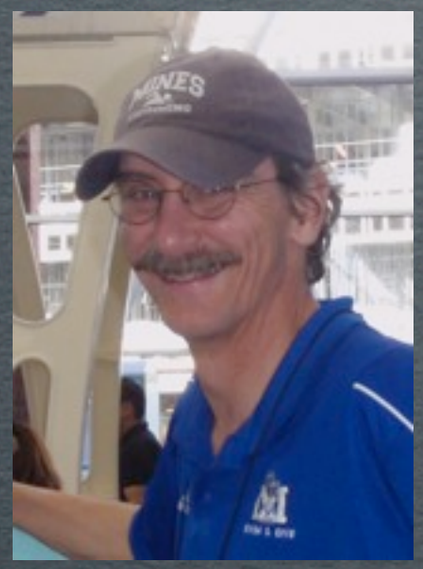
The Mines JTK and multicore computing



Dave Hale
Center for Wave Phenomena
Colorado School of Mines

Mines Java Toolkit (JTK)

Mines Java Toolkit (JTK)

a software library

like libcwp, VTK, ...

Mines Java Toolkit (JTK)

not a processing system

like Seismic Unix, ProMAX, ...

Mines Java Toolkit (JTK)

not an interpretation system like OpendTect, SeisWorks, ...

Mines Java Toolkit (JTK)

used in systems for processing, interpretation, teaching, finance, ...

Mines Java Toolkit (JTK)

13 packages

280 classes

Mines Java Toolkit (JTK)

cross-platform

Windows, Linux, Mac OS X

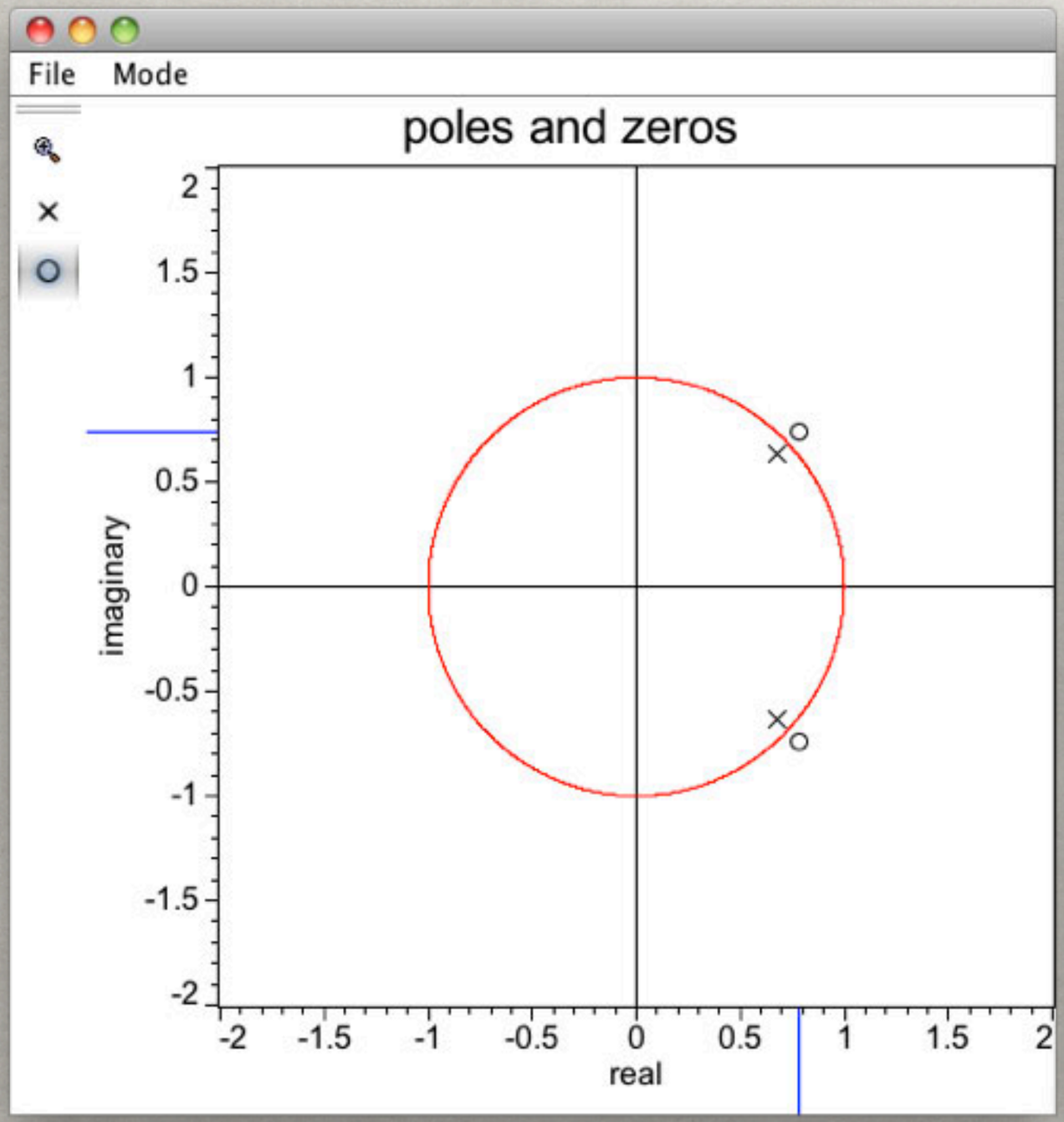
Mines Java Toolkit (JTK)

115,000 lines of source code
+ 48,000 lines of comments

Mines Java Toolkit (JTK)

open-source
Common Public License
used in commercial software

interactive graphics



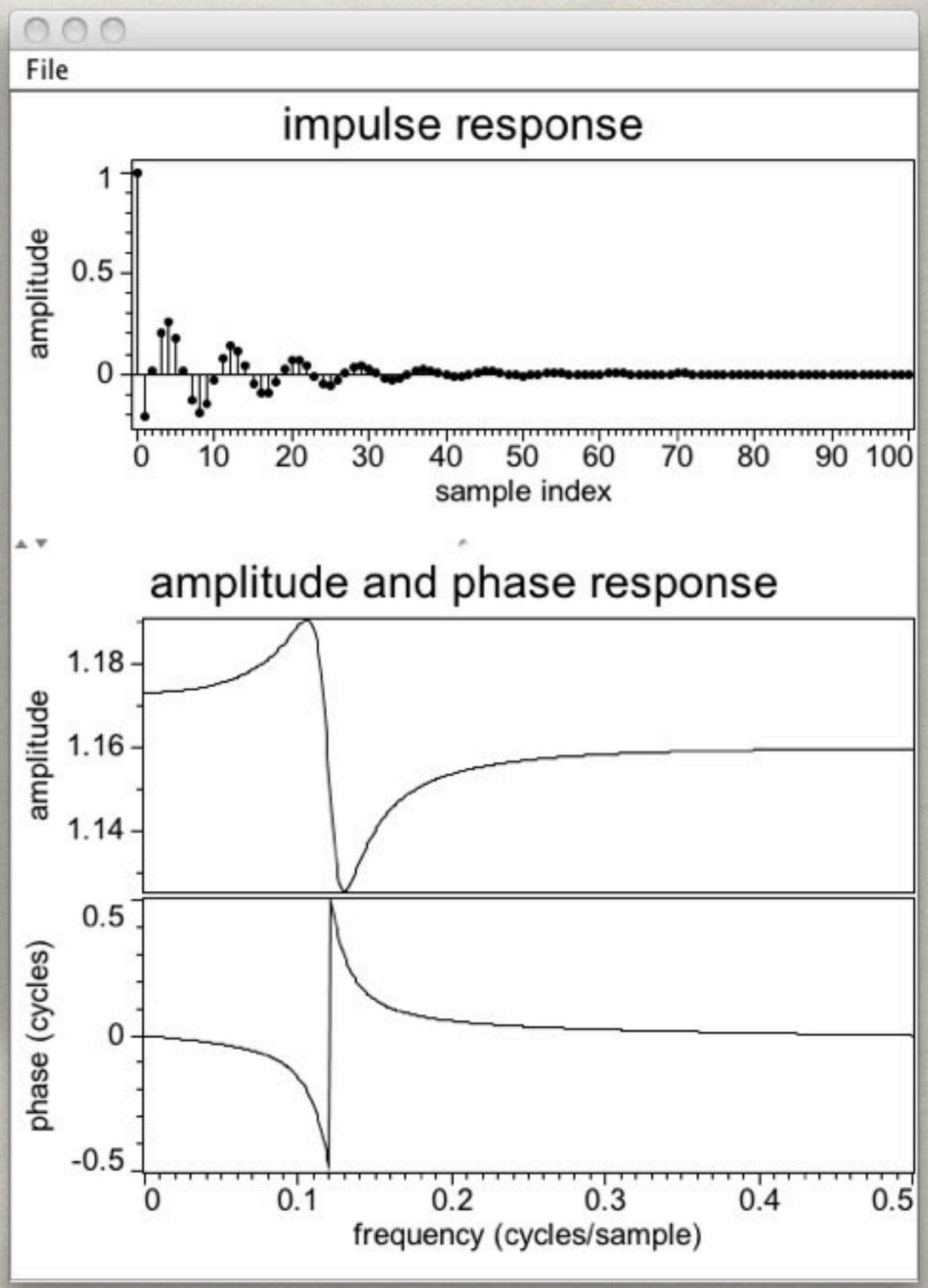
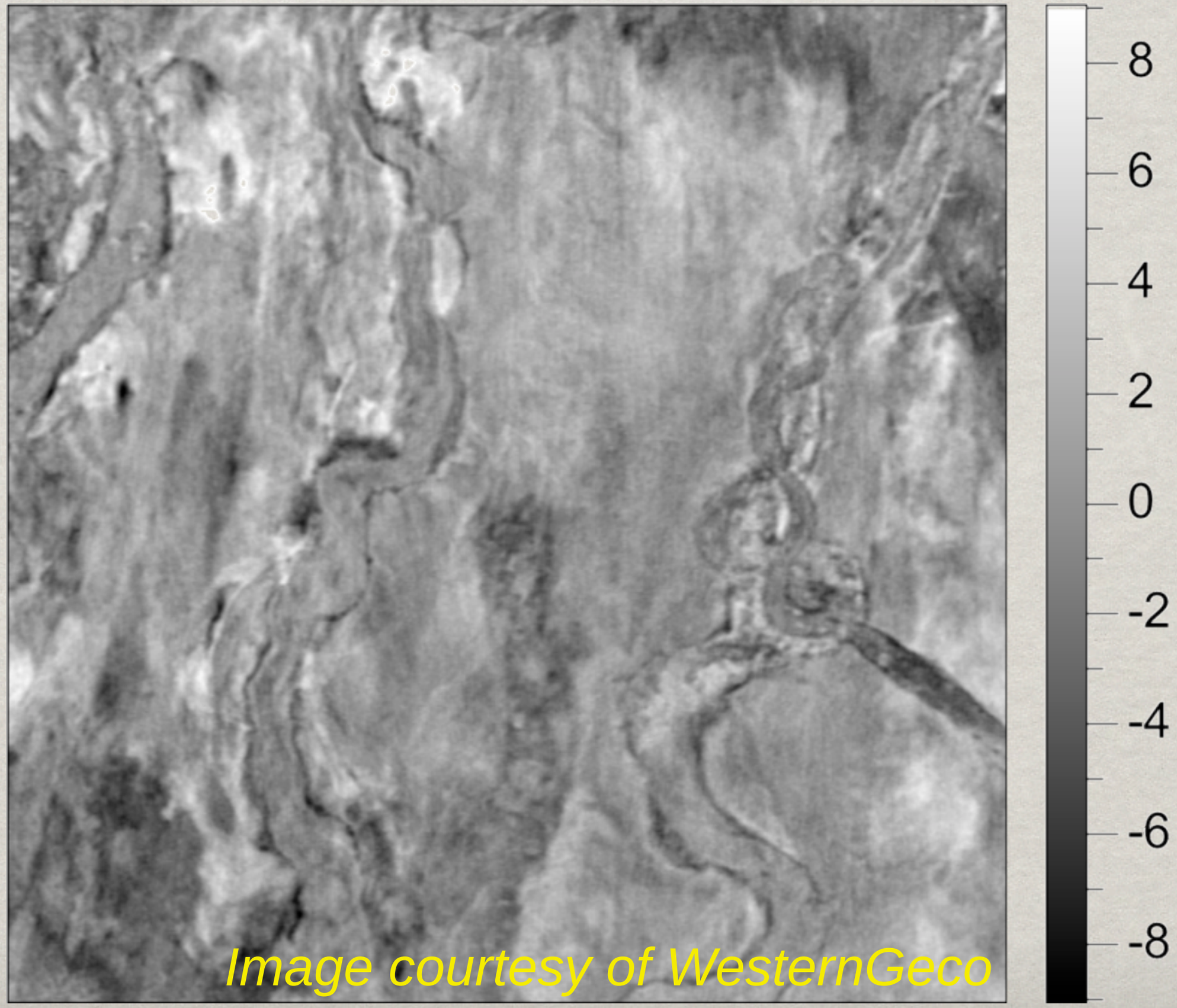
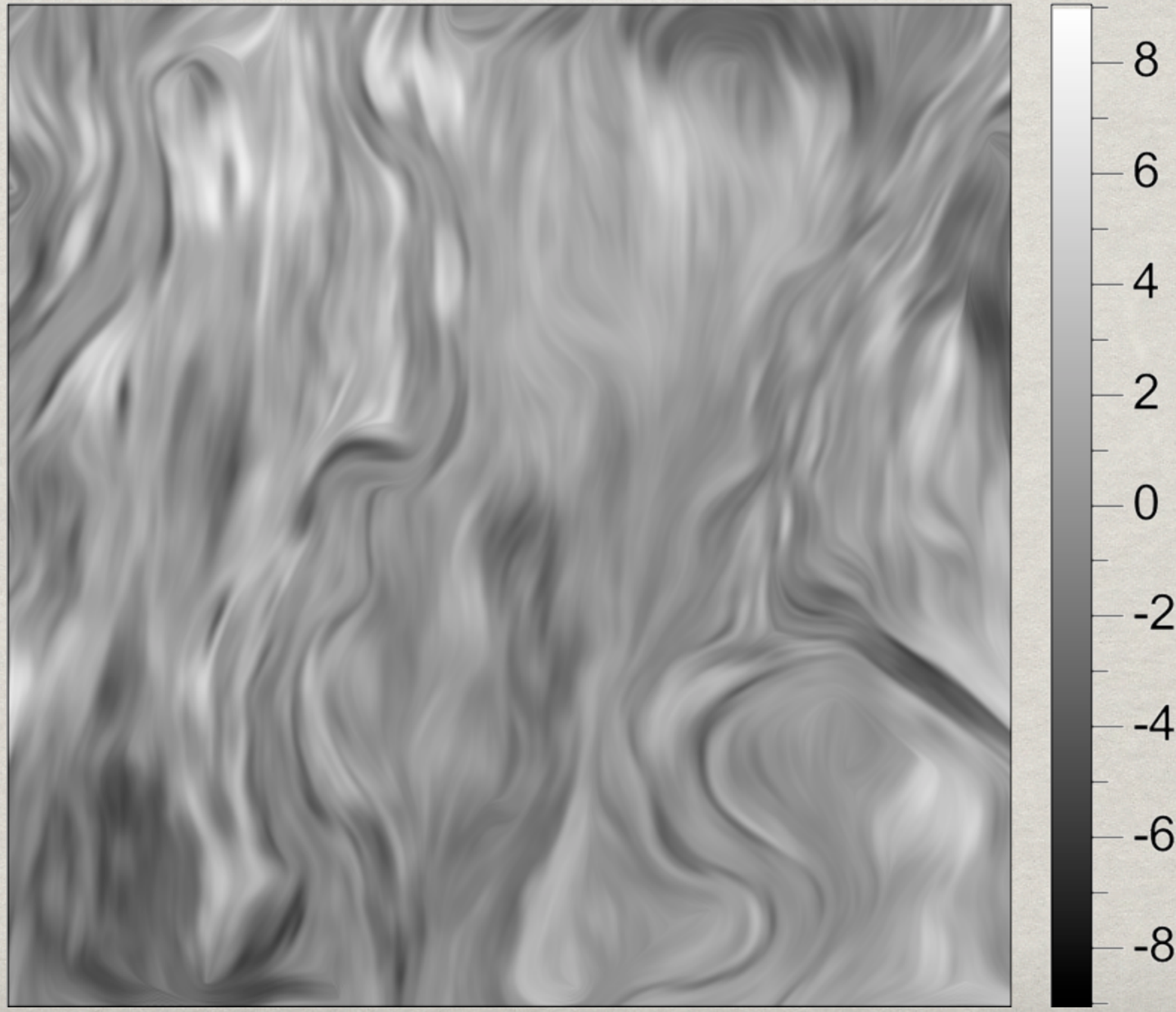


