ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING

CUDA-Based Implementation of GSLIB: The Geostatistical Software Library

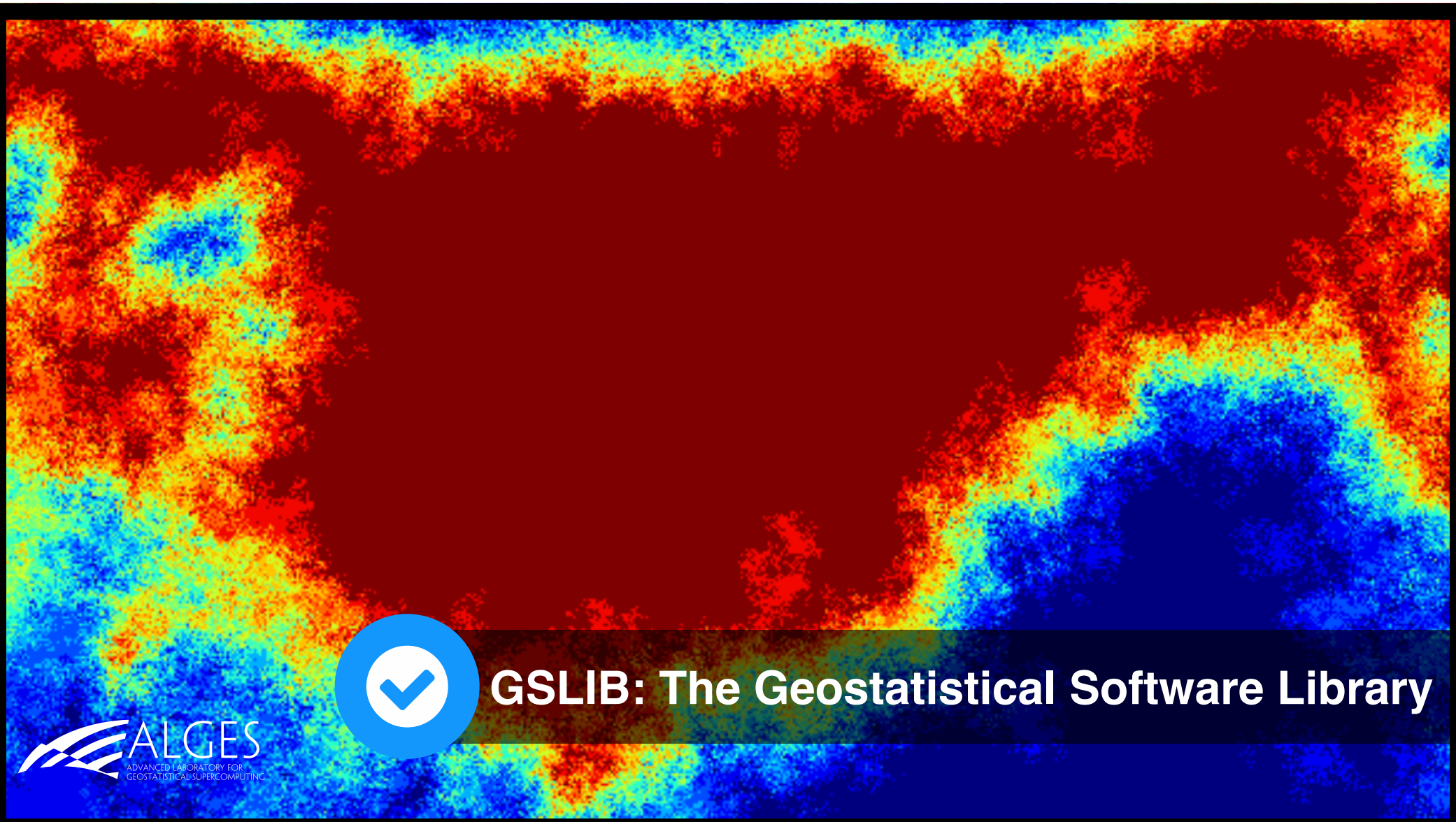
Daniel Baeza
dabaeza@alges.cl

Oscar Peredo
operedo@alges.cl

NVIDIA CUDA RESEARCH CENTER

amtc
ADVANCED MINING TECHNOLOGY CENTER


ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING
The Mining Process
Exploration
Evaluation
Planning
Operation


GSLIB: The Geostatistical Software Library
ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING

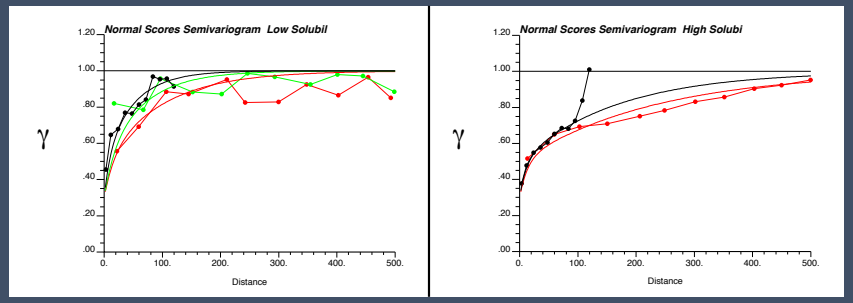
GSLIB: The Geostatistical Software Library

- GSLIB is a software package composed by a set of utilities and applications related with geostatistics
- Full implemented in Fortran 77/90
- Run in OSX, Linux and Windows
- Widely used for academics, researchers, engineers

GSLIB: Geostatistical Software Library and user's guide *(1998)*
Deutsch, Clayton V, Journel, André G

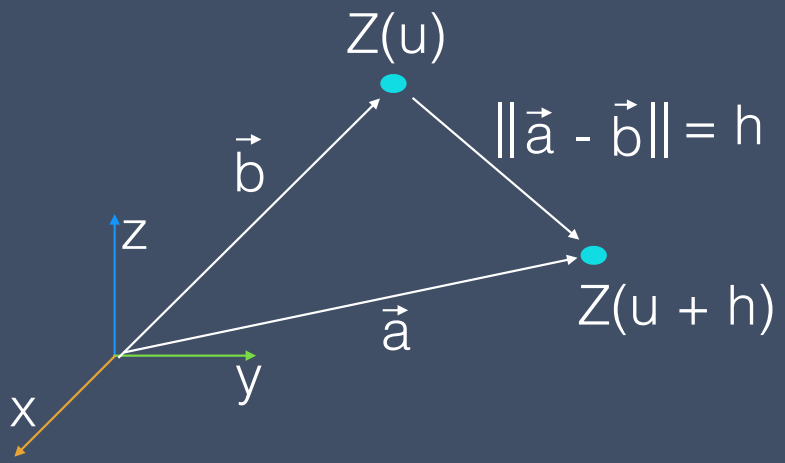
Variogram calculation with GSLIB

- **gamv** is the GSLIB variogram calculation method
- It's a fundamental tool in geostatistics
- Allow to quantify the spatial variability of a variable
- Used in geostatistical estimation and simulation
- High computational cost

