

CUDA-Based Implementation of GSLIB: The Geostatistical Software Library

Daniel Baeza

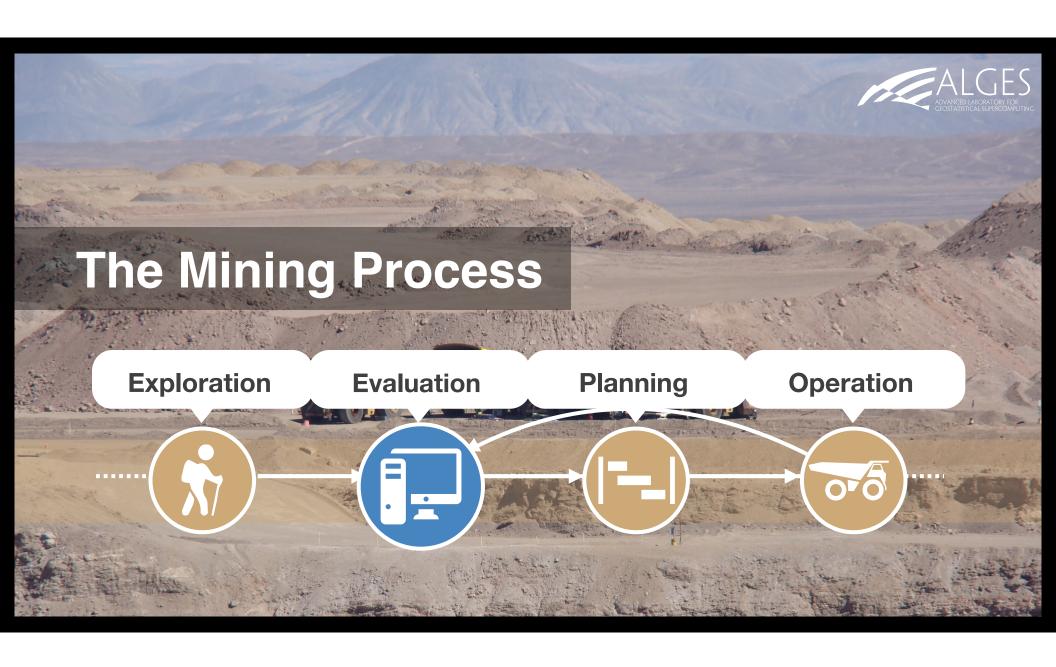
dabaeza@alges.cl

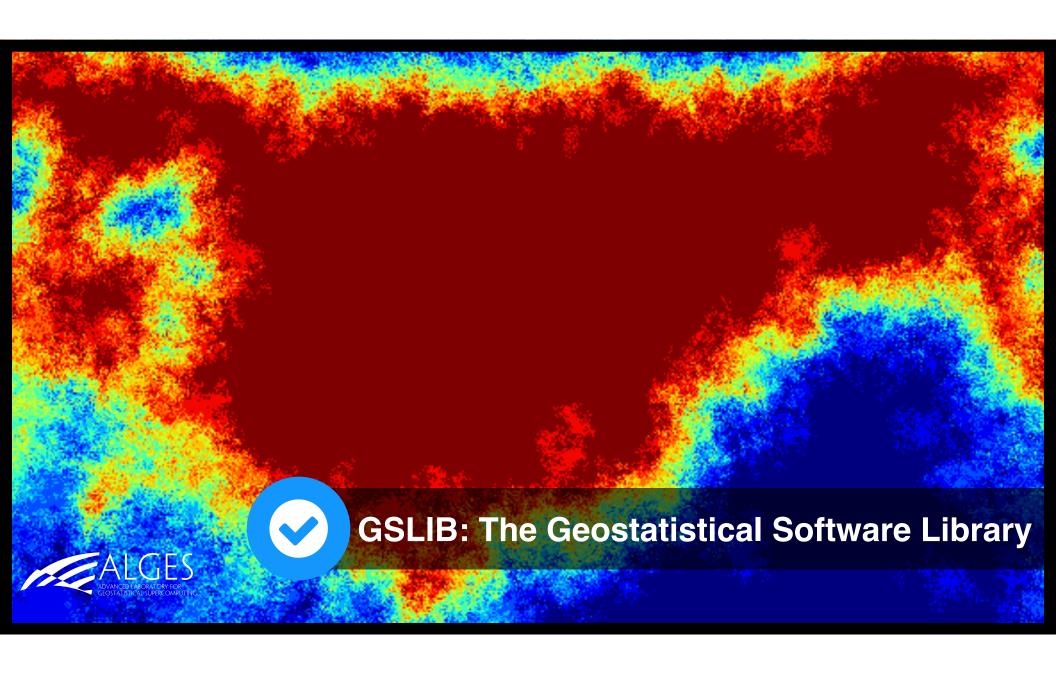
Oscar Peredo

operedo@alges.cl









GSLIB: The Geostatistical Software Library

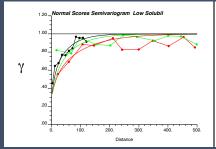
 GSLIB is a software package composed by a set of utilities and applications related with geostatistics

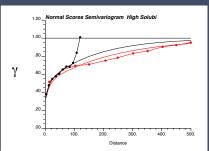
- Full implemented in Fortran 77/90