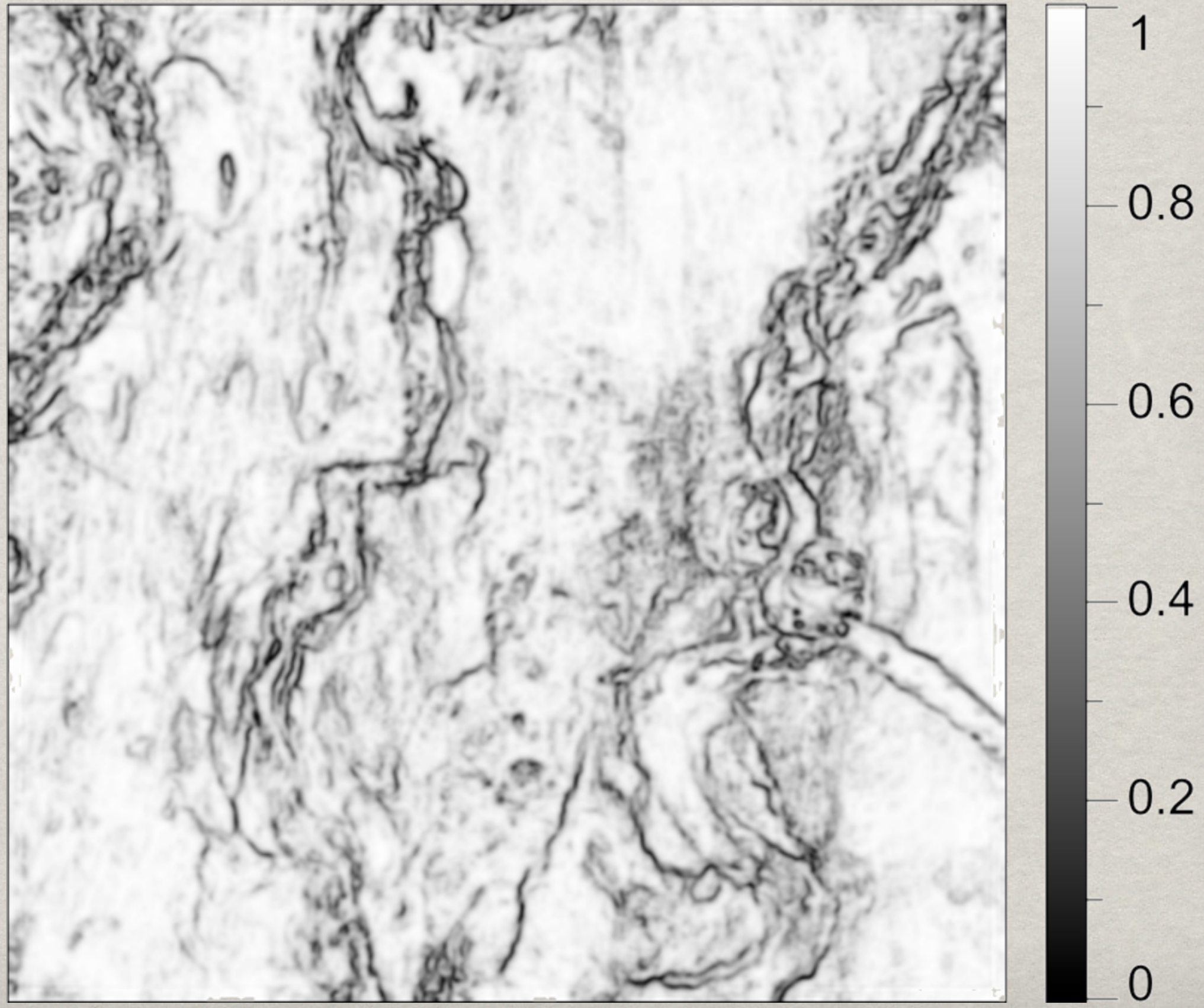
image processing



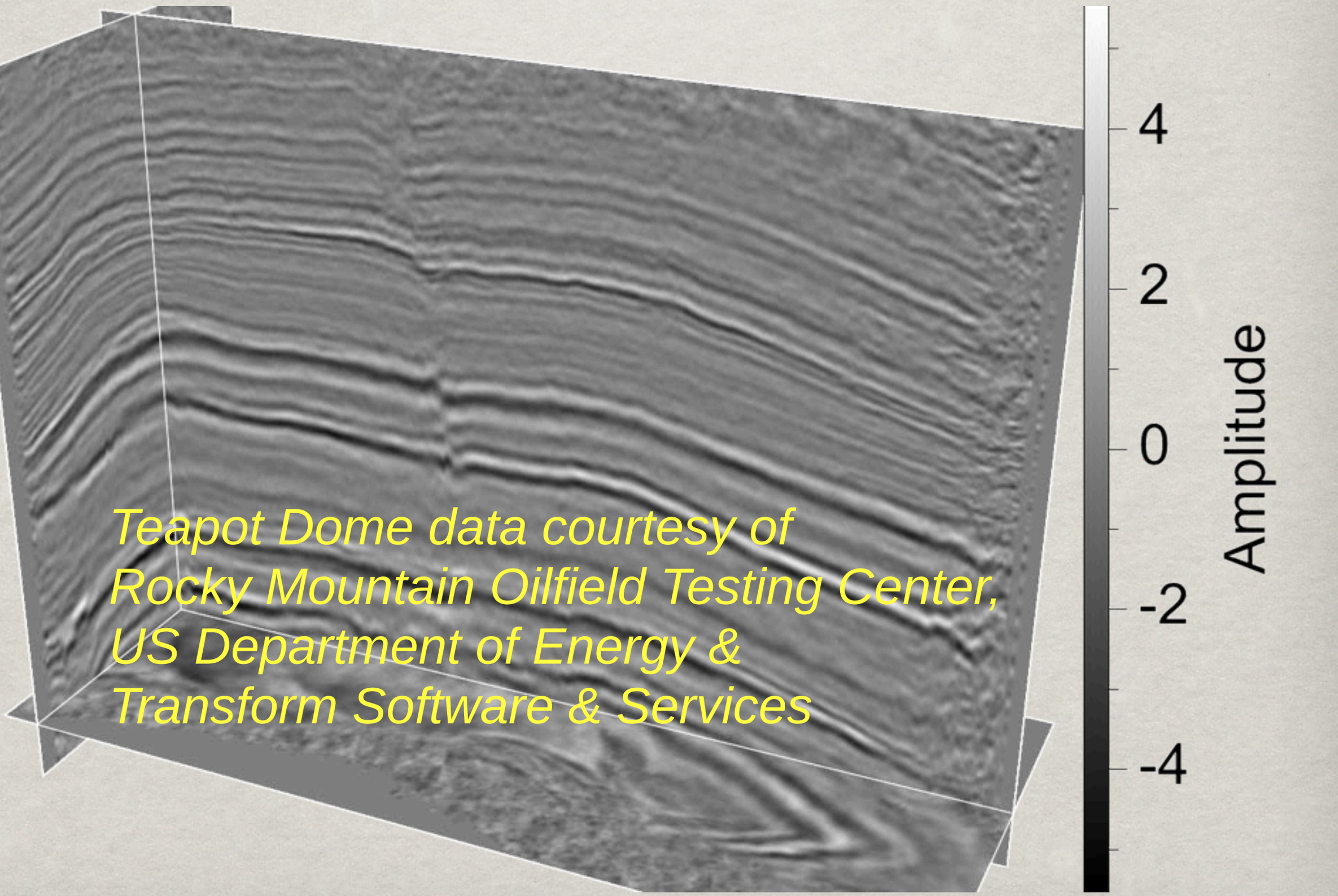
structure-oriented smoothing



structure-oriented semblance

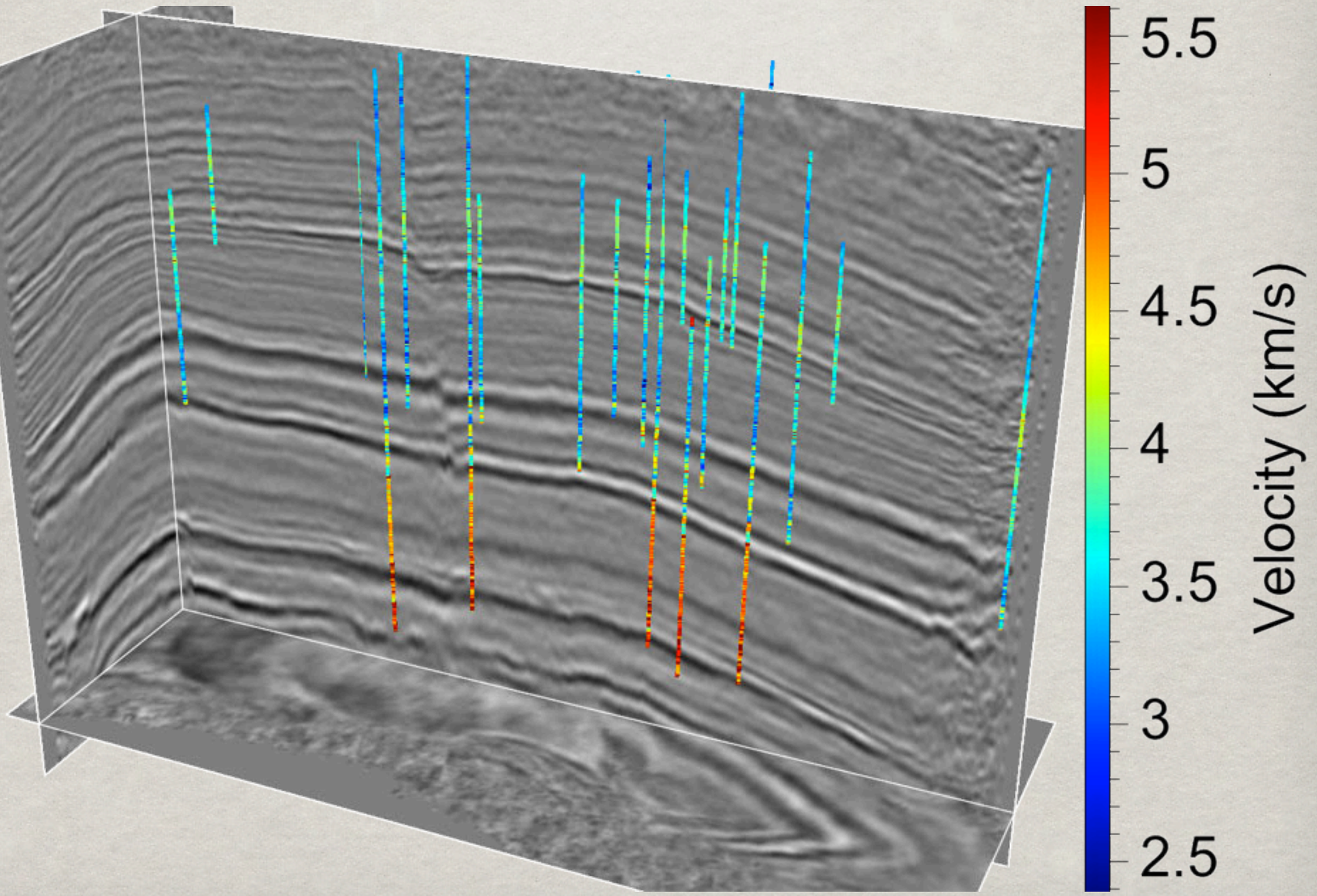


3D interactive visualization

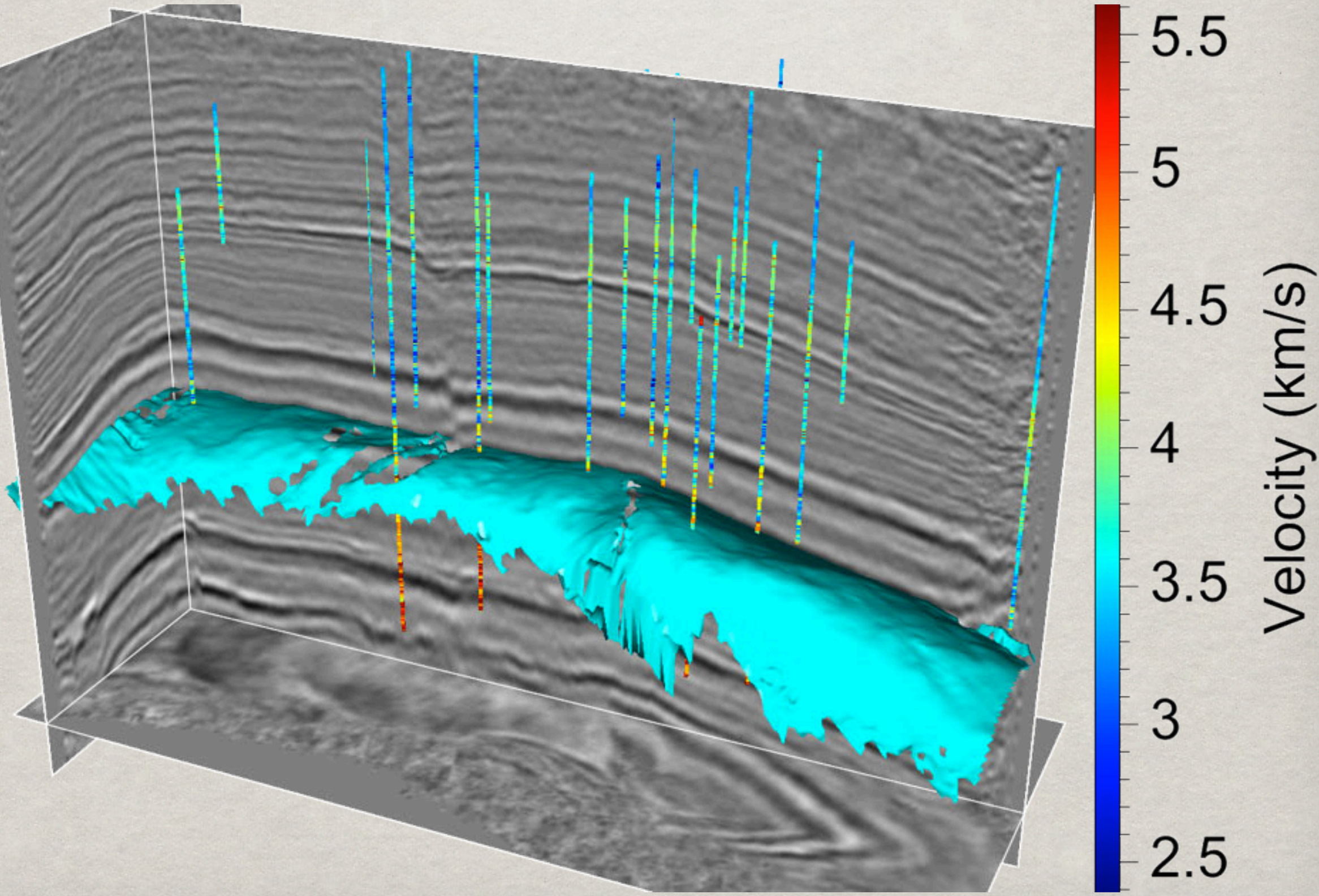


Teapot Dome data courtesy of Rocky Mountain Oilfield Testing Center, US Department of Energy & Transform Software & Services