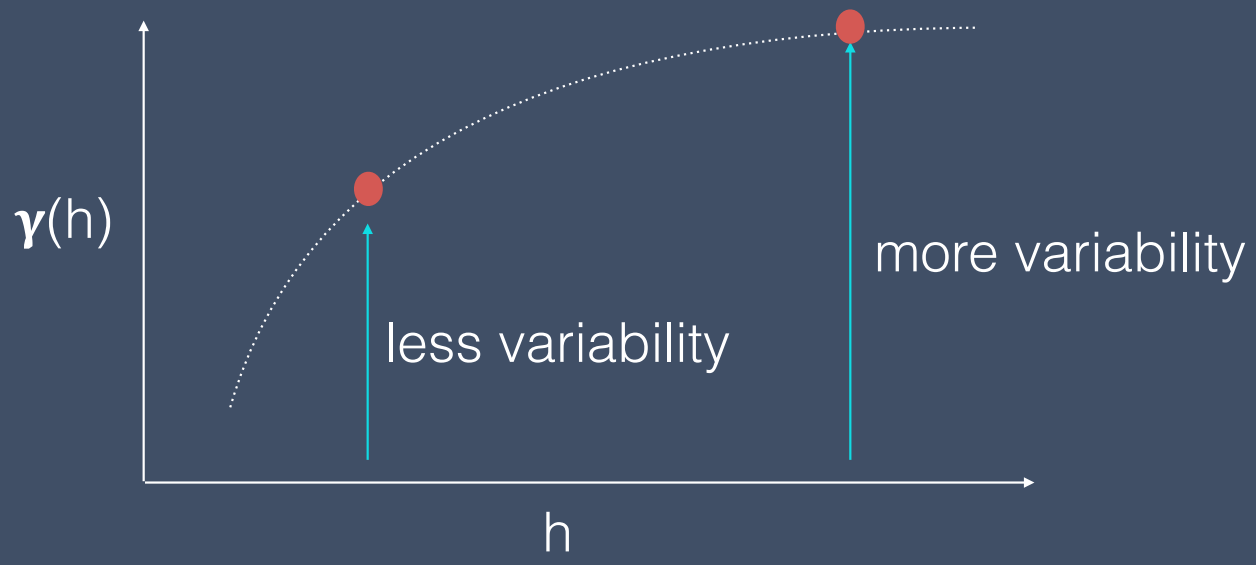


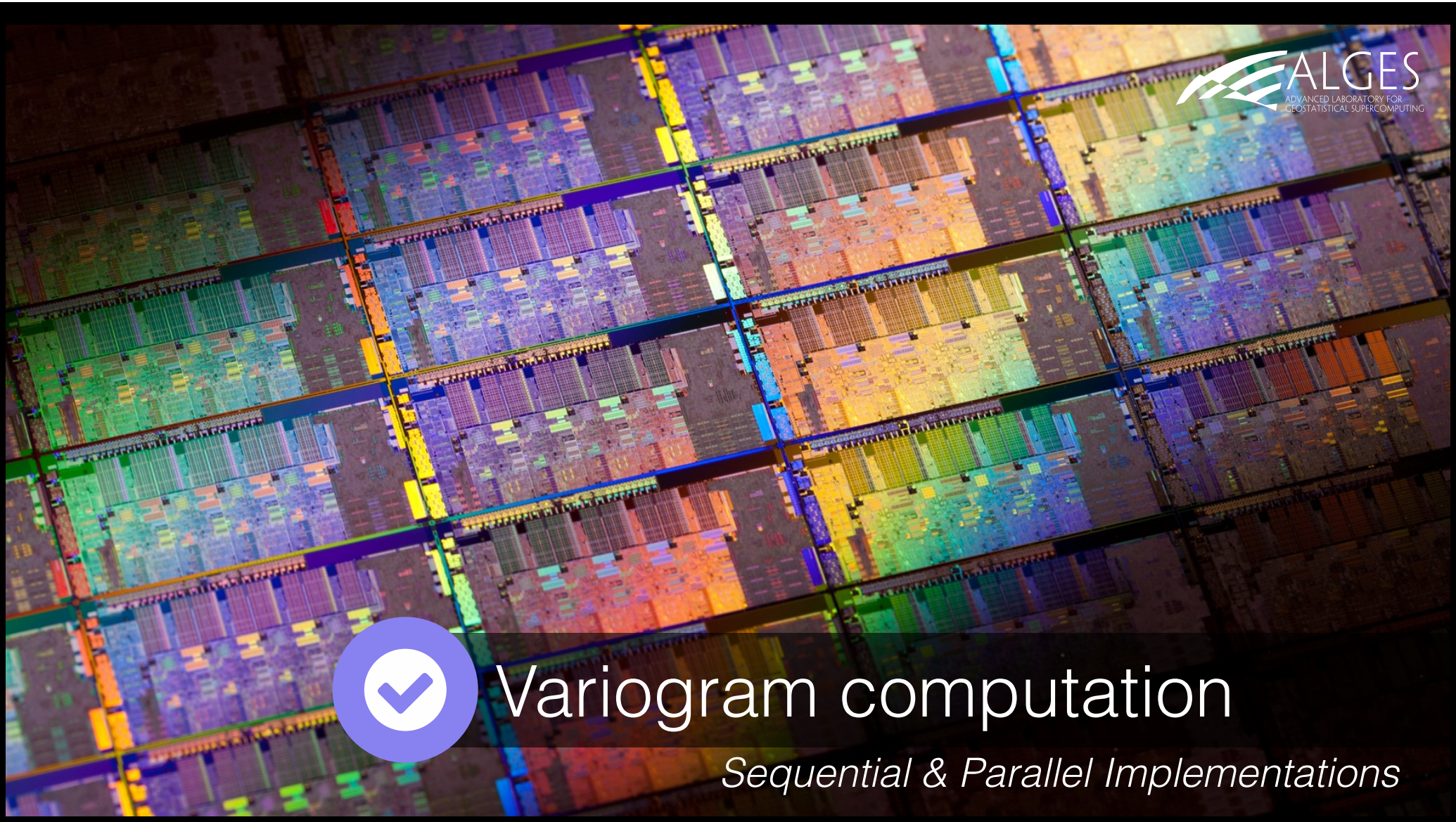
ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING

Variogram calculation



$$2\gamma(h) = \frac{1}{N(h)} \sum [z(u) - z(u + h)]^2$$





Variogram computation

Sequential & Parallel Implementations

Sequential implementation

Input:

- $(\mathbf{VR}, \Omega)$: sample data values $\mathbf{VR}$ (m columns) defined in a 3D domain of coordinates Ω
- $nvar$: number of variables ($nvar \leq m$)
- $nlag$: number of lags
- h: lag separation distance
- $ndir$: number of directions
- $\mathbf{h}_1, \ldots, \mathbf{h}_{ndir}$: directions
- $\tau_1, \ldots, \tau_{ndir}$: geometrical tolerance parameters
- $nvarg$: number of variograms
- $(ivtype_1, ivtail_1, ivhead_1), \ldots, (ivtype_{nvarg}, ivtail_{nvarg}, ivhead_{nvarg})$: variogram types

Setup parameters & Load data

```
Read input parameter file;
Read sample data values file;
β ← zeros(nvar × nlag × ndir × nvarg);
for i ∈ {1, ..., |Ω|} do
    for j ∈ {i, ..., |Ω|} do
        for id ∈ {1, ..., ndir} do
            for iv ∈ {1, ..., nvarg} do
                for il ∈ {1, ..., nlag} do
                    p_i = (x_i, y_i, z_i) ∈ Ω;
                    p_j = (x_j, y_j, z_j) ∈ Ω;
                    if (p_i, p_j) satisfy tolerances τ_id and ||p_i − p_j|| ≈ h_id × il × h then
                        Save (VR_{i,ivhead_iv} or VR_{i,ivtail_iv}) and (VR_{j,ivhead_iv} or VR_{j,ivtail_iv}) according to variogram ivtype_iv into β;
γ ← build variogram using statistics β Write γ in the output file
```

