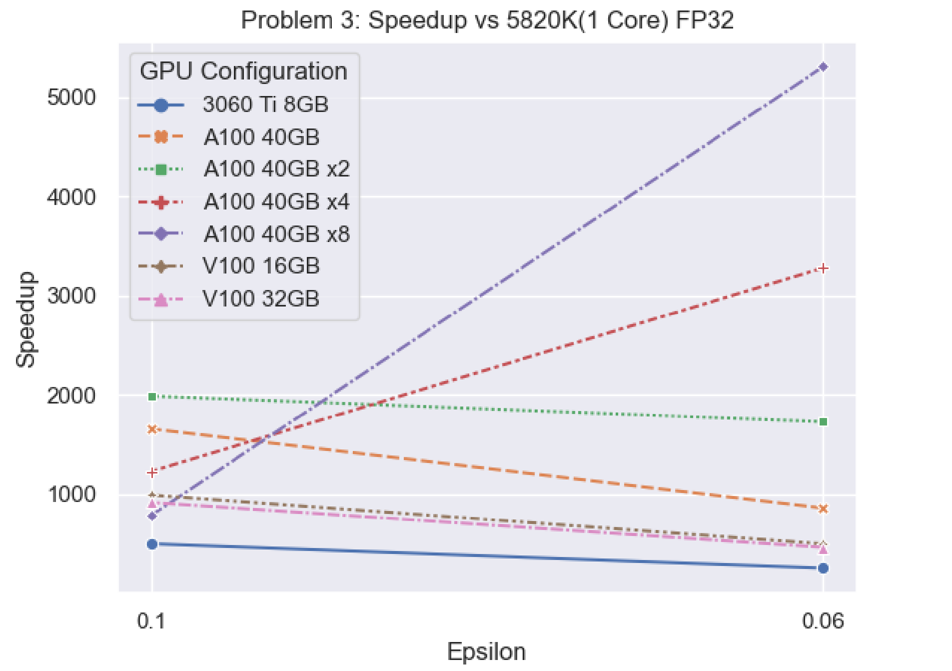


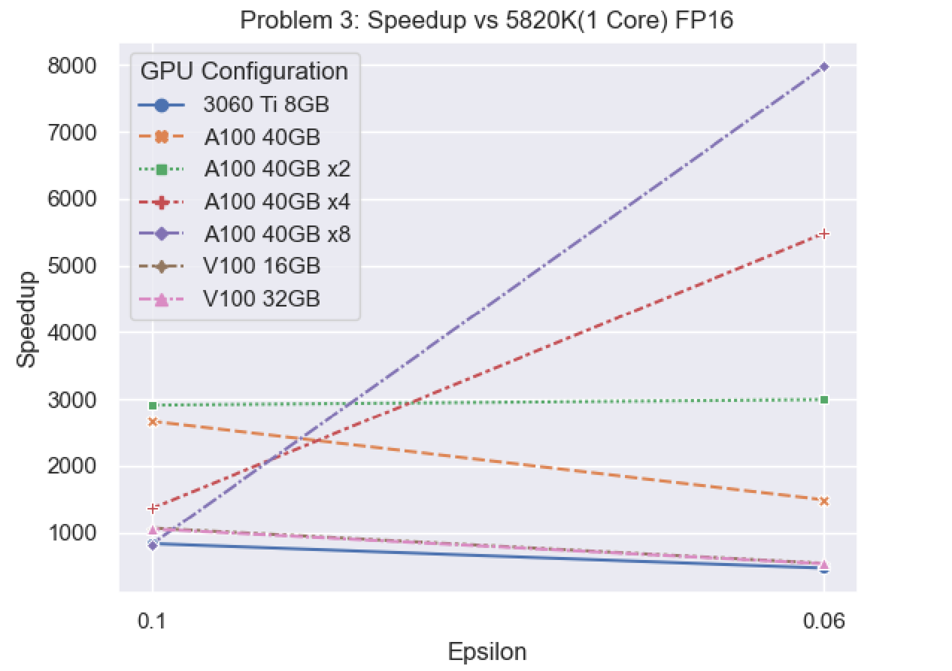
- FP32
- FP16

Problem 3

Speedup

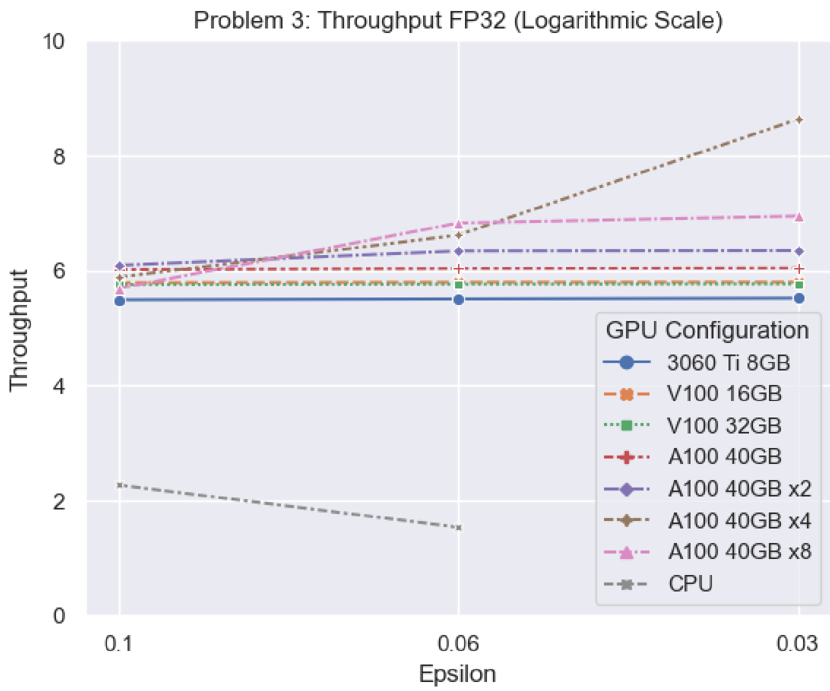
- **Key result**: 5000+ speedup when using 8x A100 GPUs