- Run in OSX, Linux and Windows
- Widely used for academics, researchers, engineers

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

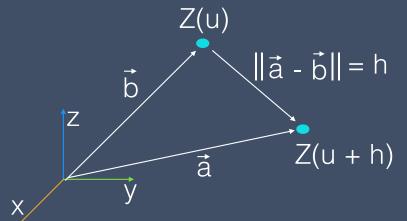
Variogram calculation with GSLIB

- 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

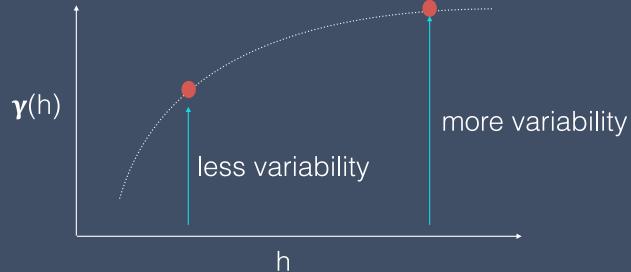


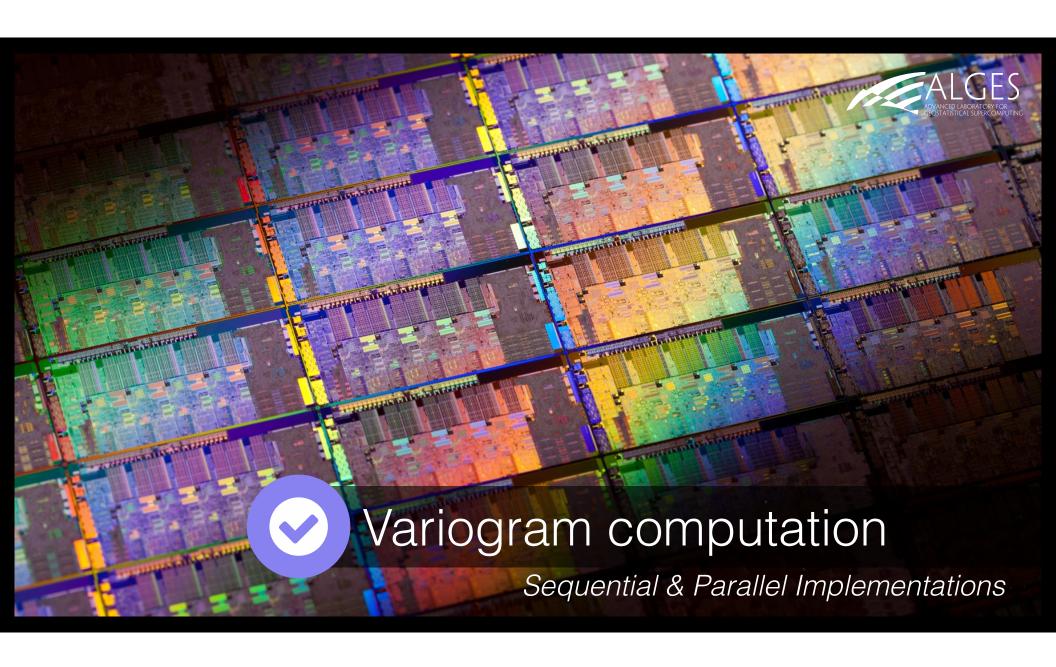


Variogram calculation



$$2\mathbf{\gamma}(h) = \mathbf{1} \sum_{\mathsf{N}(h)} \left[z(\mathsf{u}) - z(\mathsf{u} + \mathsf{h}) \right]^2$$





```
Input:
```