Main computation: Loop over pairs of points

Read computation results

Output: Output file with γ values

Sequential implementation

```
4  for i ∈ {1, ..., |Ω|} do
5      for j ∈ {i, ..., |Ω|} do
6          for id ∈ {1, ..., ndir} do
7              for iv ∈ {1, ..., nvarg} do
8                  for il ∈ {1, ..., nlag} do
9                      p_i = (x_i, y_i, z_i) ∈ Ω;
10                     p_j = (x_j, y_j, z_j) ∈ Ω;
11                     if (p_i, p_j) satisfy tolerances τ_id and ||p_i − p_j|| ≈ h_id × il × h then
12                         Save (VR_{i,ivhead_iv} or VR_{i,ivtail_iv}) and (VR_{j,ivhead_iv} or VR_{j,ivtail_iv}) according to variogram ivtype_iv into β;
```

Sequential implementation

```
4  for i ∈ {1, ..., |Ω|} do
5      for j ∈ {i, ..., |Ω|} do
6          for id ∈ {1, ..., ndir} do
7              for iv ∈ {1, ..., nvarg} do
8                  for il ∈ {1, ..., nlag} do
9                      p_i = (x_i, y_i, z_i) ∈ Ω;
10                     p_j = (x_j, y_j, z_j) ∈ Ω;
11                     if (p_i, p_j) satisfy tolerances τ_id and ||p_i − p_j|| ≈ h_id × il × h then
12                         Save (VR_{i,ivhead_iv} or VR_{i,ivtail_iv}) and (VR_{j,ivhead_iv} or VR_{j,ivtail_iv}) according to variogram ivtype_iv into β;
```

Sequential implementation

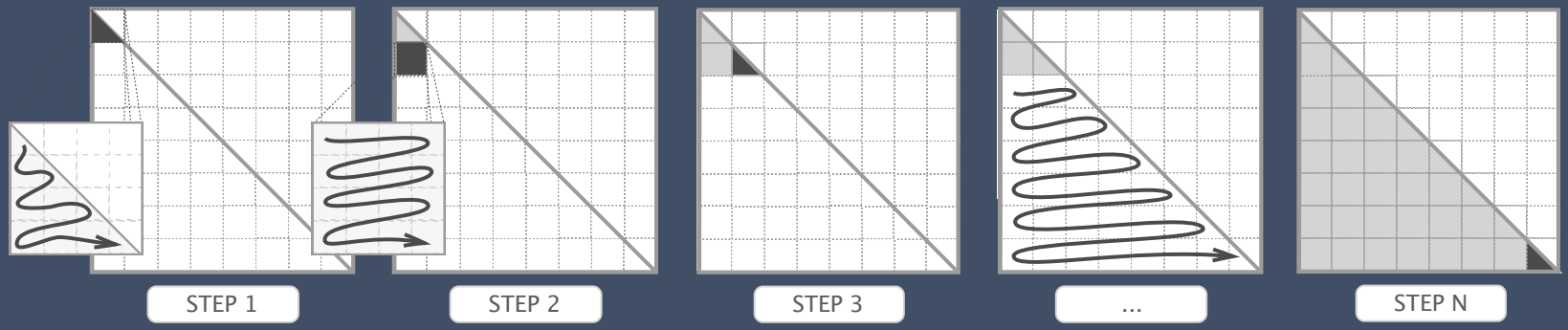
```
4  for i ∈ {1, ..., |Ω|} do
5      for j ∈ {i, ..., |Ω|} do
6          for id ∈ {1, ..., ndir} do
7              for iv ∈ {1, ..., nvarg} do
8                  for il ∈ {1, ..., nlag} do
9                      p_i = (x_i, y_i, z_i) ∈ Ω;
10                     p_j = (x_j, y_j, z_j) ∈ Ω;
11                     if (p_i, p_j) satisfy tolerances τ_id and ||p_i − p_j|| ≈ h_id × il × h then
12                         Save (VR_{i,ivhead_iv} or VR_{i,ivtail_iv}) and (VR_{j,ivhead_iv} or VR_{j,ivtail_iv}) according to variogram ivtype_iv into β;
```

COMPUTE_STATISTICS(p_i, p_j)

Sequential implementation

```
4  for i ∈ {1, ..., |Ω|} do
5      for j ∈ {i, ..., |Ω|} do
6          for id ∈ {1, ..., ndir} do
7              for iv ∈ {1, ..., nvarg} do
8                  for il ∈ {1, ..., nlag} do
9                      p_i = (x_i, y_i, z_i) ∈ Ω;
10                     p_j = (x_j, y_j, z_j) ∈ Ω;
11                     if (p_i, p_j) satisfy tolerances τ_id and ||p_i − p_j|| ≈ h_id × il × h then
12                         Save (VR_{i,ivhead_iv} or VR_{i,ivtail_iv}) and (VR_{j,ivhead_iv} or VR_{j,ivtail_iv}) according to variogram ivtype_iv into β;
```

COMPUTE_STATISTICS(p_i, p_j)



Parallel implementation

GPU Kernel

```
1  id_x = blockId.x*blockDim.x + threadId.x                /* x threads coord in the GPU grid */
2  id_y = blockId.y*blockDim.y + threadId.y                /* y threads coord in the GPU grid */
3  Set and Initialize shared memory in the block
4  __syncthreads()
5  iter_x = id_x
6  iter_y = id_y
7  while (iter_x & iter_y ∈ ChunkPoints)                   /* Chunk points belongs thread (x,y) */
8  do
9      j = iter_x + |Ω|/2
10     i = iter_y
11     COMPUTE_STATISTICS(p_i, p_j)  <--
12     store result in shared memory via atomic functions
13     if (iter_x > iter_y) then
14         i = iter_y
15         j = iter_x
16         COMPUTE_STATISTICS(p_i, p_j)  <--
17         store result in shared memory via atomic functions
18     else
19         if (iter_x == iter_y) then
20             i = iter_y
21             j = iter_y
22             COMPUTE_STATISTICS(p_i, p_j)  <--
23             store result in shared memory via atomic functions
24         i = iter_x + |Ω|/2
25         j = iter_y + |Ω|/2
26         COMPUTE_STATISTICS(p_i, p_j)  <--
27         store result in shared memory via atomic functions
28     UP_DATE(iter_x, iter_y)
29 __syncthreads()
30 save Statistics values that are in shared memory into globa memory via atomic functions
   Output: Array β
```

Each thread compute
only a couple of correlation values

Parallel implementation

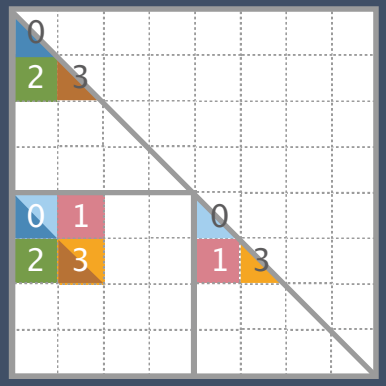
```
1  id_x = blockId.x*blockDim.x + threadId.x                    /* x threads coord in the GPU grid */
2  id_y = blockId.y*blockDim.y + threadId.y                    /* y threads coord in the GPU grid */
3  Set and Initialize shared memory in the block
4  __syncthreads()
5  iter_x = id_x
6  iter_y = id_y
7  while (iter_x & iter_y ∈ ChunkPoints)                       /* Chunk points belongs thread (x,y) */
8  do
9      j = iter_x + |Ω|/2
10     i = iter_y
11     COMPUTE_STATISTICS(p_i, p_j)
12     store result in shared memory via atomic functions
13     if (iter_x > iter_y) then
14         i = iter_y
15         j = iter_x
16         COMPUTE_STATISTICS(p_i, p_j)
17         store result in shared memory via atomic functions
18     else
19         if (iter_x == iter_y) then
20             i = iter_y
21             j = iter_y
22             COMPUTE_STATISTICS(p_i, p_j)
23             store result in shared memory via atomic functions
24         i = iter_x + |Ω|/2
25         j = iter_y + |Ω|/2
26         COMPUTE_STATISTICS(p_i, p_j)
27         store result in shared memory via atomic functions
28     UP_DATE(iter_x, iter_y)
29 __syncthreads()
30 save Statistics values that are in shared memory into globa memory via atomic functions
```