  - 8000 with FP16 variables

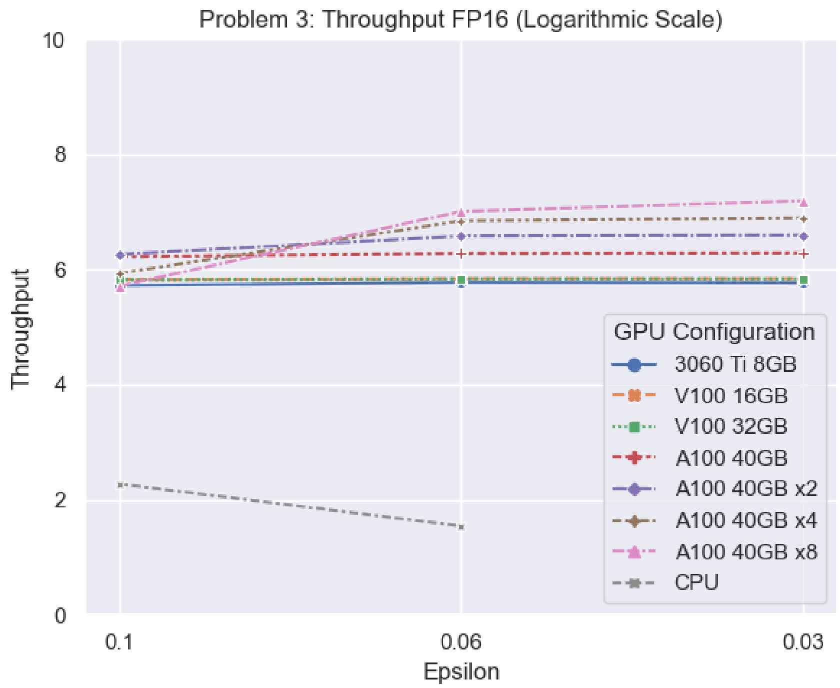




Throughput

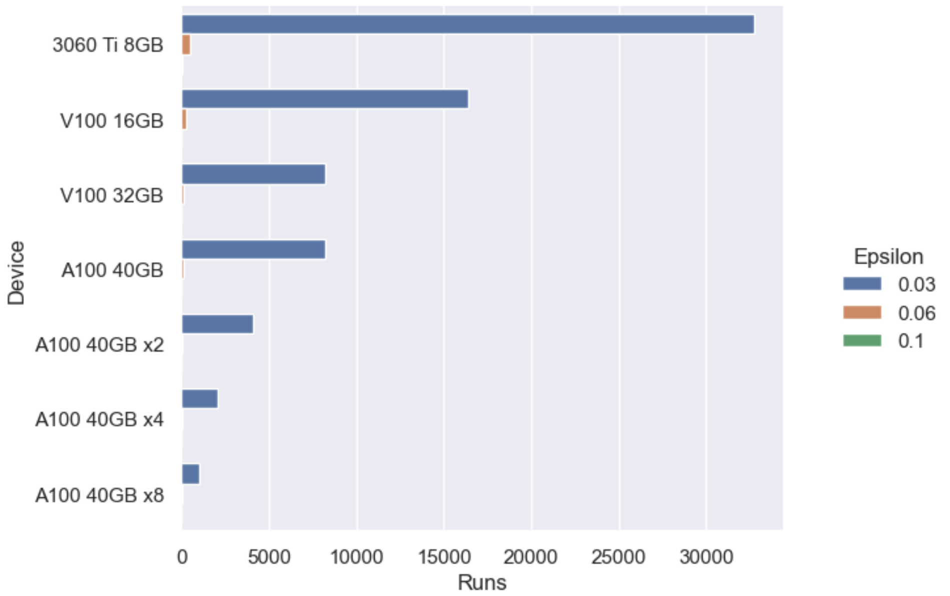
- Speedup findings align with the throughput
- CPU throughput shrinks with increasing problem size