 $\gamma \leftarrow$ build variogram using statistics β Write γ in the output file

Output: Output file with γ values

• (VR, Ω) : sample data values VR (m columns) defined in a 3D domain of coordinates Ω Setup parameters • *nvar*: number of variables $(nvar \le m)$ • nlag: number of lags • h: lag separation distance I oad data • ndir: number of directions • $\mathbf{h}_1, \dots, \mathbf{h}_{ndir}$: directions • $\tau_1, \ldots, \tau_{ndir}$: geometrical tolerance parameters • *nvarg*: number of variograms • $(ivtype_1, ivtail_1, ivhead_1), \dots, (ivtype_{nvarg}, ivtail_{nvarg}, ivhead_{nvarg})$: variogram types Read input parameter file; Read sample data values file; $\beta \leftarrow zeros(nvar \times nlag \times ndir \times nvarg);$ for $i \in \{1, \ldots, |\Omega|\}$ do Main computation: Loop over pairs of points for $j \in \{i, \ldots, |\Omega|\}$ do for $id \in \{1, \dots, ndir\}$ do for $iv \in \{1, \dots, nvarg\}$ do for $il \in \{1, \dots, nlag\}$ do $p_i = (x_i, y_i, z_i) \in \Omega;$ $p_i = (x_i, y_i, z_i) \in \Omega;$ **if** (p_i, p_j) satisfy tolerances τ_{id} and $||p_i - p_j|| \approx \mathbf{h}_{id} \times il \times h$ **then** Save $(\mathbf{V}\mathbf{R}_{i,ivheadiv})$ or $\mathbf{V}\mathbf{R}_{i,ivheadiv}$ and $(\mathbf{V}\mathbf{R}_{i,ivheadiv})$ or $\mathbf{V}\mathbf{R}_{i,ivtailiv}$) according to variogram $ivtype_{iv}$ into β ; Read computation results

```
4 for i \in \{1, \dots, |\Omega|\} do

5 | for j \in \{i, \dots, |\Omega|\} do

6 | for id \in \{1, \dots, ndir\} do

7 | for iv \in \{1, \dots, nvarg\} do

8 | for il \in \{1, \dots, nlag\} do

9 | p_i = (x_i, y_i, z_i) \in \Omega;

10 | p_i = (x_i, y_j, z_j) \in \Omega;

11 | if (p_i, p_j) satisfy tolerances \tau_{id} and ||p_i - p_j|| \approx \mathbf{h}_{id} \times il \times h then

12 | Save (\mathbf{VR}_{i,ivhead_{iv}}) or \mathbf{VR}_{i,ivtail_{iv}} and (\mathbf{VR}_{j,ivhead_{iv}}) or \mathbf{VR}_{j,ivtail_{iv}}) according to variogram ivtype_{iv} into \beta;
```

```
4 for i \in \{1, \dots, |\Omega|\} do

5 | for j \in \{i, \dots, |\Omega|\} do

6 | for id \in \{1, \dots, ndir\} do

7 | for iv \in \{1, \dots, nvarg\} do

8 | for il \in \{1, \dots, nlag\} do

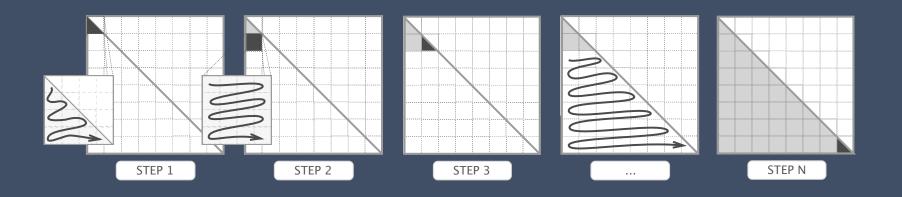
9 | p_i = (x_i, y_i, z_i) \in \Omega;

10 | p_j = (x_j, y_j, z_j) \in \Omega;

11 | p_j = (x_j, y_j, z_j) \in \Omega;

12 | if (p_i, p_j) satisfy tolerances \tau_{id} and ||p_i - p_j|| \approx \mathbf{h}_{id} \times il \times h then

12 | Save (\mathbf{VR}_{i,ivhead_{iv}}) or \mathbf{VR}_{i,ivtail_{iv}} and (\mathbf{VR}_{j,ivhead_{iv}}) or \mathbf{VR}_{j,ivtail_{iv}}) according to variogram ivtype_{iv} into \beta;
```