Output: Array β

Parallel implementation

```
1  id_x = blockId.x*blockDim.x + threadId.x                    /* x threads coord in the GPU grid */
2  id_y = blockId.y*blockDim.y + threadId.y                    /* y threads coord in the GPU grid */
3  Set and Initialize shared memory in the block
4  __syncthreads()
5  iter_x = id_x
6  iter_y = id_y
7  while (iter_x & iter_y ∈ ChunkPoints)                       /* Chunk points belongs thread (x,y) */
8  do
9      j = iter_x + |Ω|/2
10     i = iter_y
11     COMPUTE_STATISTICS(p_i, p_j)
12     store result in shared memory via atomic functions
13     if (iter_x > iter_y) then
14         i = iter_y
15         j = iter_x
16         COMPUTE_STATISTICS(p_i, p_j)
17         store result in shared memory via atomic functions
18     else
19         if (iter_x == iter_y) then
20             i = iter_y
21             j = iter_y
22             COMPUTE_STATISTICS(p_i, p_j)
23             store result in shared memory via atomic functions
24         i = iter_x + |Ω|/2
25         j = iter_y + |Ω|/2
26         COMPUTE_STATISTICS(p_i, p_j)
27         store result in shared memory via atomic functions
28     UP_DATE(iter_x, iter_y)
29 __syncthreads()
30 save Statistics values that are in shared memory into globa memory via atomic functions
```

Output: Array β

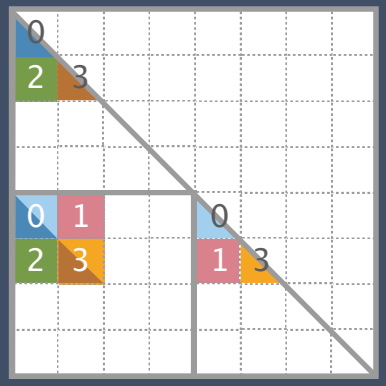


STEP 1

Parallel implementation

```
1  id_x = blockId.x*blockDim.x + threadId.x                    /* x threads coord in the GPU grid */
2  id_y = blockId.y*blockDim.y + threadId.y                    /* y threads coord in the GPU grid */
3  Set and Initialize shared memory in the block
4  __syncthreads()
5  iter_x = id_x
6  iter_y = id_y
7  while (iter_x & iter_y ∈ ChunkPoints)                       /* Chunk points belongs thread (x,y) */
8  do
9      j = iter_x + |Ω|/2
10     i = iter_y
11     COMPUTE_STATISTICS(p_i, p_j)
12     store result in shared memory via atomic functions
13     if (iter_x > iter_y) then
14         i = iter_y
15         j = iter_x
16         COMPUTE_STATISTICS(p_i, p_j)
17         store result in shared memory via atomic functions
18     else
19         if (iter_x == iter_y) then
20             i = iter_y
21             j = iter_y
22             COMPUTE_STATISTICS(p_i, p_j)
23             store result in shared memory via atomic functions
24         i = iter_x + |Ω|/2
25         j = iter_y + |Ω|/2
26         COMPUTE_STATISTICS(p_i, p_j)
27         store result in shared memory via atomic functions
28     UP_DATE(iter_x, iter_y)
29 __syncthreads()
30 save Statistics values that are in shared memory into globa memory via atomic functions
```

Output: Array β

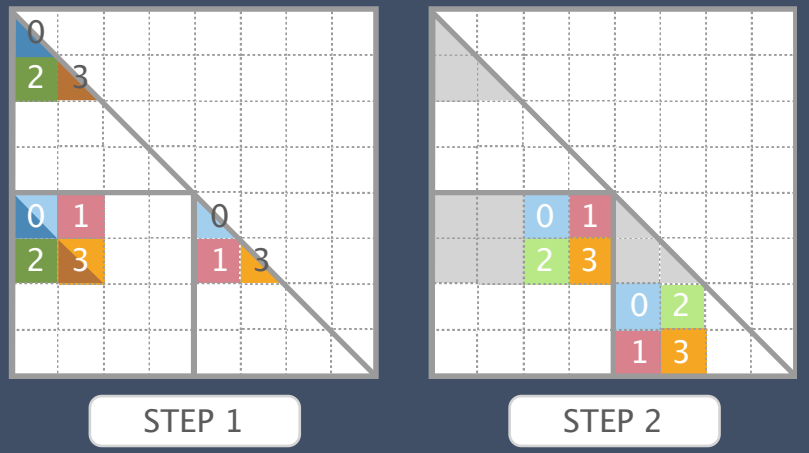


STEP 1

Parallel implementation

```
1  id_x = blockId.x*blockDim.x + threadId.x                    /* x threads coord in the GPU grid */
2  id_y = blockId.y*blockDim.y + threadId.y                    /* y threads coord in the GPU grid */
3  Set and Initialize shared memory in the block
4  __syncthreads()
5  iter_x = id_x
6  iter_y = id_y
7  while (iter_x & iter_y ∈ ChunkPoints)                       /* Chunk points belongs thread (x,y) */
8  do
9      j = iter_x + |Ω|/2
10     i = iter_y
11     COMPUTE_STATISTICS(p_i, p_j)
12     store result in shared memory via atomic functions
13     if (iter_x > iter_y) then
14         i = iter_y
15         j = iter_x
16         COMPUTE_STATISTICS(p_i, p_j)
17         store result in shared memory via atomic functions
18     else
19         if (iter_x == iter_y) then
20             i = iter_y
21             j = iter_y
22             COMPUTE_STATISTICS(p_i, p_j)
23             store result in shared memory via atomic functions
24         i = iter_x + |Ω|/2
25         j = iter_y + |Ω|/2
26         COMPUTE_STATISTICS(p_i, p_j)
27         store result in shared memory via atomic functions
28     UP_DATE(iter_x, iter_y)
29 __syncthreads()
30 save Statistics values that are in shared memory into globa memory via atomic functions
```