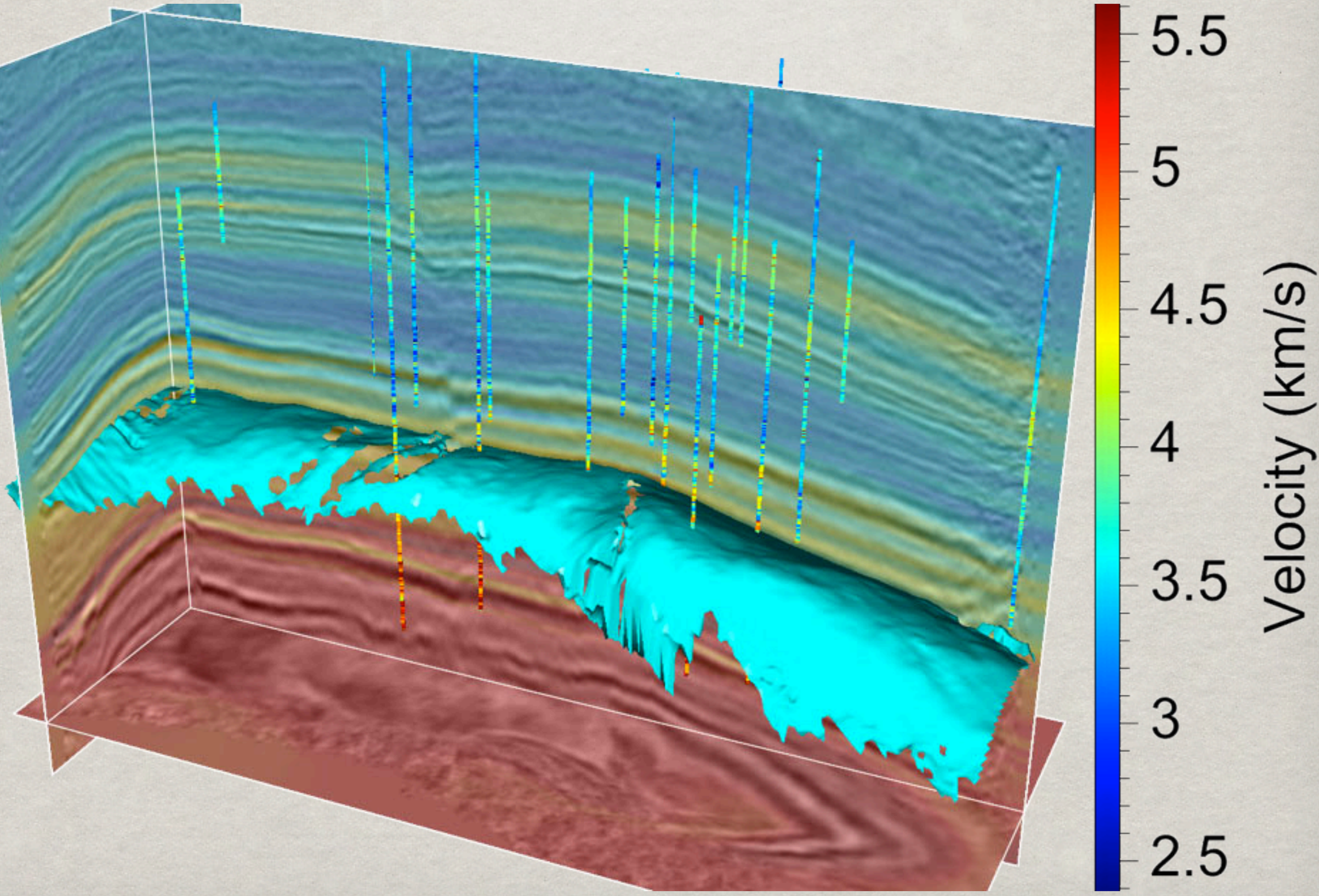
# of multiple objects



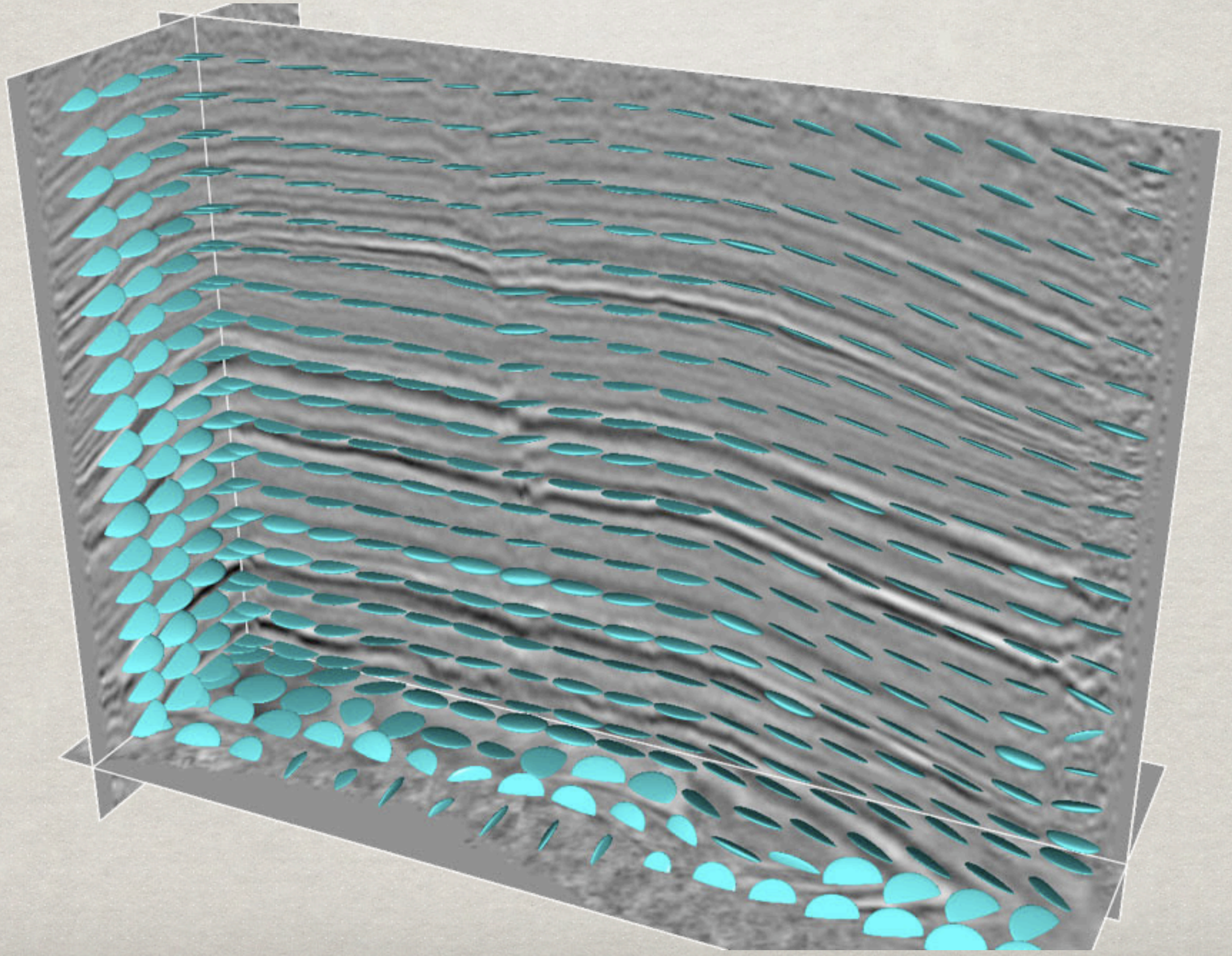
# of multiple objects



with transparency

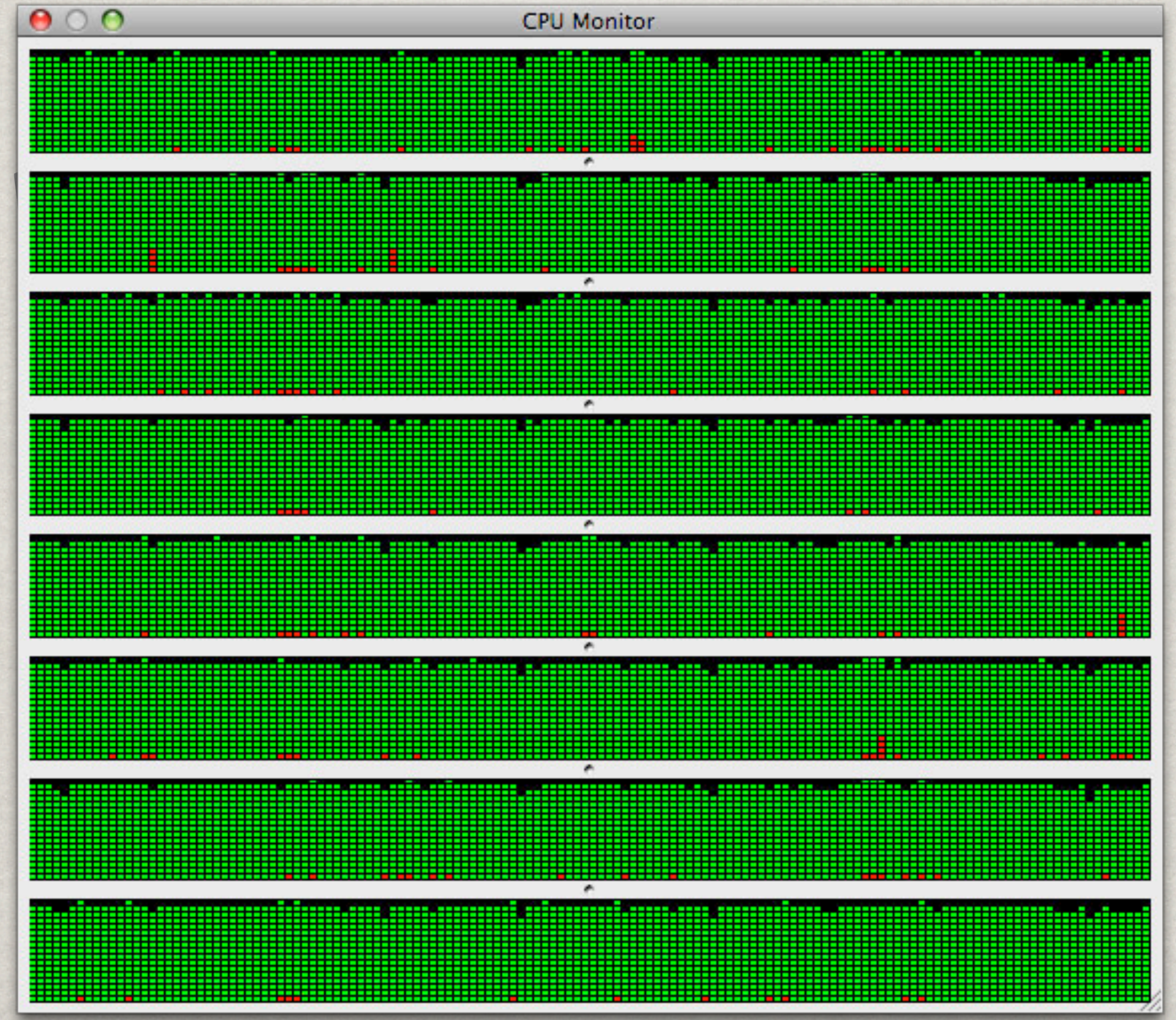


with custom objects

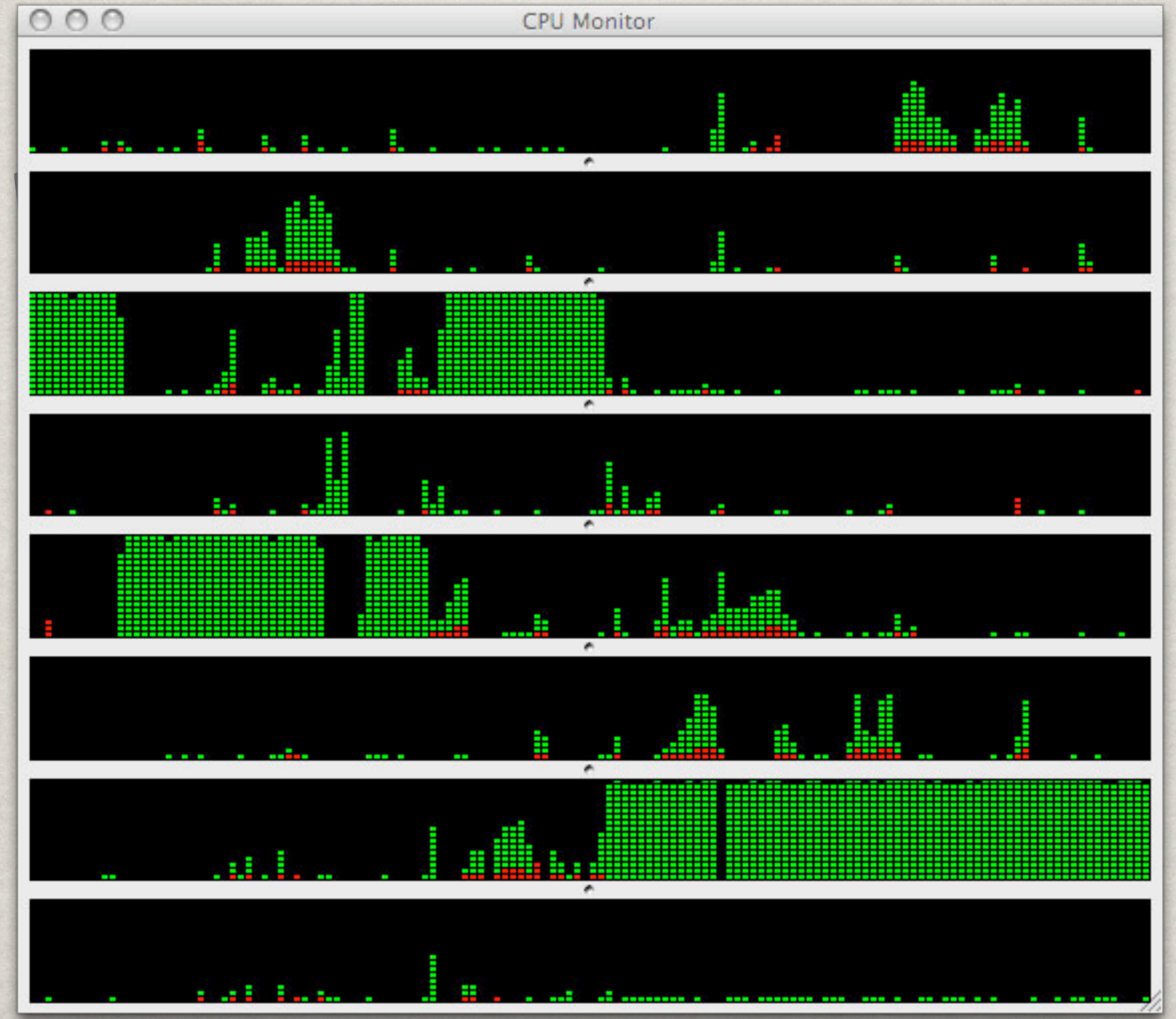


multicore computing

CPU monitor (I wish)



CPU monitor (typical)



Dear Students,

If you are not yet writing parallel programs, say, because you work only with small 2D images, then your half-life is roughly two years, which is about how long it takes for the number of cores to double.

Cheers,
Dave

November 29, 2010

The half-life of
any *software library*
not designed for
parallel computing
is two years.

Second-order linear recursion

for $i = 0, 1, 2, \ldots, n-1$

$$y_i = b_0 x_i + b_1 x_{i-1} + b_2 x_{i-2} - a_1 y_{i-1} - a_2 y_{i-2}$$

- useful
- easy to implement
- inherently serial (not parallel)
- 9 FLOPs per load and store

Second-order linear recursion

```
float yim2 = 0.0f;
float yim1 = 0.0f;
float xim2 = 0.0f;
float xim1 = 0.0f;
for (int i=0; i<n; ++i) {
  float xi = x[i];
  float yi = b0*xi+b1*xim1+b2*xim2
                        -a1*yim1-a2*yim2;
  y[i] = yi;
  yim2 = yim1;
  yim1 = yi;
  xim2 = xim1;
  xim1 = xi;
}
```

Second-order linear recursion

```
float yim2 = 0.0f;
float yim1 = 0.0f;
float xim2 = 0.0f;
float xim1 = 0.0f;
for (int i=0; i<n; ++i) {
  float xi = x[i];
  float yi = b0*xi+b1*xim1+b2*xim2
                         -a1*yim1-a2*yim2;
  y[i] = yi;