/* x threads coord in the GPU grid */

```
id_v = 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_v = id_v
7 while (iter, & iter, \in ChunkPoints)
                                                                /* Chunk points belongs thread (x,y) */
      COMPUTE_STATISTICS(p_i, p_j)
11
       store result in shared memory via atomic functions
12
      if (iter_x > iter_y) then
13
14
          i = iter_{v}
15
          j = iter_x
16
          COMPUTE_STATISTICS(p_i, p_j)
17
          store result in shared memory via atomic functions
      else
18
          if (iter_x == iter_y) then
19
              i = iter_v
21
              j = iter_{v}
              COMPUTE_STATISTICS(p_i, p_j)
              store result in shared memory via atomic functions
23
          i = iter_x + \frac{|\Omega|}{2}
          j = iter_v + \frac{|\Omega|}{2}
          COMPUTE_STATISTICS(p_i, p_j)
          store result in shared memory via atomic functions
       UP_DATE(iter_x, iter_y)
30 save Statistics values that are in shared memory into globa memory via atomic functions
   Output: Array \beta
```

1 id_x = blockId.x*blockDim.x + threadId.x

GPU Kernel

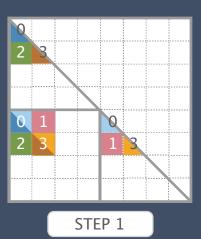
Each thread compute only a couple of correlation values

/* x threads coord in the GPU grid */

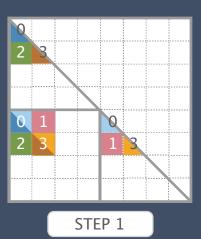
```
id_v = blockId.y*blockDim.y + threadId.y
                                                                       /* y threads coord in the GPU grid */
3 Set and Initialize shared memory in the block
4 __syncthreads()
5 iter_x = id_x
 6 iter_v = id_v
7 while (iter_x \& iter_y \in ChunkPoints)
                                                                    /* Chunk points belongs thread (x,y) */
       j = iter_x + \frac{|\Omega|}{2}
       COMPUTE_STATISTICS(p_i, p_j)
11
       store result in shared memory via atomic functions
12
       if (iter_x > iter_y) then
13
14
           i = iter_v
15
           j = iter_x
           COMPUTE_STATISTICS(p_i, p_j)
16
           store result in shared memory via atomic functions
17
       else
18
           if (iter_x == iter_y) then
19
               i = iter_v
               j = iter_v
21
               COMPUTE_STATISTICS(p_i, p_i)
               store result in shared memory via atomic functions
23
           i = iter_x + \frac{|\Omega|}{2}
           j = iter_y + \frac{|\overline{\Omega}|}{2}
           COMPUTE_STATISTICS(p_i, p_j)
           store result in shared memory via atomic functions
       UP_DATE(iter_x, iter_y)
30 save Statistics values that are in shared memory into globa memory via atomic functions
   Output: Array \beta
```

 $id_x = blockId.x*blockDim.x + threadId.x$

```
1 id_x = blockId.x*blockDim.x + threadId.x
                                                                      /* x threads coord in the GPU grid */
id_v = 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_v = id_v
7 while (iter_x \& iter_y \in ChunkPoints)
                                                                   /* Chunk points belongs thread (x,y) */
 8 do
       j = iter_x + \frac{|\Omega|}{2}
       COMPUTE_STATISTICS(p_i, p_j)
11
       store result in shared memory via atomic functions
12
       if (iter_x > iter_y) then
13
14
           i = iter_v
15
           j = iter_x
           COMPUTE_STATISTICS(p_i, p_j)
16
           store result in shared memory via atomic functions
17
       else
18
           if (iter_x == iter_y) then
19
               i = iter_v
               j = iter_v
21
               COMPUTE_STATISTICS(p_i, p_i)
               store result in shared memory via atomic functions
23
           i = iter_x + \frac{|\Omega|}{2}
           j = iter_y + \frac{|\overline{\Omega}|}{2}
           COMPUTE_STATISTICS(p_i, p_j)
           store result in shared memory via atomic functions
       UP\_DATE(iter_x, iter_y)
30 save Statistics values that are in shared memory into globa memory via atomic functions
   Output: Array \beta
```