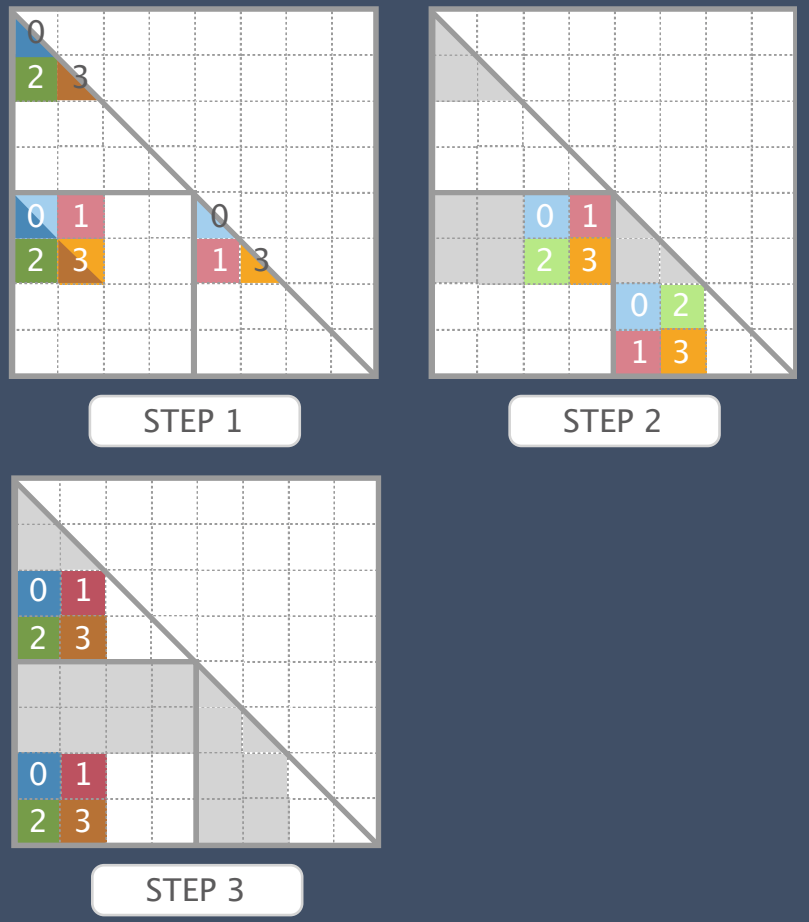
Output: Array β



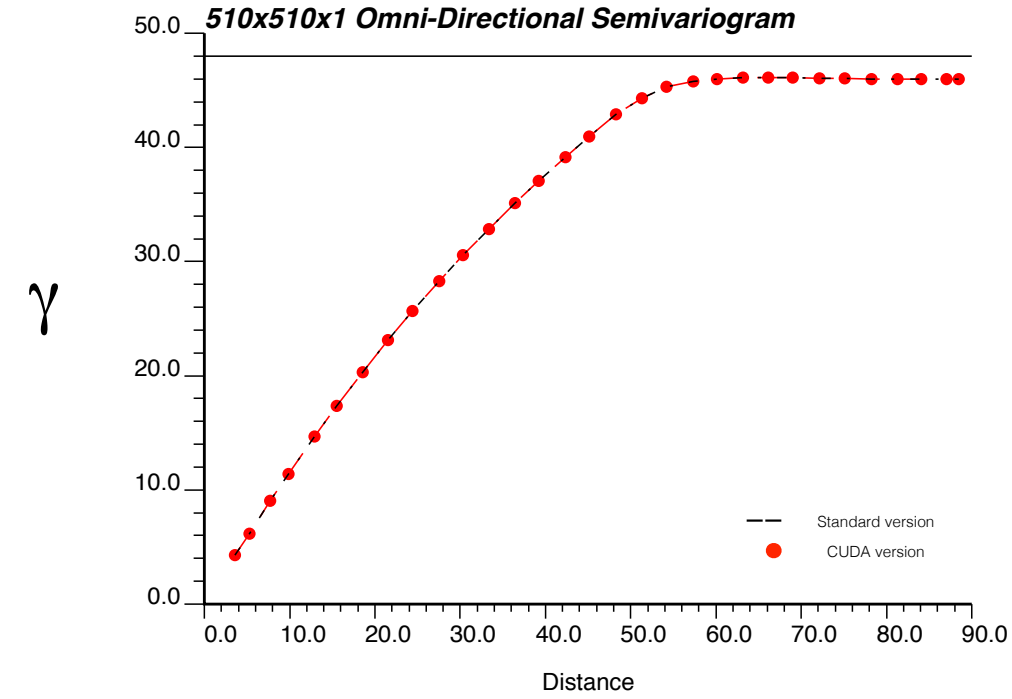
Parallel implementation

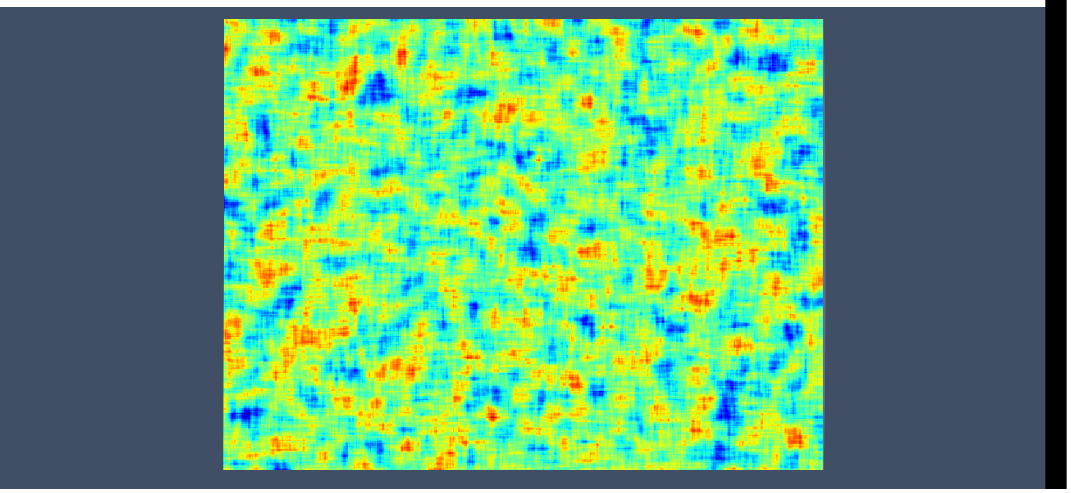
```
1  id_x = blockId.x*blockDim.x + threadId.x                    /* x threads coord in the GPU grid */
2  id_y = blockId.y*blockDim.y + threadId.y                    /* y threads coord in the GPU grid */
3  Set and Initialize shared memory in the block
4  __syncthreads()
5  iter_x = id_x
6  iter_y = id_y
7  while (iter_x & iter_y ∈ ChunkPoints)                       /* Chunk points belongs thread (x,y) */
8  do
9      j = iter_x + |Ω|/2
10     i = iter_y
11     COMPUTE_STATISTICS(p_i, p_j)
12     store result in shared memory via atomic functions
13     if (iter_x > iter_y) then
14         i = iter_y
15         j = iter_x
16         COMPUTE_STATISTICS(p_i, p_j)
17         store result in shared memory via atomic functions
18     else
19         if (iter_x == iter_y) then
20             i = iter_y
21             j = iter_y
22             COMPUTE_STATISTICS(p_i, p_j)
23             store result in shared memory via atomic functions
24         i = iter_x + |Ω|/2
25         j = iter_y + |Ω|/2
26         COMPUTE_STATISTICS(p_i, p_j)
27         store result in shared memory via atomic functions
28     UP_DATE(iter_x, iter_y)
29 __syncthreads()
30 save Statistics values that are in shared memory into globa memory via atomic functions
```