  yim2 = yim1;
  yim1 = yi;
  xim2 = xim1;
  xim1 = xi;
}
```

load

Second-order linear recursion

```
float yim2 = 0.0f;
float yim1 = 0.0f;
float xim2 = 0.0f;
float xim1 = 0.0f;
for (int i=0; i<n; ++i) {
  float xi = x[i];
  float yi = b0*xi+b1*xim1+b2*xim2
                      -a1*yim1-a2*yim2;
  y[i] = yi;
  yim2 = yim1;
  yim1 = yi;
  xim2 = xim1;
  xim1 = xi;
}
```

compute

Second-order linear recursion

```
float yim2 = 0.0f;
float yim1 = 0.0f;
float xim2 = 0.0f;
float xim1 = 0.0f;
for (int i=0; i<n; ++i) {
   float xi = x[i];
   float yi = b0*xi+b1*xim1+b2*xim2
                       -a1*yim1-a2*yim2;
```

store `y[i] = yi;`

```
   yim2 = yim1;
   yim1 = yi;
   xim2 = xim1;
   xim1 = xi;
}
```

Second-order linear recursion

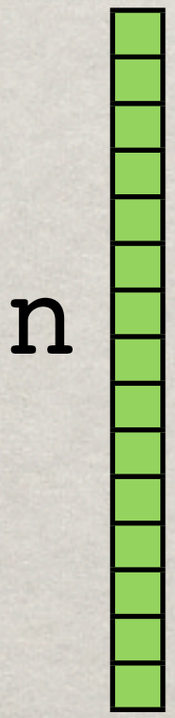
```
float yim2 = 0.0f;
float yim1 = 0.0f;
float xim2 = 0.0f;
float xim1 = 0.0f;
for (int i=0; i<n; ++i) {
  float xi = x[i];
  float yi = b0*xi+b1*xim1+b2*xim2
                        -a1*yim1-a2*yim2;
  y[i] = yi;
  yim2 = yim1;
  yim1 = yi;
  xim2 = xim1;
  xim1 = xi;
}
```

1D arrays (C/C++)

```
void solr1(float a1, float a2,
           float b0, float b1, float b2,
           int n, float* x, float* y) {
   ... // details inside library code
}
```

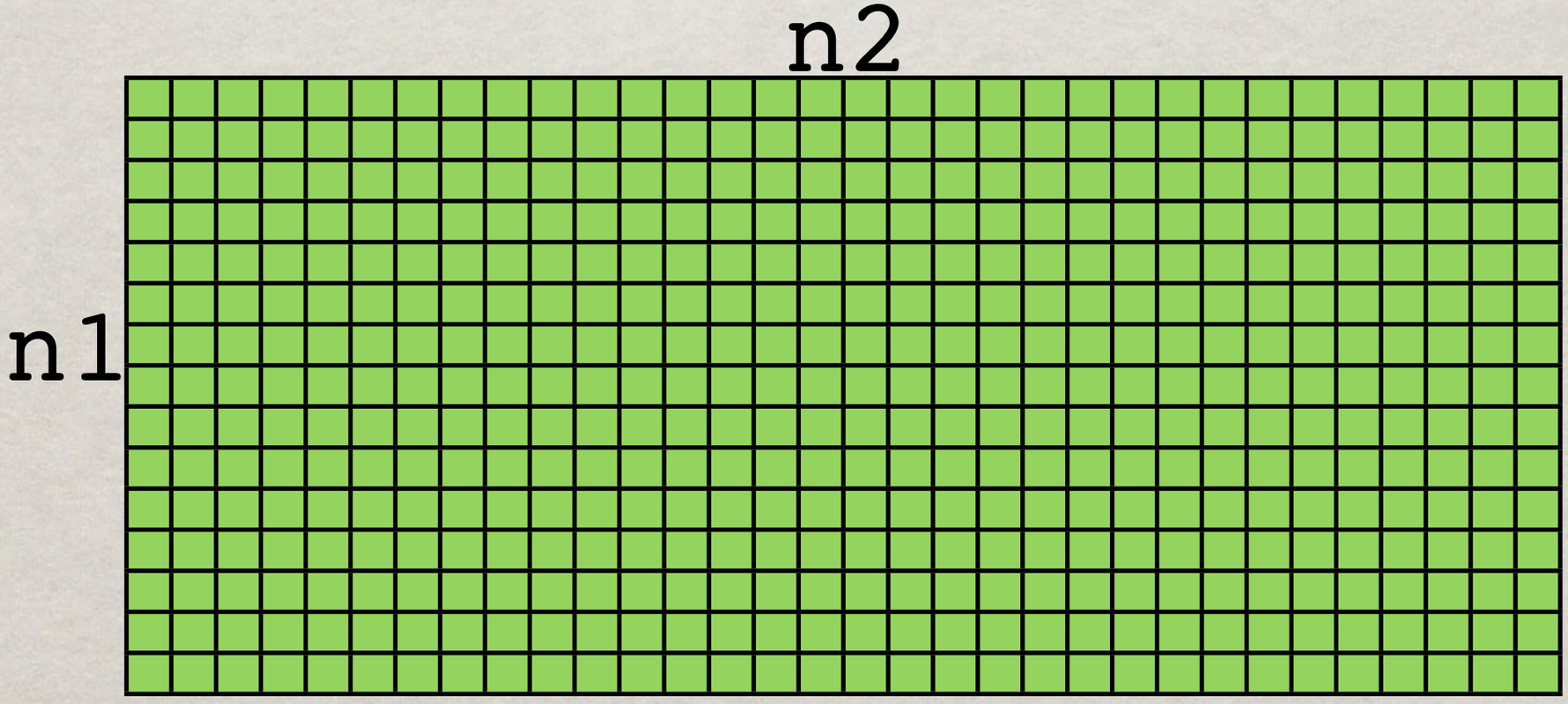
1D arrays (C/C++)

```
void solr1(float a1, float a2,
           float b0, float b1, float b2,
           int n, float* x, float* y) {
   ... // details inside library code
}
```