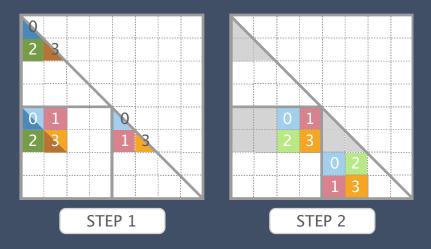
```
1 id_x = blockId.x*blockDim.x + threadId.x
                                                                      /* x threads coord in the GPU grid */
id_v = blockId.y*blockDim.y + threadId.y
                                                                      /* y threads coord in the GPU grid */
3 Set and Initialize shared memory in the block
4 __syncthreads()
5 iter_x = id_x
 6 iter_v = id_v
7 while (iter_x \& iter_y \in ChunkPoints)
                                                                   /* Chunk points belongs thread (x,y) */
 8 do
       j = iter_x + \frac{|\Omega|}{2}
       COMPUTE_STATISTICS(p_i, p_j)
11
       store result in shared memory via atomic functions
12
       if (iter_x > iter_y) then
13
14
           i = iter_v
15
           j = iter_x
           COMPUTE_STATISTICS(p_i, p_j)
16
           store result in shared memory via atomic functions
17
       else
18
           if (iter_x == iter_y) then
19
               i = iter_v
               j = iter_v
21
               COMPUTE_STATISTICS(p_i, p_i)
               store result in shared memory via atomic functions
23
           i = iter_x + \frac{|\Omega|}{2}
           j = iter_y + \frac{|\overline{\Omega}|}{2}
           COMPUTE_STATISTICS(p_i, p_j)
           store result in shared memory via atomic functions
       UP\_DATE(iter_x, iter_y)
30 save Statistics values that are in shared memory into globa memory via atomic functions
   Output: Array \beta
```



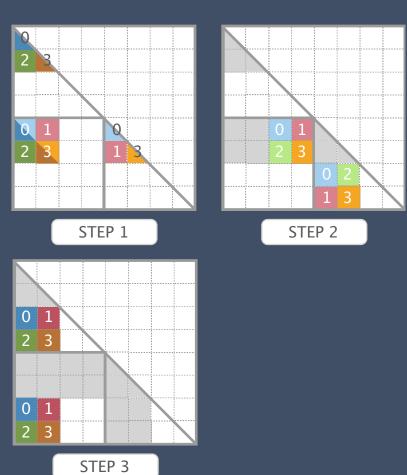
/* x threads coord in the GPU grid */

```
id_v = 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_v = id_v
7 while (iter_x \& iter_y \in ChunkPoints)
                                                                    /* Chunk points belongs thread (x,y) */
 8 do
       j = iter_x + \frac{|\Omega|}{2}
       i = iter_{v}
       COMPUTE_STATISTICS(p_i, p_j)
11
       store result in shared memory via atomic functions
12
       if (iter_x > iter_y) then
13
14
           i = iter_v
15
           j = iter_x
           COMPUTE_STATISTICS(p_i, p_j)
16
           store result in shared memory via atomic functions
17
       else
18
           if (iter_x == iter_y) then
19
               i = iter_v
               j = iter_v
21
22
               COMPUTE_STATISTICS(p_i, p_i)
               store result in shared memory via atomic functions
23
           i = iter_x + \frac{|\Omega|}{2}
           j = iter_y + \frac{|\overline{\Omega}|}{2}
           COMPUTE_STATISTICS(p_i, p_j)
           store result in shared memory via atomic functions
       UP\_DATE(iter_x, iter_y)
30 save Statistics values that are in shared memory into globa memory via atomic functions
   Output: Array \beta
```