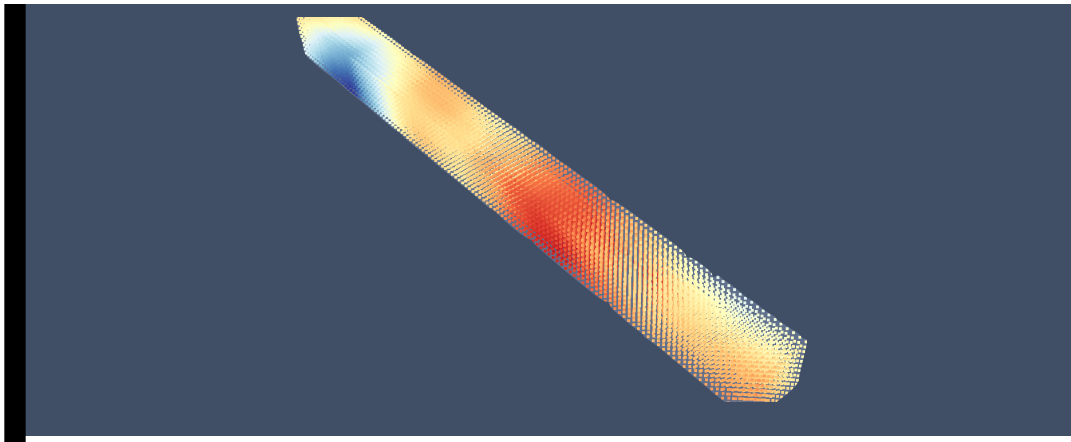
Output: Array β

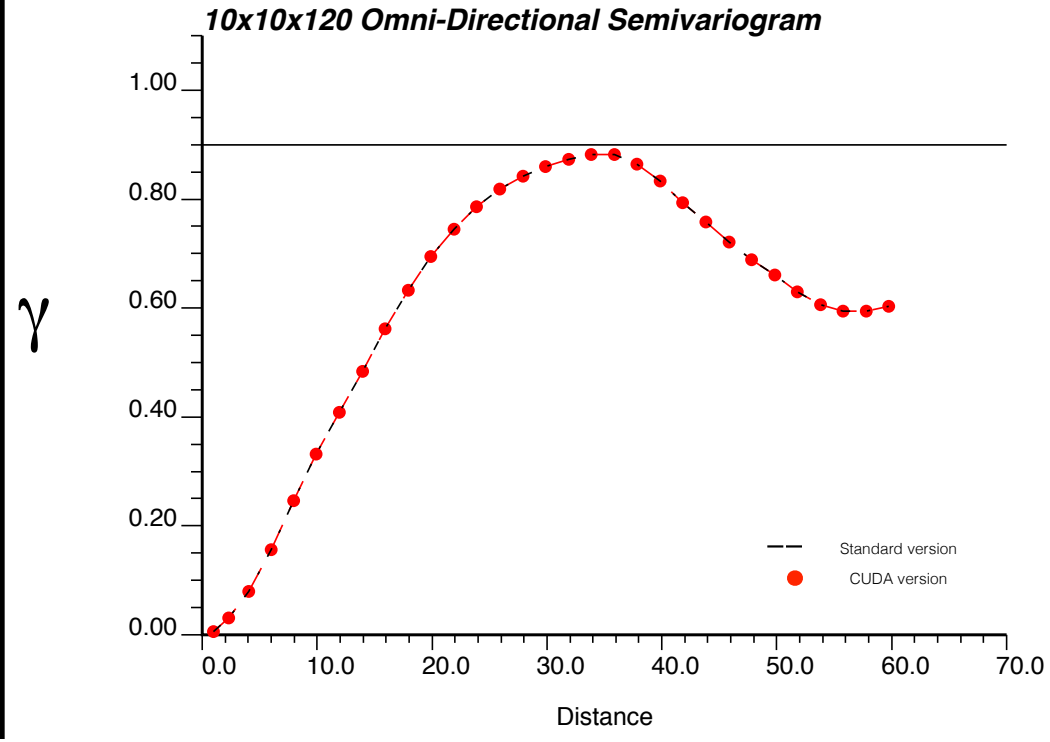



Results
ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING


510x510x1 Omni-Directional Semivariogram
γ
50.0
40.0
30.0
20.0
10.0
0.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
Distance
Standard version
CUDA version

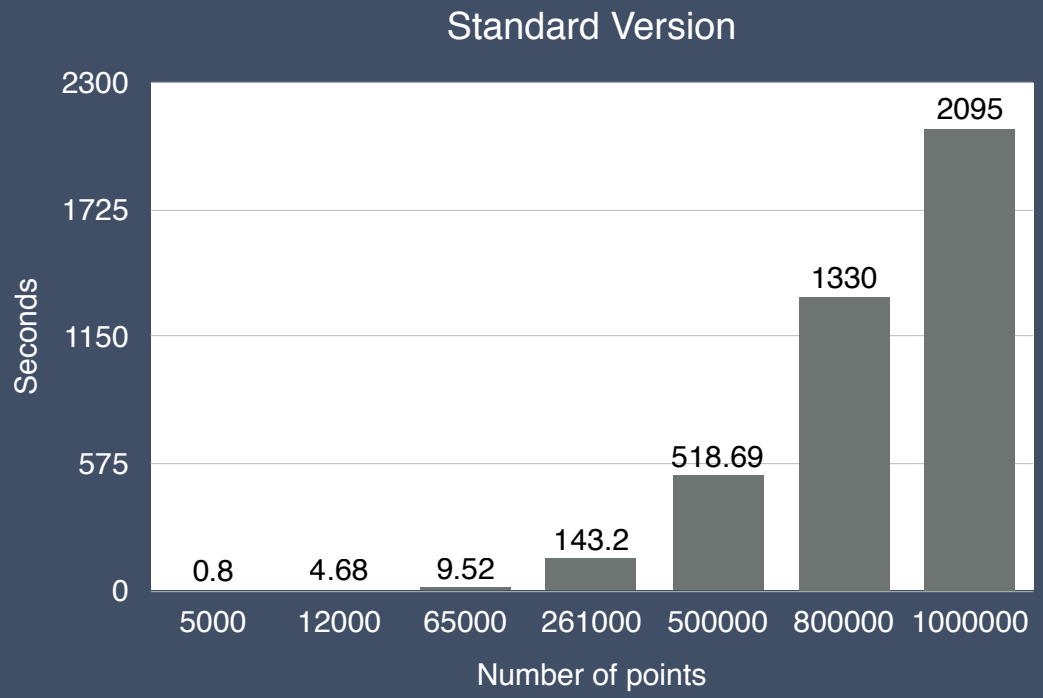





10x10x120 Omni-Directional Semivariogram
γ
1.00
0.80
0.60
0.40
0.20
0.00
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
Distance
Standard version
CUDA version

Dell Precision T7600

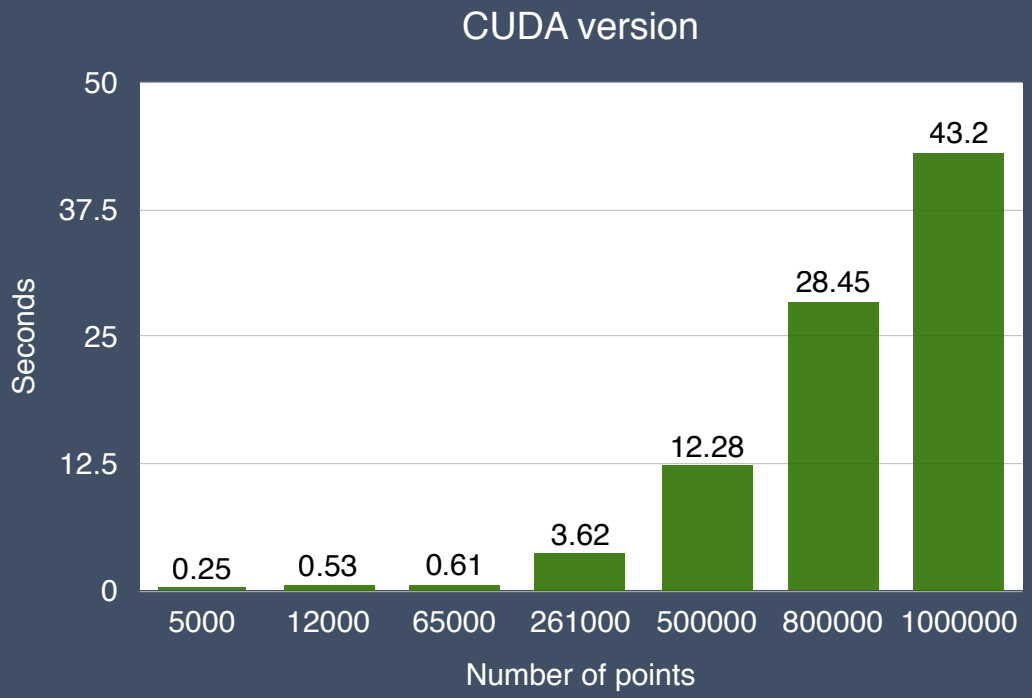
- Intel Xeon E5
- 3.3 GHz CPU Frequency
- 10 MB cache
- 16GB RAM (1,6 MHz)
- 1 TB HDD
- Linux