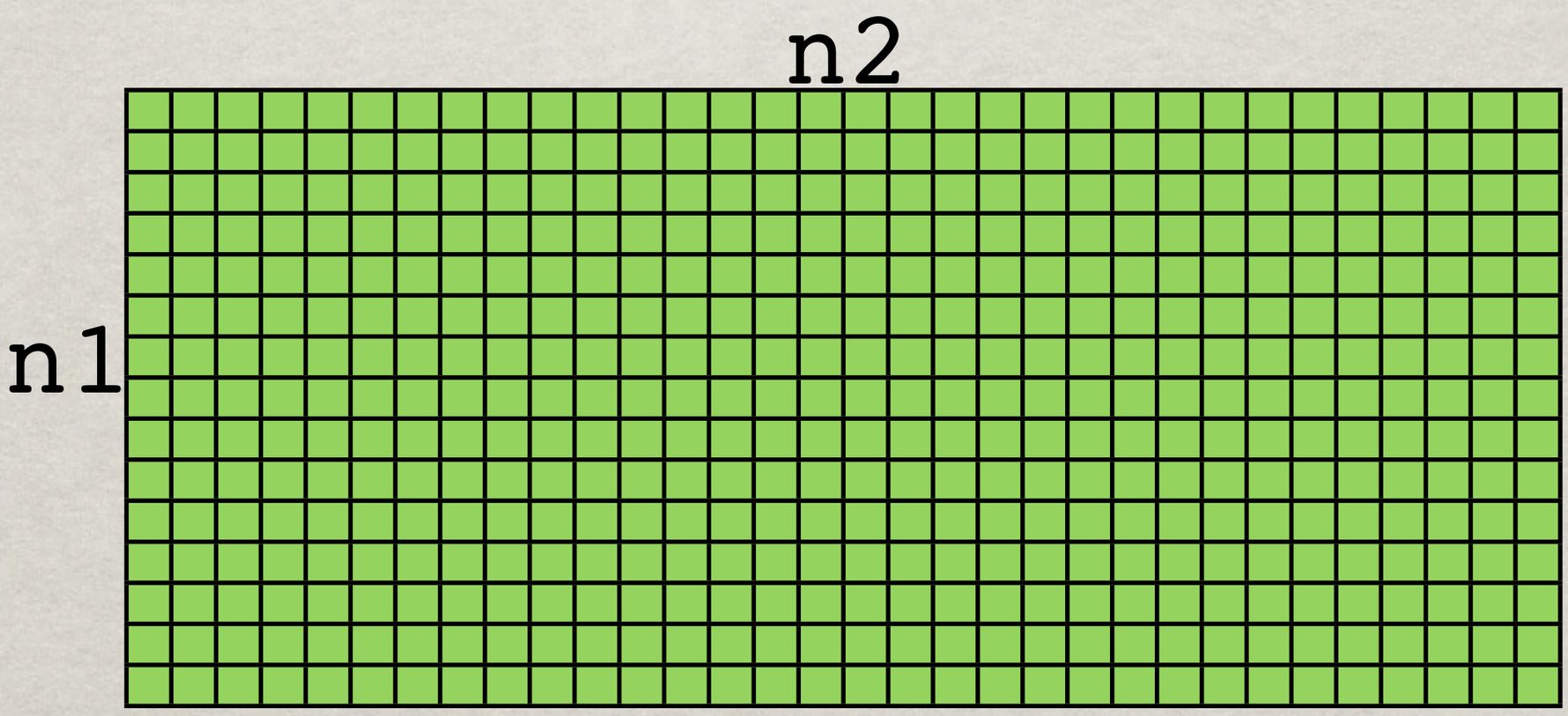
2D arrays (C/C++)

```
void solr2(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2,
           float** x, float** y) {
   for (int i2=0; i2<n2; ++i2)
      solr1(a1,a2,b0,b1,b2,n1,x[i2],y[i2]);
}
```



Serial (C/C++)

```
void solr2(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2,
           float** x, float** y) {
  for (int i2=0; i2<n2; ++i2)
    solr1(a1,a2,b0,b1,b2,n1,x[i2],y[i2]);
}
```



Serial (C/C++)

```
void solr2(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2,
           float** x, float** y) {
   for (int i2=0; i2<n2; ++i2)
      solr1(a1,a2,b0,b1,b2,n1,x[i2],y[i2]);
}
```

the problem

Parallel (C/C++)

```
void solr2(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2,
           float** x, float** y) {
  #pragma omp parallel for schedule(dynamic)
  for (int i2=0; i2<n2; ++i2)
    solr1(a1,a2,b0,b1,b2,n1,x[i2],y[i2]);
}
```

OpenMP

Serial (C/C++)

```
void solr2(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2,
           float** x, float** y) {
  for (int i2=0; i2<n2; ++i2)
    solr1(a1,a2,b0,b1,b2,n1,x[i2],y[i2]);
}
```

Serial (Java)

```
void solr2(float a1, float a2,
           float b0, float b1, float b2,
           float[][] x, float[][] y) {
   int n2 = x.length;
   for (int i2=0; i2<n2; ++i2)
      solr1(a1,a2,b0,b1,b2,n1,x[i2],y[i2]);
}
```

Parallel (Java)

```
void solr2(float a1, float a2,
           float b0, float b1, float b2,
           float[][] x, float[][] y) {
  int n2 = x.length;
  loop(n2, new LoopInt() {
  public void compute(int i2) {
    solr1(a1,a2,b0,b1,b2,x[i2],y[i2]);
  }});
}
```

Java fork-join framework
(with edu.mines.jtk.util.Parallel)

Serial (Scala)

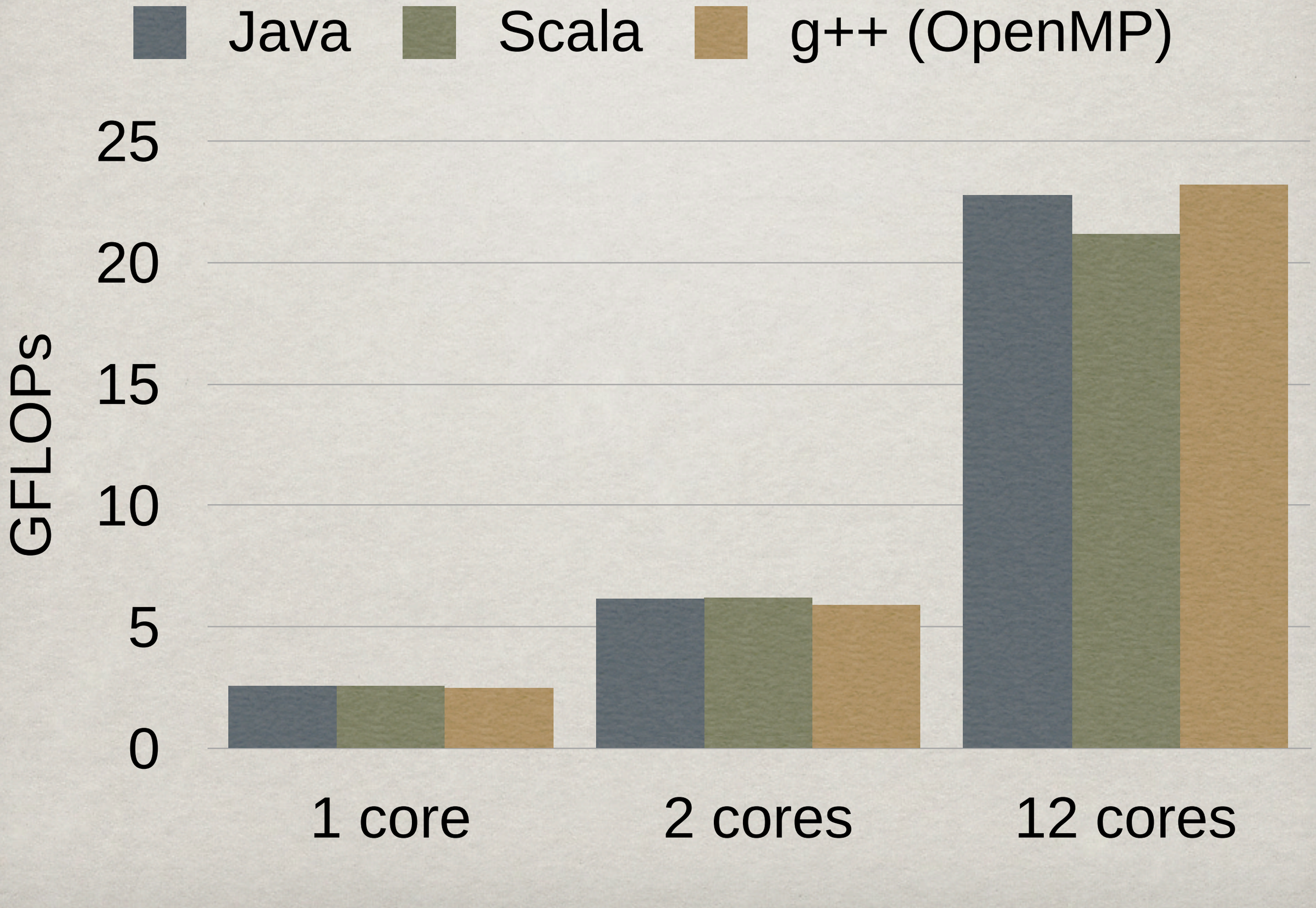
```
def solr2(a1:Float, a2:Float,
          b0:Float, b1:Float, b2:Float,
          x:Float2 x, y:Float2):Unit = {
   x.indices.foreach(
      i2=>solr1(a1,a2,b0,b1,b2,x(i2),y(i2))
}
```

Parallel (Scala)

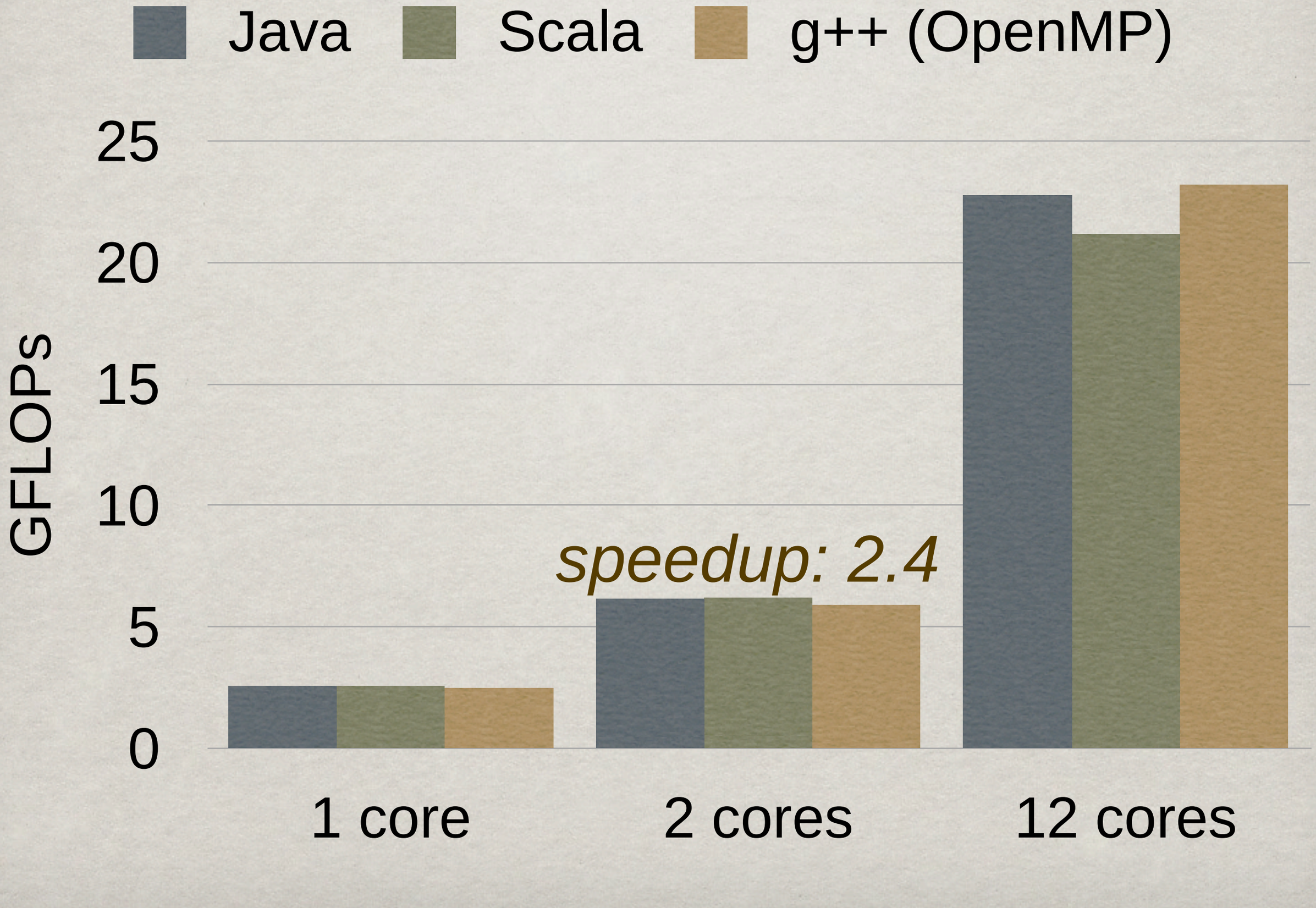
```
def solr2(a1:Float, a2:Float,
          b0:Float, b1:Float, b2:Float,
          x:Float2 x, y:Float2):Unit = {
   x.indices.par.foreach(
      i2=>solr1(a1,a2,b0,b1,b2,x(i2),y(i2))
}
```