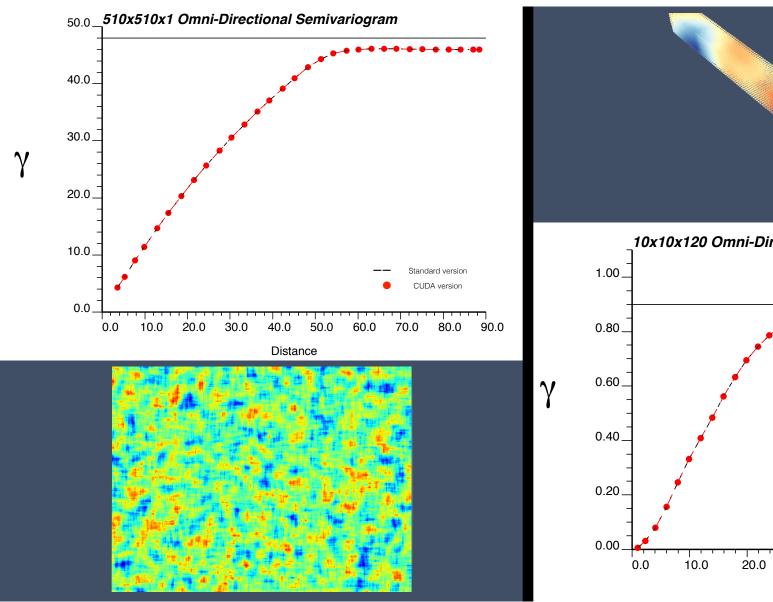
1 $id_x = blockId.x*blockDim.x + threadId.x$

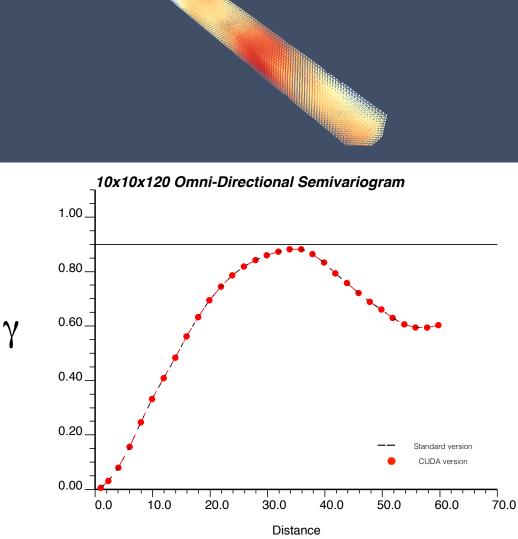


```
1 id_x = blockId.x*blockDim.x + threadId.x
                                                                      /* x threads coord in the GPU grid */
id_v = blockId.y*blockDim.y + threadId.y
                                                                      /* y threads coord in the GPU grid */
3 Set and Initialize shared memory in the block
4 __syncthreads()
5 iter_x = id_x
 6 iter_v = id_v
                                                                   /* Chunk points belongs thread (x,y) */
7 while (iter_x \& iter_y \in ChunkPoints)
 8 do
       j = iter_x + \frac{|\Omega|}{2}
       i = iter_{v}
11
       COMPUTE_STATISTICS(p_i, p_j)
       store result in shared memory via atomic functions
12
       if (iter_x > iter_y) then
13
14
           i = iter_v
15
           j = iter_x
           COMPUTE_STATISTICS(p_i, p_j)
16
           store result in shared memory via atomic functions
17
       else
18
           if (iter_x == iter_y) then
19
               i = iter_v
               j = iter_{v}
21
22
               COMPUTE_STATISTICS(p_i, p_i)
               store result in shared memory via atomic functions
23
           i = iter_x + \frac{|\Omega|}{2}
24
           j = iter_y + \frac{|\overline{\Omega}|}{2}
           COMPUTE_STATISTICS(p_i, p_j)
           store result in shared memory via atomic functions
       UP\_DATE(iter_x, iter_y)
30 save Statistics values that are in shared memory into globa memory via atomic functions
   Output: Array \beta
```