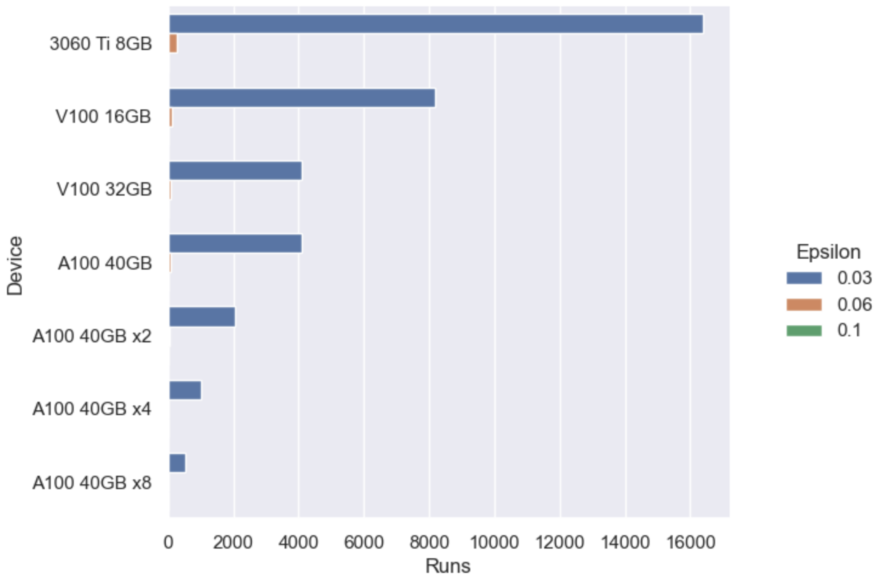


Kernel Runs

- Problem 3
  - Like problems 1&2, less VRAM = More executions



- FP32



- FP16

Profiling

Profiling

- Nvidia profiler validates assumptions of previous results

- Problem 2
  - $e = 0.001$

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
85.2	5697886176	9	633098464.0	104344400.0	56500	3964895784	1317343026.4	cudaMemcpy
10.8	722301100	1	722301100.0	722301100.0	722301100	722301100	0.0	cudaMallocHost
2.5	168768499	60	2812808.3	43000.0	20200	51632499	9549282.8	cudaDeviceSynchronize
0.6	43053000	3	14351000.0	4908400.0	293300	37851300	20482257.3	cudaFree
0.6	42401600	3	14133866.7	11234500.0	550600	30616500	15241205.0	cudaMalloc

- Problem 3
  - $e = 0.06$

** CUDA API Summary (cuda_api_sum):

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
99.1	114912155468	7681	14960572.3	54200.0	2600	374076197	59393960.7	cudaDeviceSynchronize
0.6	684587990	1	684587990.0	684587990.0	684587990	684587990	0.0	cudaMallocHost
0.2	217664907	7680	28341.8	12800.0	6300	4600899	62991.5	cudaLaunchKernel
0.1	58488800	10	5848880.0	3500.0	2100	58446300	18480850.2	cudaFree
0.0	44460599	40	1111515.0	2050.0	1600	43923599	6943062.0	cudaMalloc
0.0	24409100	549	44461.0	40600.0	9500	317700	34166.6	cudaMemcpy

Conclusion

Conclusion

- GPU parallelization is very compatible with SIVIA when intensive memory transfers are not a requirement
- Mathematical Techniques should accompany parallelization efforts
  - E.g. Optimization problems have their search space reduced by calculating the hull of an iteration's *best* Boxes
- Proposed implementation not a one-size-fits-all technique
- Scales very well with multiple GPUs

Future Work

- Expansion of GPU Interval Ecosystem
- Better Fine-tuning and resource allocation
- Test algorithm with optimization problems
- Experiment with in-GPU synchronization strategies
- Test and compare modern parallel interval environments[1][2]