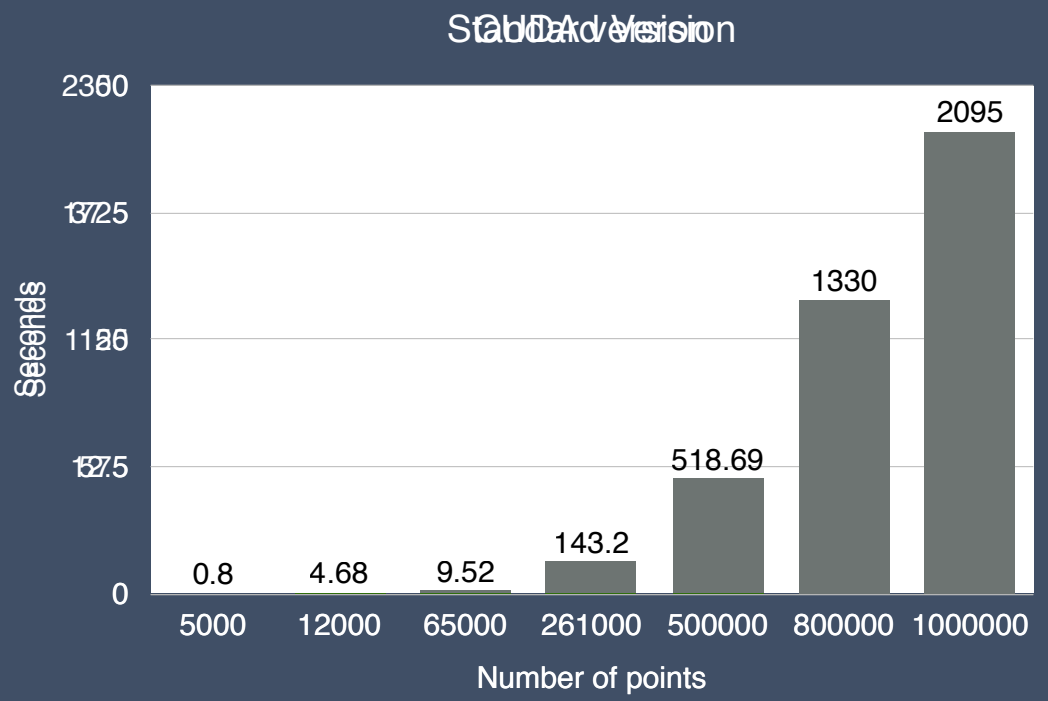
ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING

Tesla c2075

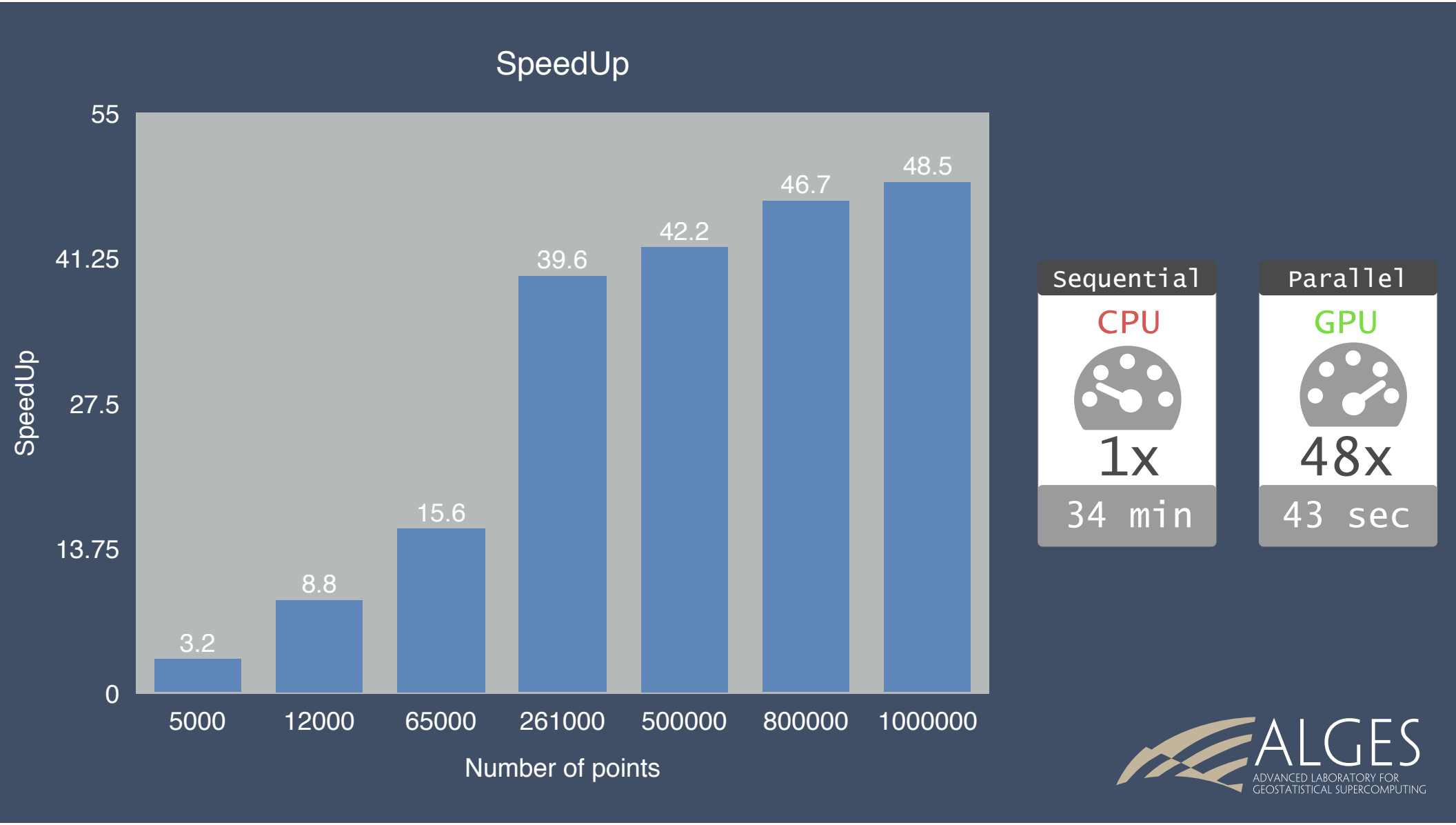
- 448 CUDA cores
- 1.15 GHz Frequency of CUDA cores
- 6GB RAM
- 114 GB/sec Memory bandwidth