4 685.96	78 210 .81 3.130 1.04 5.89 8.81 4.83 8.88 2.8 16.14 2.8	87.8 6,4 187.1	36.89 34.91 869.93 3,690.35 4,555.87		2,804.03 1,239.03
4,887.11 1,880.78 1,680.64 6,889.89 1,563.04 6,993.83 908.33 3,803 4,685.96 4,58	78 4,537 .63 210 .81 3,136 .88 6,89 .88 4,83 .65.14 2,8	6,08 1,78 0,91 0,043 1,7 1,7 1,7 1,7 1,7 1,7 1,7 1,7	869.93 869.93 ,590.35		2,904.03
1,880.78 1,680.64 5,889.89 1,663.04 5,993.83 908.33 3,809 4,68	63 210 .81 3,130 1.04 5,89 8.81 4,83 8.88 2,88 65.14 2,83 65.14 3,93	0.78 0.91 0.43 35.95 384.99	869.93 869.93 ,590.35		2,904.03
1,880.84 6,889.89 1,563.04 5,993.83 908.33 4,685.96 4,685.96	81 3,130 6,89 8.81 4,83 8.88 2,8 65.14 3,9	0.91 0.43 35.95 384.99	A 555.85		2,904.03
5,889.89 1,663.04 5,118 5,993.83 908.33 4,58	8.81 6,89 8.88 4,83 8.514 2,8	35.95	A 555.9	36	
1,563.04 5,118 5,993.83 3,809 908.33 4,58	8.88 4,8 8.65.14 2,8	84.99	3.89	301	22 3
6,993.83 908.33 4,58	8.88 2,8 86.14 3,5	384.99	3.80		4804.
908.33 4,58	3,5.14	0 221		325.89	088.7
4 685.96	0 381	930.22	7	3,210.78	8,70
	10.00	623.09		- 200-101	1.5
3 9 18 88	84.54	3,478.31	15/8		
734.69	21.84	459.93	5,108.03	8,302.04	1
2 9 2 9 0 2 5 9	181.019	989.09		8,822.1	08
E 990 93	985-4	8,392.71	- 097.501	2,981	180
1,781.07	818.78	9,930.77	- 2000.2	809	All a comments
7,781.0	,885.18	3.091.99	8,398.91		
4,318.73	571.01	946.18	8,398.0	100	401.00
4,885.93	571.0	3,110.91	2 890.	30	
671.09	3,410.81	3,770 90	1881	.93	8504 93
2 490 31	3 184.08	3,630.90	81	1.40	9,934.93
2 224 03	6 904.51	7,890.83		37.04	3.470.0
	4 340.44	8,557.97	8,4	84.88	2,430
8,74	5,189.84	9.738.9	100	1,890.008,1	6,94
4,230.00	1,811.11	7.093.	09/	1,850.0	1.7
1,252.34	1,81	879	1.93	6,441.38	- 11
499.11	7,410.18	- 09	80.8	1881.03	
7,890.93	1571.41	3,98	- 03	3,450.38	
7,850	5,810.18	9.2	79.03	3,40 44	1
1,781.27	6,810	- 0.5	88.00	3,310.41	1
- 090.93	8,322.81	3,	-0.21	8, 198,8	10.
5,890.93	5.581.33	3	772.21	0.730	97
8,912.88			100.001	2,310	
	444.66	-			0.30/
1,785.37	Res		758.016	3.70	
		UII S	41		4 4 70 4 7
2.981.20					
Allie		-	70		6.70 (6.7)
NDVANCED LARGEATORY TOO	4,814,3	9	640.10		THE RESERVE THE PARTY NAMED IN
GEOSTATISTICAL SUPERCOMPUTING	4.814.0	-			1,909.35
3,106.1		0	1.989.93	-	





Dell Precision T7600

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



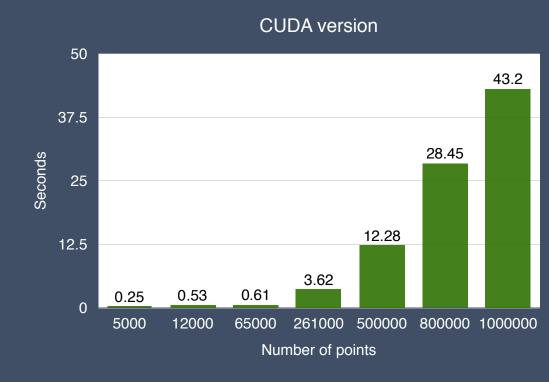
Standard Version 2300 2095 1725 1330 Seconds 1150 518.69 575 143.2 9.52 8.0 4.68 12000 65000 261000 500000 800000 1000000 5000 Number of points