Scala parallel arrays

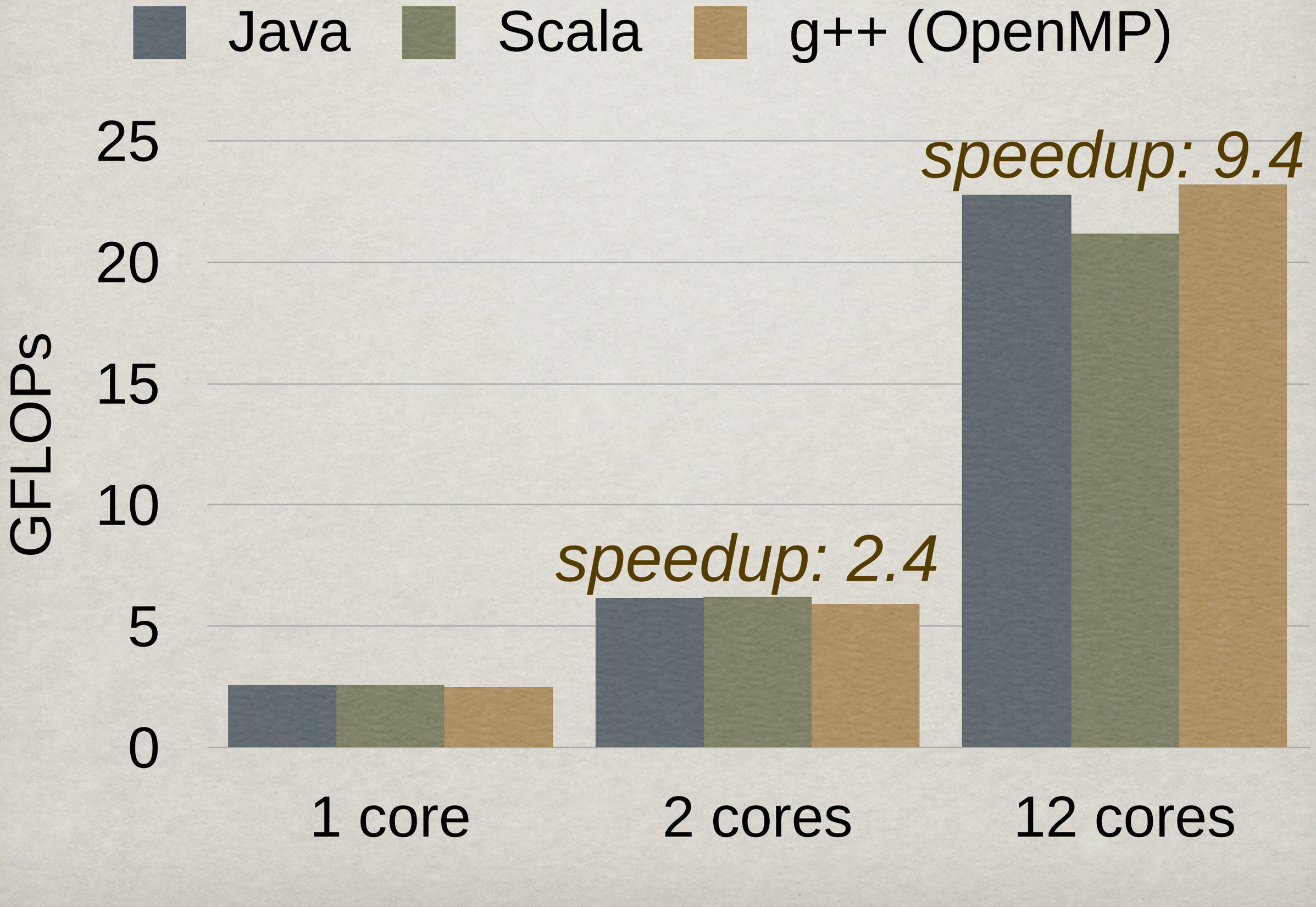
2D: 1000 x 50000



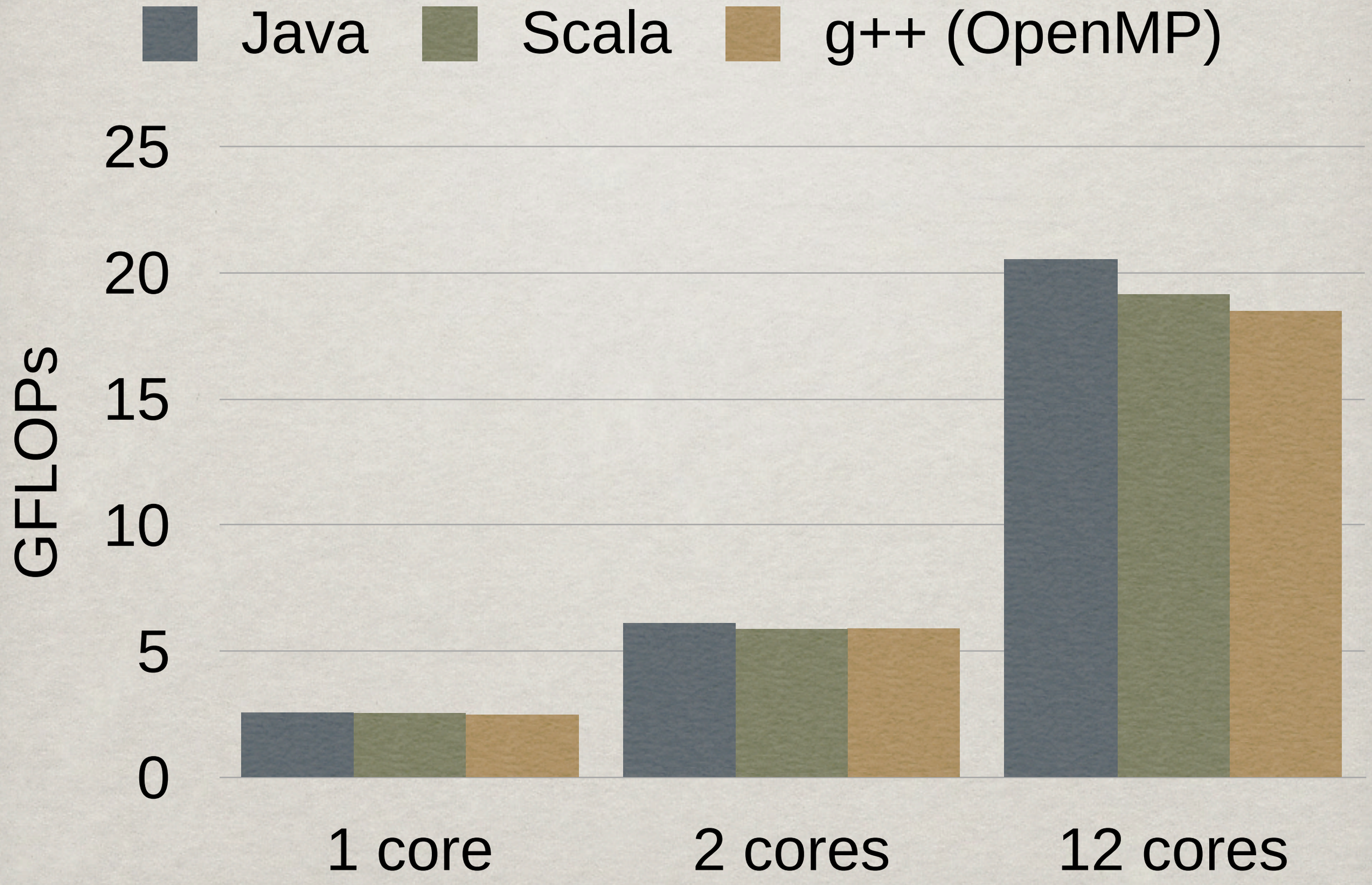
2D: 1000 x 50000



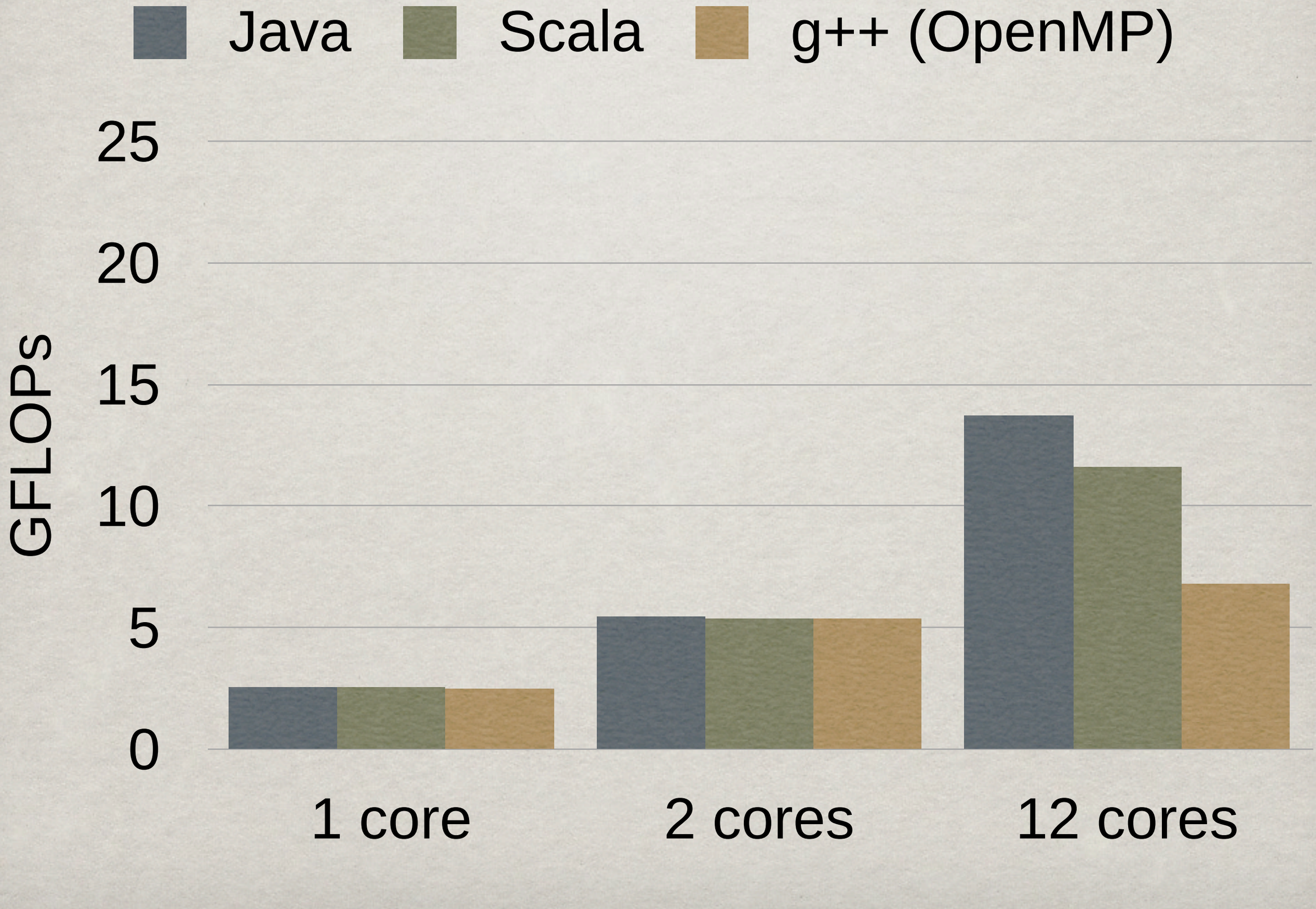
2D: 1000 x 50000



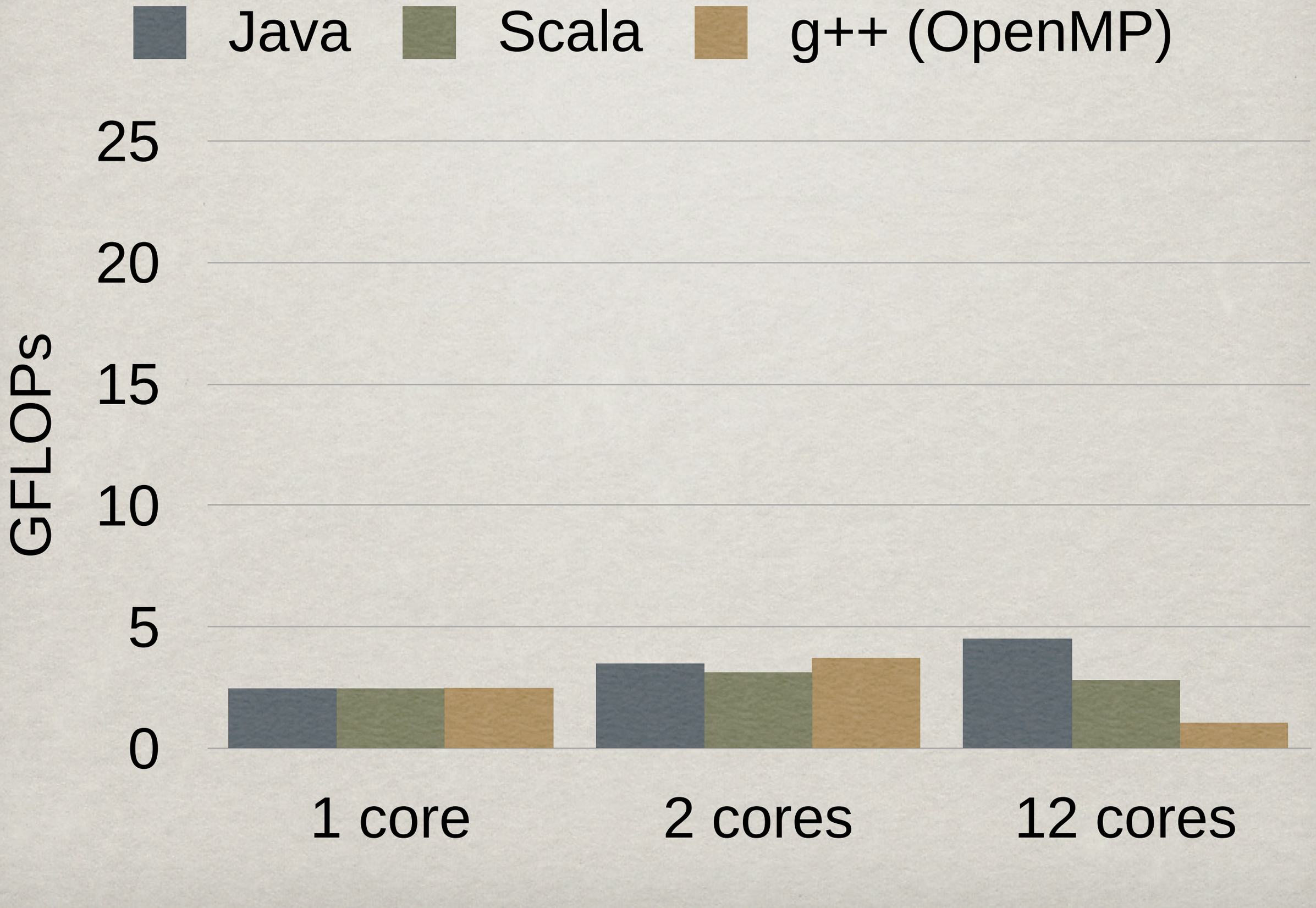
2D: 1000 x 5000



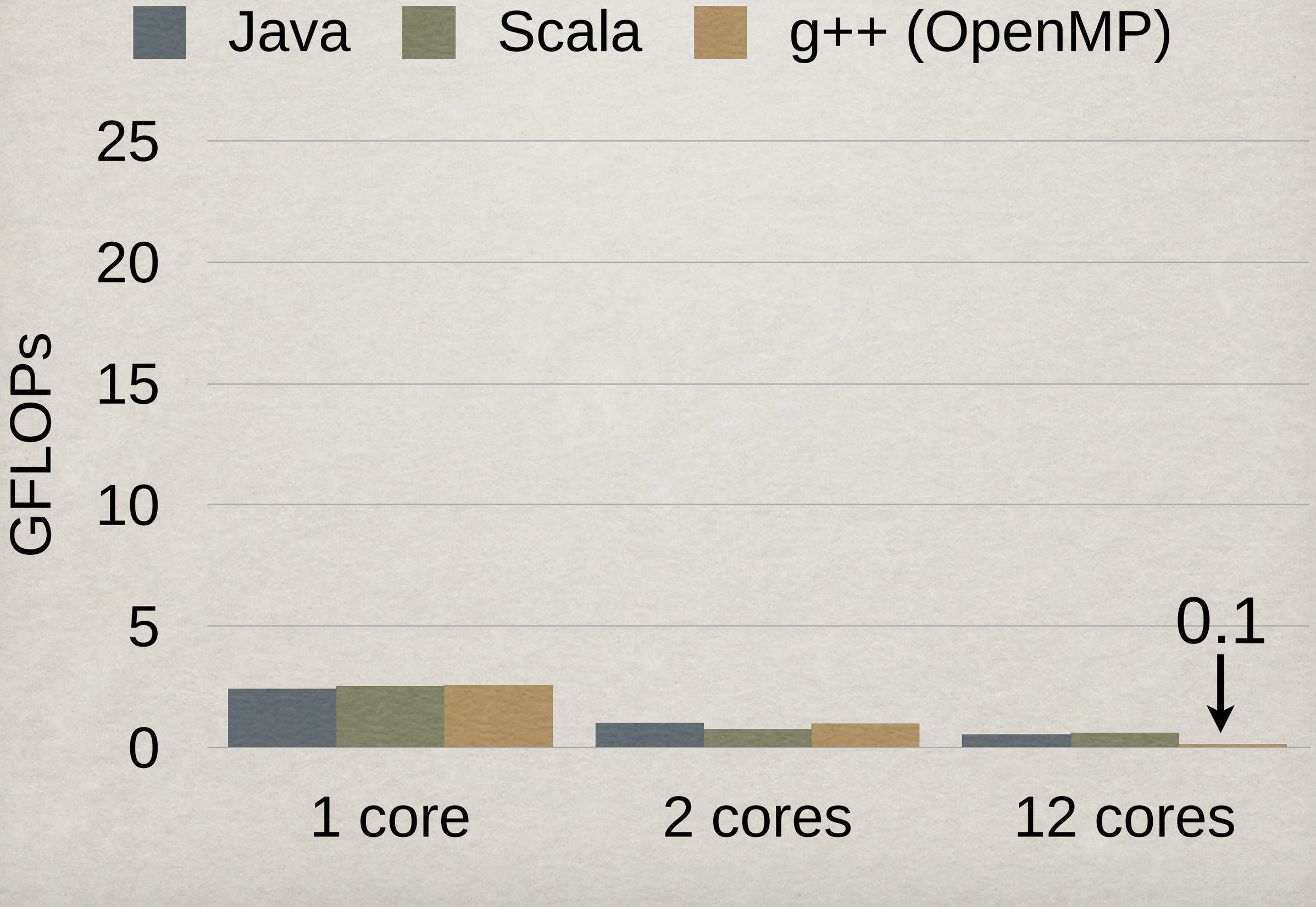
2D: 1000 x 500


Java
Scala
g++ (OpenMP)
GFLOPS
25
20
15
10
5
0
1 core
2 cores
12 cores

2D: 1000 x 50


Java
Scala
g++ (OpenMP)
GFLOPS
25
20
15
10
5
0
1 core
2 cores
12 cores

2D: 1000 x 5


Java
Scala
g++ (OpenMP)
GFLOPS
25
20
15
10
5
0
0.1
1 core
2 cores
12 cores

3D arrays (C/C++)

```
void solr3(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2, int n3,
           float*** x, float*** y) {
  for (int i3=0; i3<n3; ++i3)
    solr2(a1,a2,b0,b1,b2,n1,n2,x[i3],y[i3]);
}
```

3D arrays (C/C++)

```
void solr3(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2, int n3,