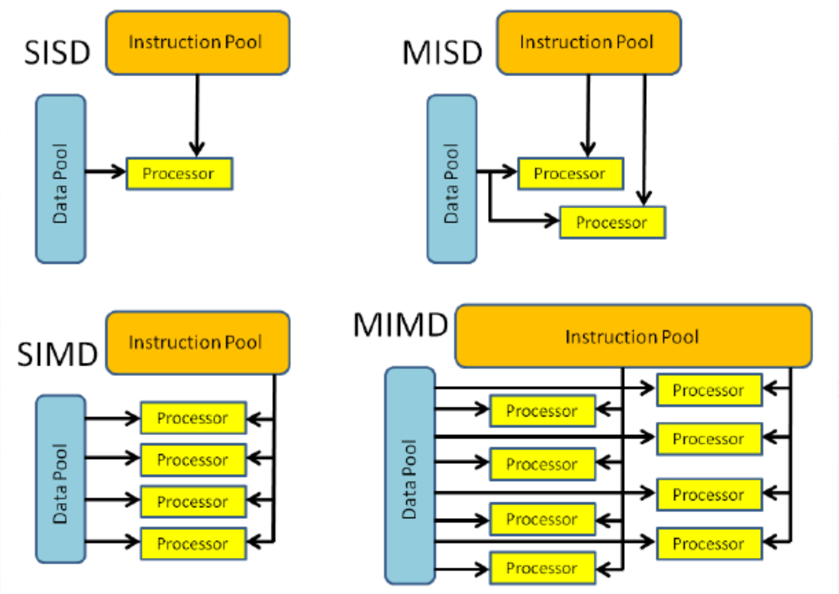
❖ [1]Lorenz Gillner, Ekaterina Auer., *Interval Methods for the GPU, SWIM, 2023.*
❖ [2]https://github.com/JuliaIntervals/IntervalArithmetic.jl

Acknowledgements

Thank you for your time!

Flynn’s Taxonomy

- SISD – Single Instruction Single Data
- MISD – Multiple Instructions Multiple Data
- **SIMD** – Single Instruction Multiple Data
- **MIMD** – Multiple Instructions Multiple Data

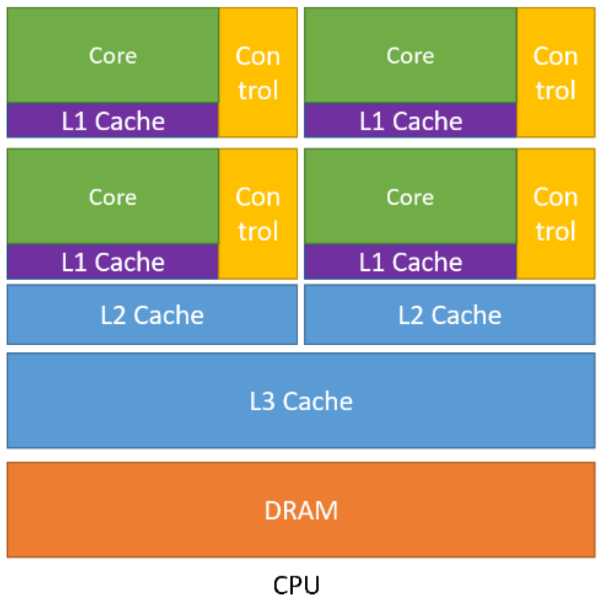


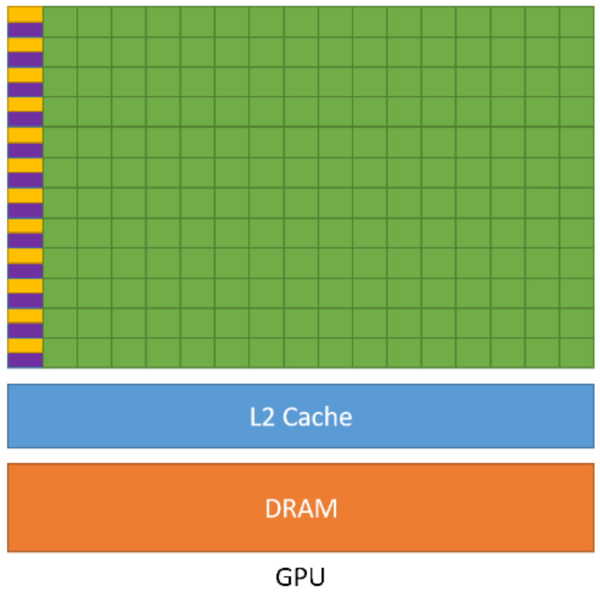
- Lorenz Gillner, Ekaterina Auer., *Interval Methods for the GPU, SWIM, 2023.*
- P. Pacheco and M. Malensek, *An introduction to parallel programming*. Morgan Kaufmann, 2021

Parallel Computing

- Task Parallelism
- Data Parallelism

- Fine-Grained
- Coarse-Grained





- Lorenz Gillner, Ekaterina Auer., *Interval Methods for the GPU, SWIM, 2023.*
- P. Pacheco and M. Malensek, *An introduction to parallel programming*. Morgan Kaufmann, 2021