Tesla c2075

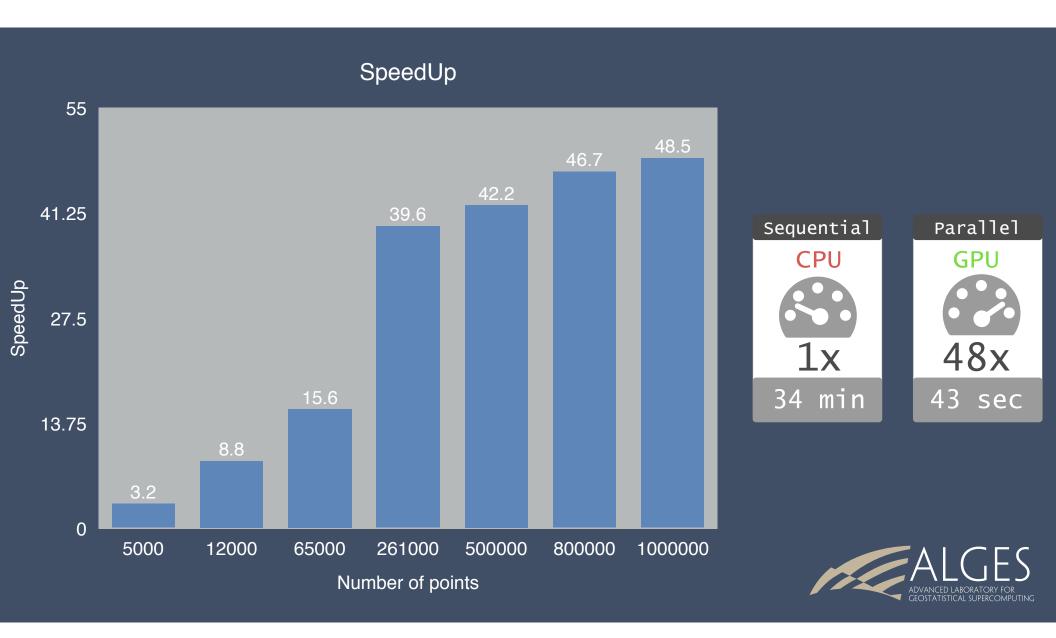
- 448 CUDA cores
- 1.15 GHz Frequency of CUDA cores
- 6GB RAM
- 114 GB/sec Memory bandwidth

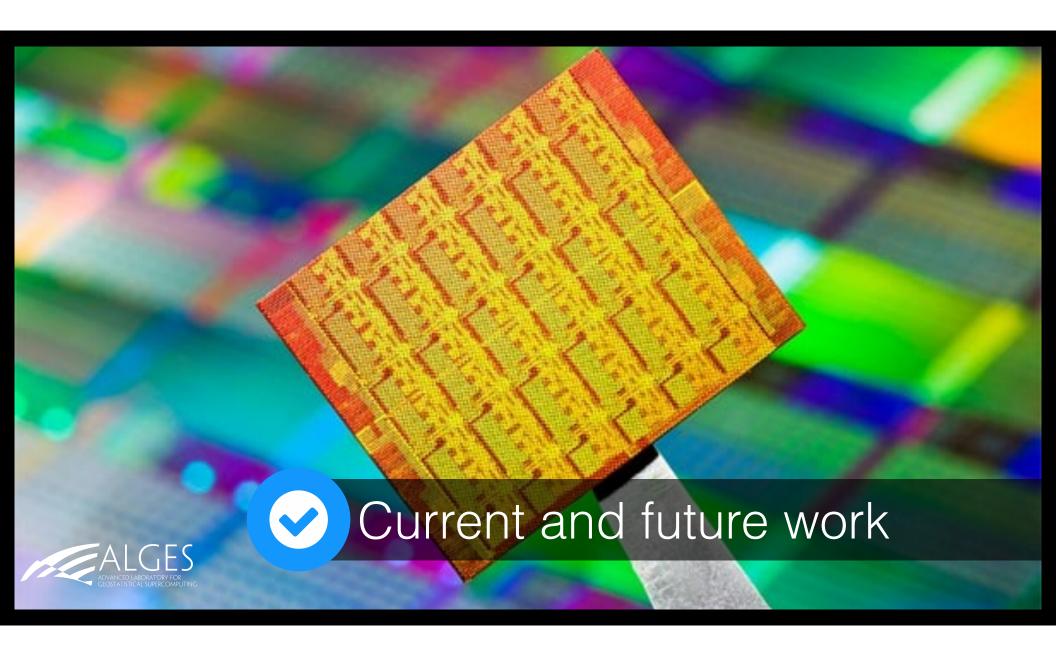




Stable And Mesision 2360 2095 137725 1330 Seeends 1126 518.69 1527.5 143.2 4.68 9.52 8.0 65000 261000 500000 800000 1000000 12000 5000 Number of points









• Release the first GSLIB-CUDA version with these three methods