           float*** x, float*** y) {
  for (int i3=0; i3<n3; ++i3)
    solr2(a1,a2,b0,b1,b2,n1,n2,x[i3],y[i3]);
}
```

Is `solr2` parallel?

3D arrays (C/C++)

```
void solr3(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2, int n3,
           float*** x, float*** y) {
  for (int i3=0; i3<n3; ++i3)
    solr2(a1,a2,b0,b1,b2,n1,n2,x[i3],y[i3]);
}
```

Should we make this loop over `i3` parallel?

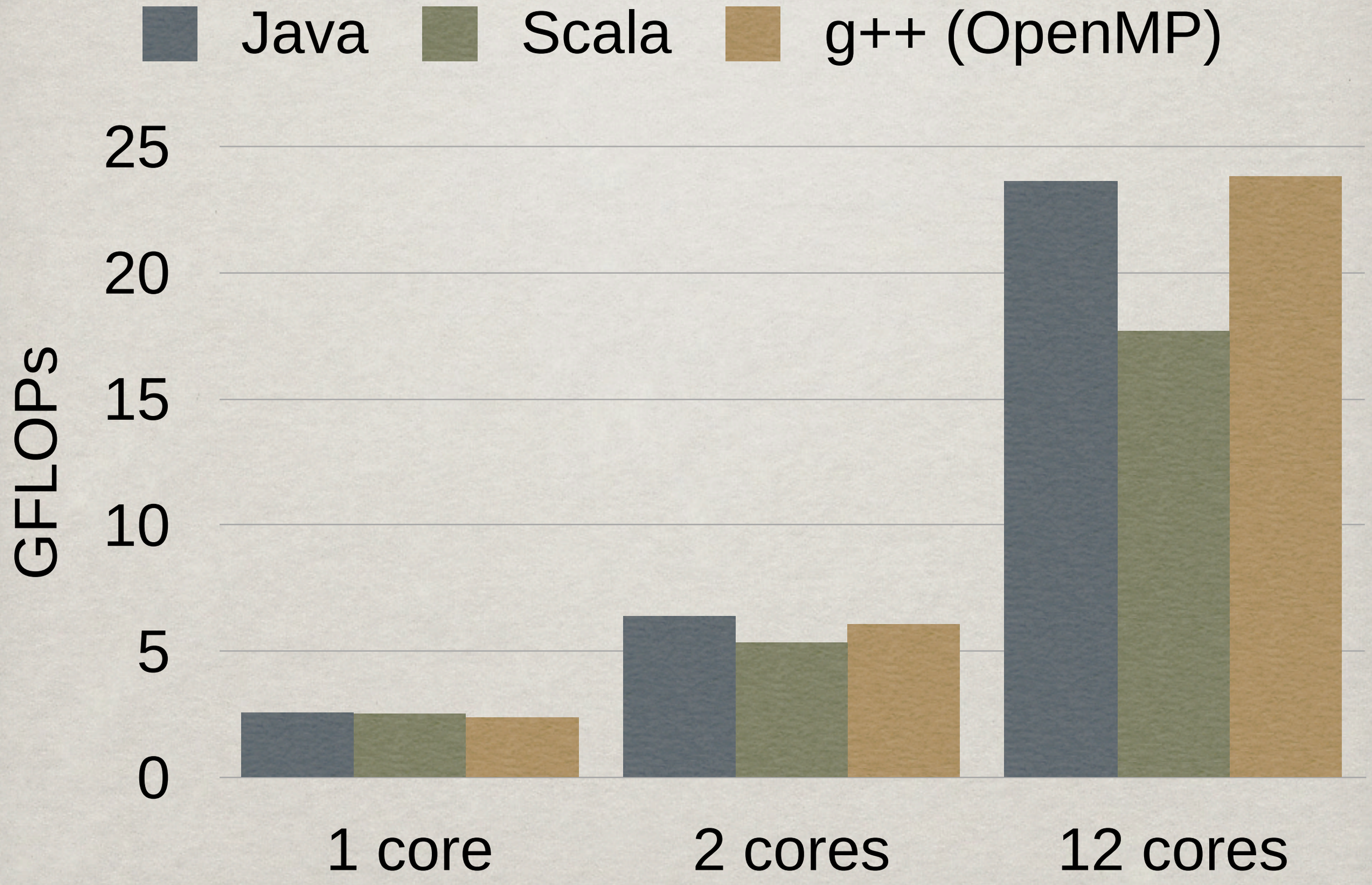
3D arrays (C/C++)

```
void solr3(float a1, float a2,
           float b0, float b1, float b2,
           int n1, int n2, int n3,
           float*** x, float*** y) {
  #pragma omp parallel for schedule(dynamic)
  for (int i3=0; i3<n3; ++i3)
    solr2(a1,a2,b0,b1,b2,n1,n2,x[i3],y[i3]);
}
```

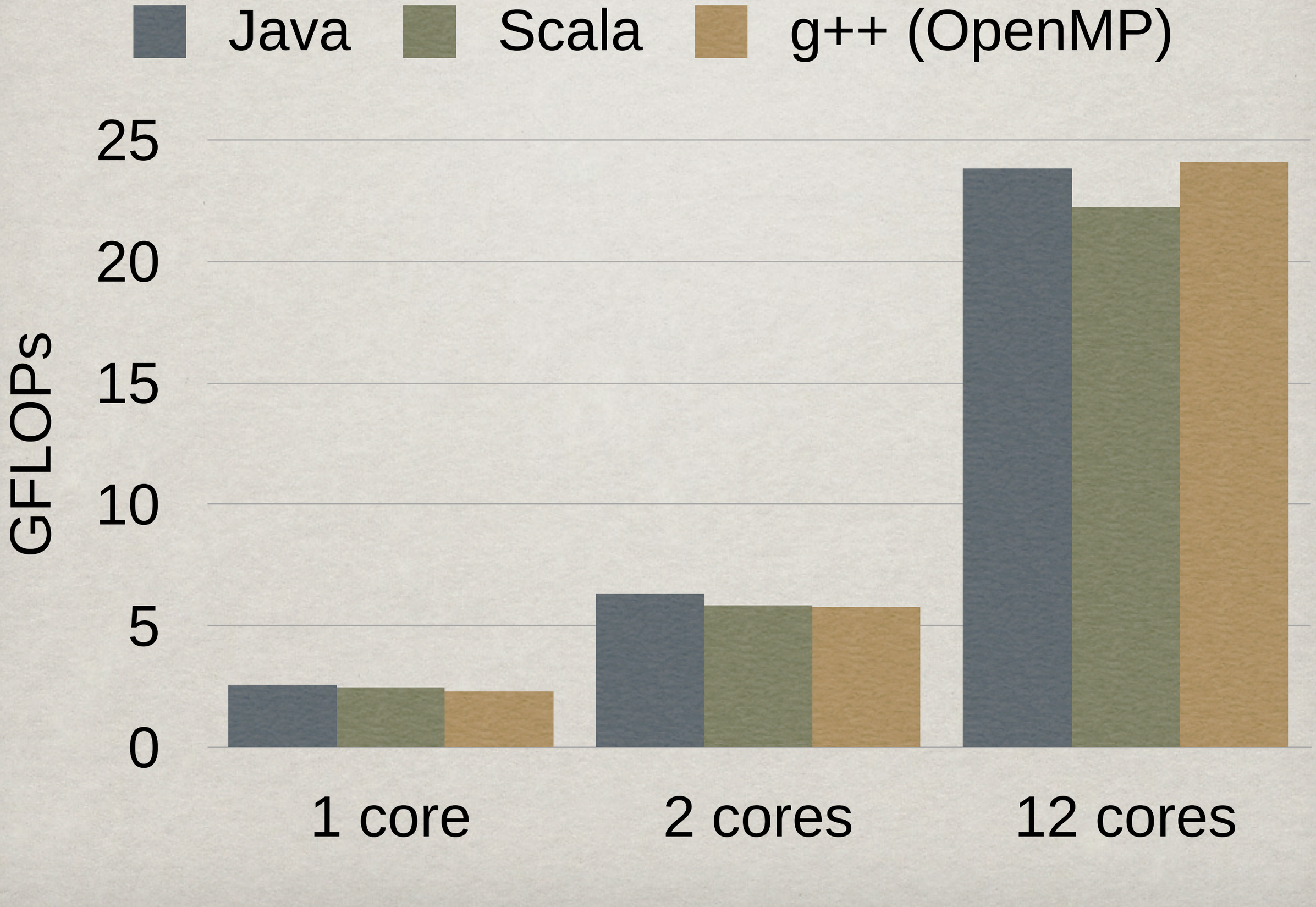
OpenMP

Any nested loops
will now be serial,
if using OpenMP.