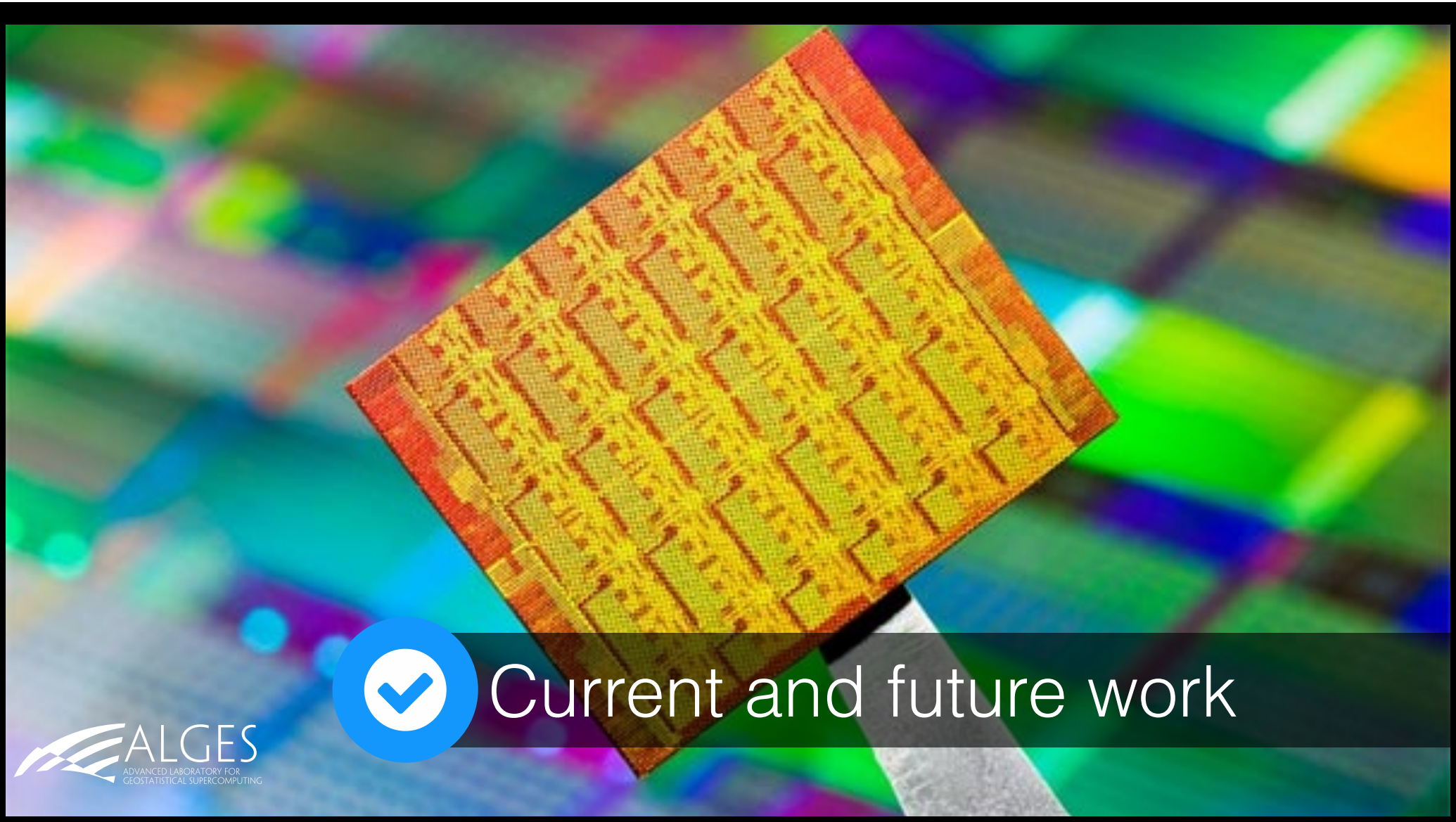


ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING

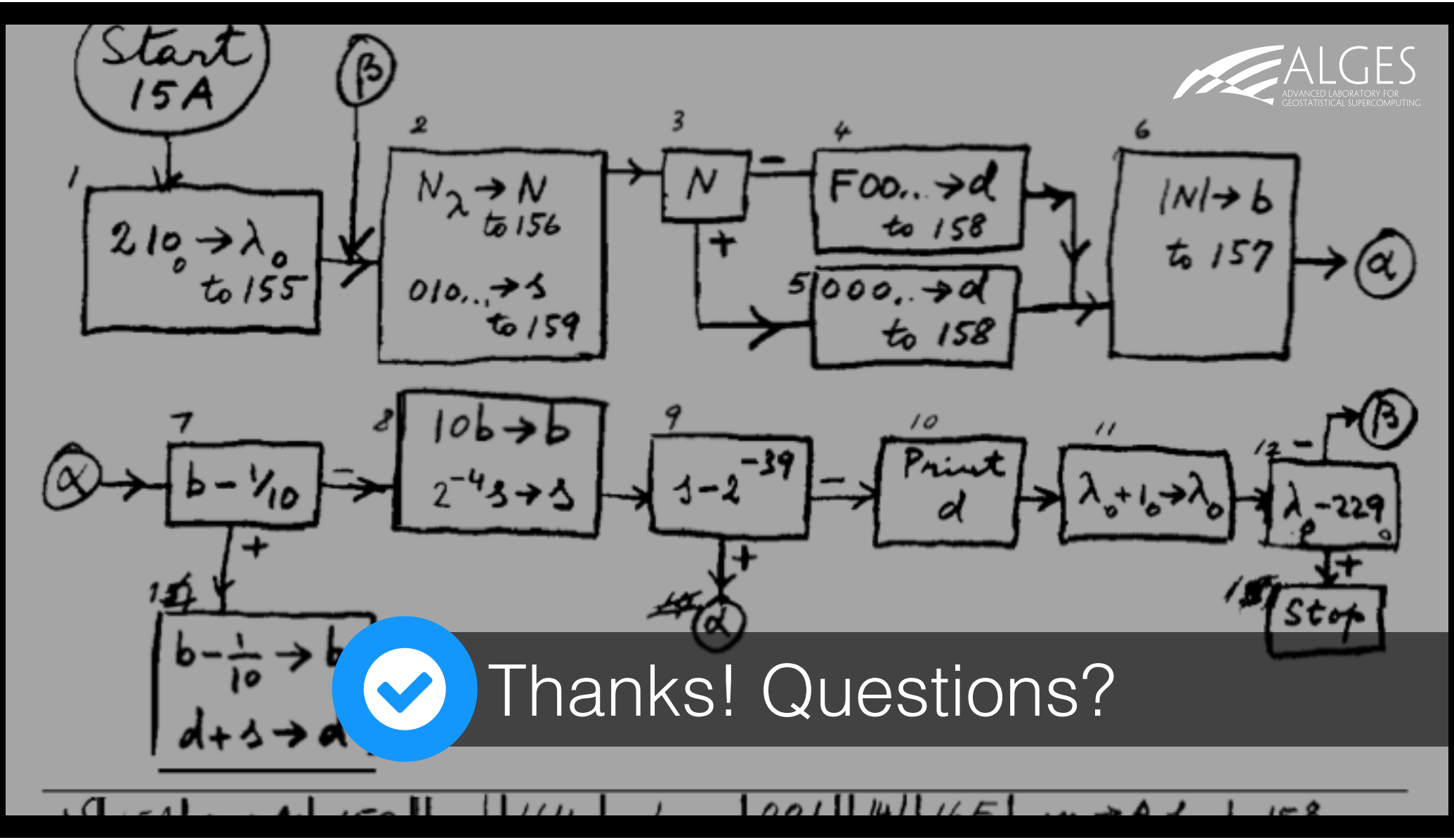


ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING


SpeedUp
55
41.25
27.5
13.75
0
SpeedUp
3.2
8.8
15.6
39.6
42.2
46.7
48.5
5000
12000
65000
261000
500000
800000
1000000
Number of points
Sequential
CPU
1x
34 min
Parallel
GPU
48x
43 sec
ALGES
ADVANCED LABORATORY FOR GEOSTATISTICAL SUPERCOMPUTING


Current and future work
ALGES
ADVANCED LABORATORY FOR
GEOSTATISTICAL SUPERCOMPUTING

- Finish the CUDA version of indicator simulation and gaussian simulation of GSLIB
- Release the first GSLIB-CUDA version with these three methods


Start
15A
β
1
2
3
4
5
6
α
7
8
9
10
11
12
13
14
15
Stop
Thanks! Questions?