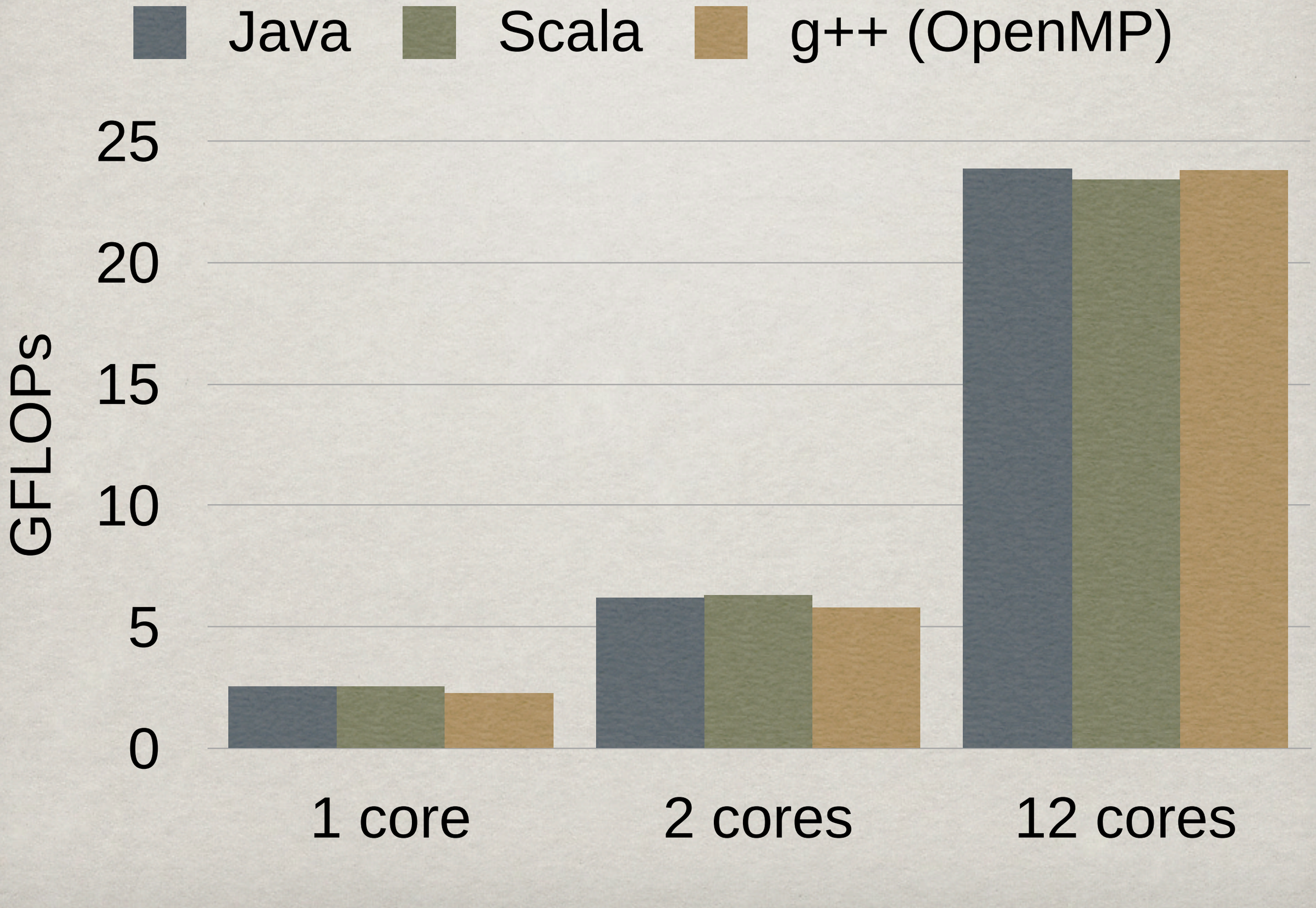
3D: 1000 x 5 x 50000



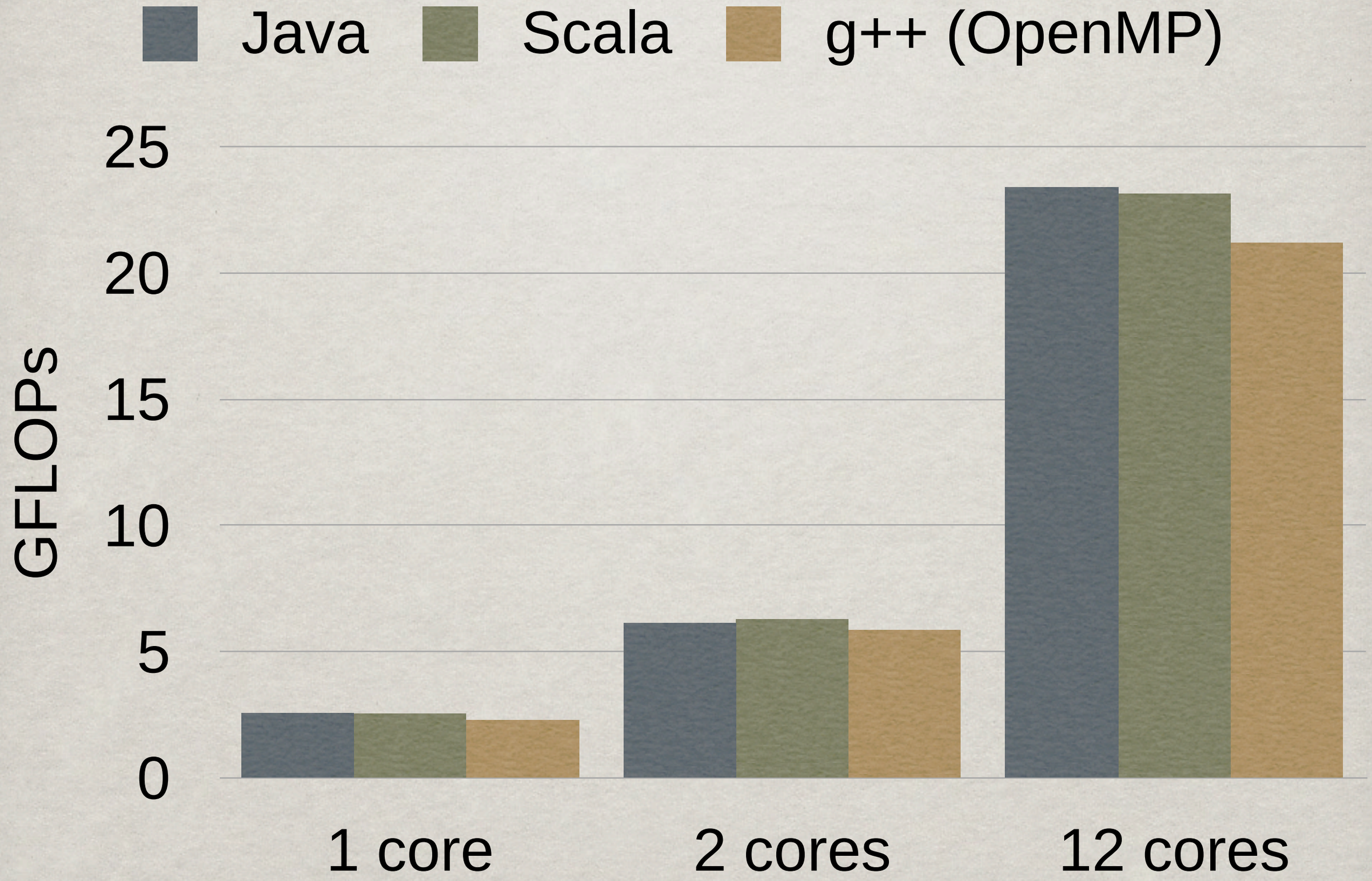
3D: 1000 x 50 x 5000



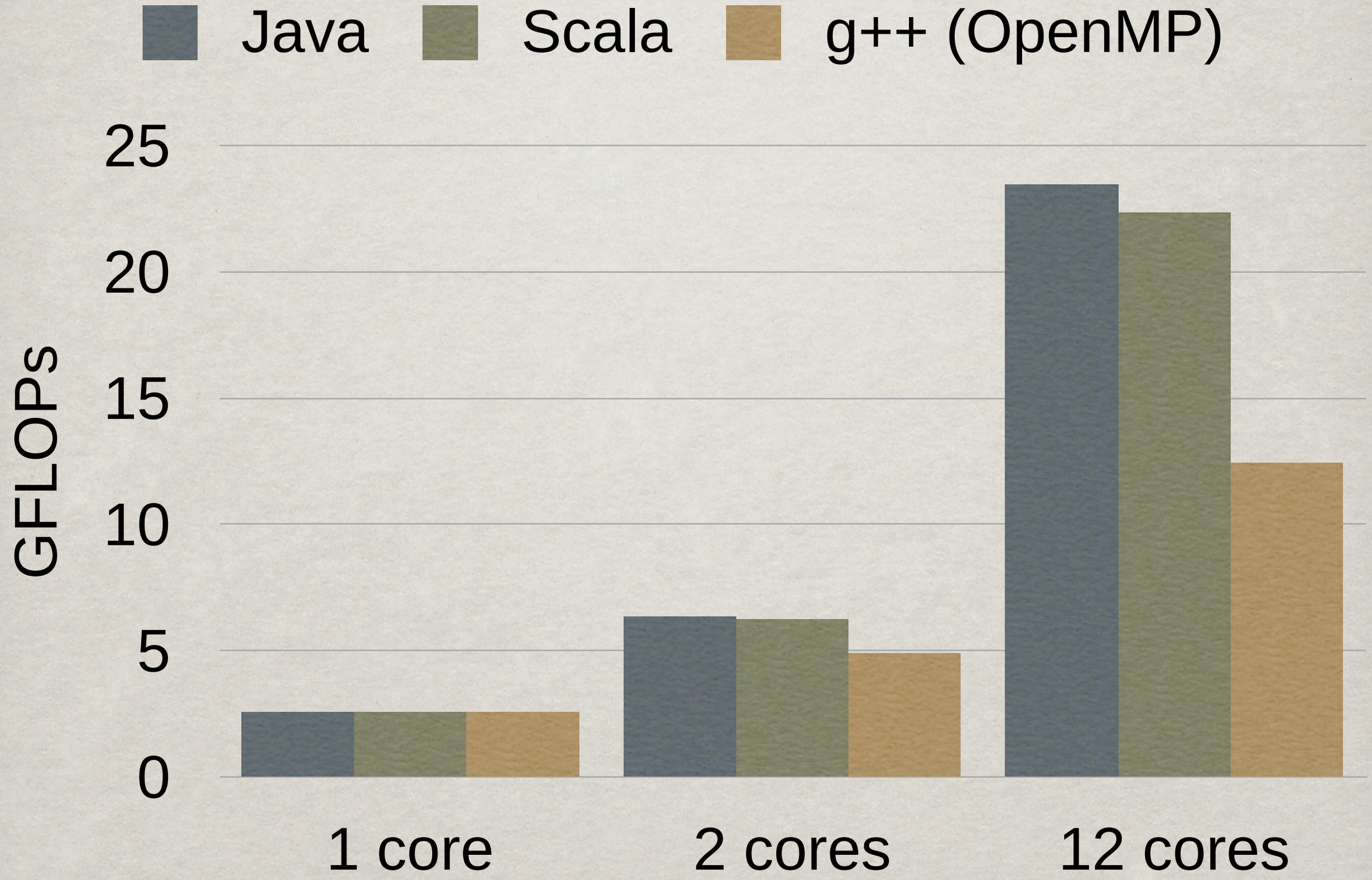
3D: 1000 x 500 x 500



3D: 1000 x 5000 x 50



3D: 1000 x 50000 x 5



Multicore computing in software libraries

Libraries *must* use standard frameworks

Multicore computing in software libraries

C: OpenMP

Multicore computing in software libraries

C++: Intel's TBB (Threading Building Blocks, which use fork-join)

Multicore computing in software libraries

Java: fork-join

Multicore computing in software libraries

Scala: parallel arrays (which use fork-join)