K-Memes

Bowman, Rosenberger, Sybrandt

Optimizing K Means

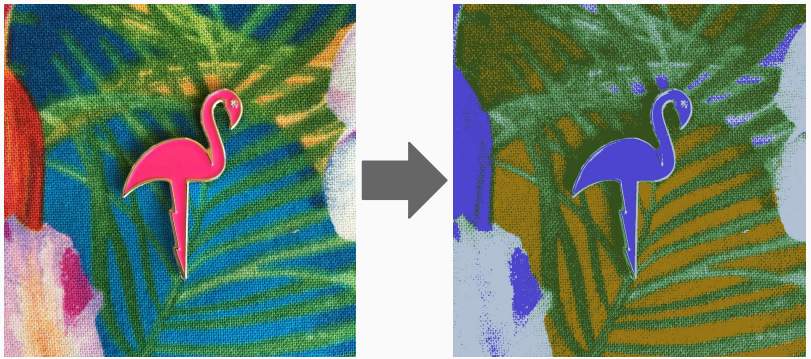
- Single Thread with and without SIMD
- OpenMP
- C++ 11 Parallel Programming Library
- OpenCL & OpenCL w/ Improved Kernel

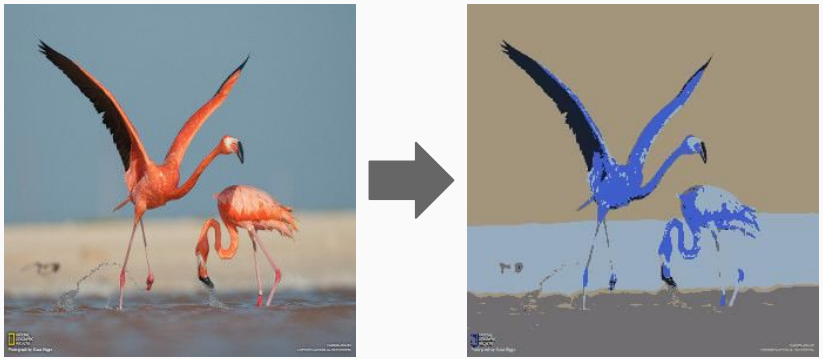
Data Sets and Testing

- Algorithm runs until centroids converge
- Input images
  - 300 x 300
  - 1000 x 1000
- StopWatch Class
- Build configuration
  - X64 - Release build
  - GCC Laptop, High Performance, plugged in

Image Results

- Utilized 5 Centroids
- Stopping execution when centroid converge within threshold





Single Threaded

- Baseline comparison for other results
- Some fluctuation in runtimes
  - Possibly due to Intel Speed Step
  - Possibly due to rand Centroid pos
- Convergence changes:

```
do{
      OLDcentroids = centroids;
      assignCentroids(pixels, centroids);
      moveCentroids(pixels, centroids);
}while(!convergence(OLDcentroids,centroids));
```

Single Threaded Optimizations

- The Pinnacle of Single Threadedness™
- Optimizations
  - Loop unrolling centroid assign and move loop
  - Bit shift instead of multiply
  - Inlined several functions
  - Shortened function names: assignCentroids() -> ac()
  - Reducing redundant index calculation where possible
  - SSE Intrinsics for centroid pixel distance

Single Threaded Optimizations

- SIMD SSE Intrinsics
  - Used for centroid/pixel distance calculation during assignment

```
inline static __m128 sseCentroidPixelDist(pixel *a,  centroid *b){
	__m128 mA = _mm_load_ps(reinterpret_cast<float*>(a));
	__m128 mB = _mm_load_ps(reinterpret_cast<float*>(b));

	__m128 res = _mm_sub_ps(mB, mA);//centroid.x,y,z - pixel.x,y,z
	__m128 res2 = _mm_mul_ps(res,res);//square results
	return res2;
}
```

Single Threaded Optimizations

- SIMD SSE Intrinsics
  - Used for centroid/pixel distance calculation during assignment

```
for (int i = 0;i<pixels.size();i++){
	int minIndex;
	float minVal = 255<<2;
	for (int j = 0;j<centroids.size();j++){
		//calc centroid pixel distance
		float *res = reinterpret_cast<float*>(&sseCentroidPixelDist(&pixels[i], &centroids[j]));
		//run sqrt to get final distance
		float diff = sqrt(res[0] + res[1] + res[2]);
		if (diff < minVal){
			minIndex = j;
			minVal = diff;}
	}
	pixels[i].cluster = minIndex;
}
```

OpenMP and C++11

Used same optimizations as the Pinnacle of Single Threadedness™

Attempted getting rid of arguments in function calls by making them members of a class, which was unsuccessful due to slower memory calls

Having centroids and points as a class member variable didn’t work for C++11’s parallel library

OpenMP just required adding “#pragma omp parallel for” to the appropriate lines

OpenMP and C++11

Tested with images of sizes 300x300 and 1000x1000

Found that unrolling to a power of 2 saves a lot of time

Being connected to power, and high performance mode, decreased the running time

Open-CL Naive

We already talked about it!

Only labeling occurs on the GPU

Was a good idea in 2002, generally regarded as bad nowadays

Open-CL “Improved”

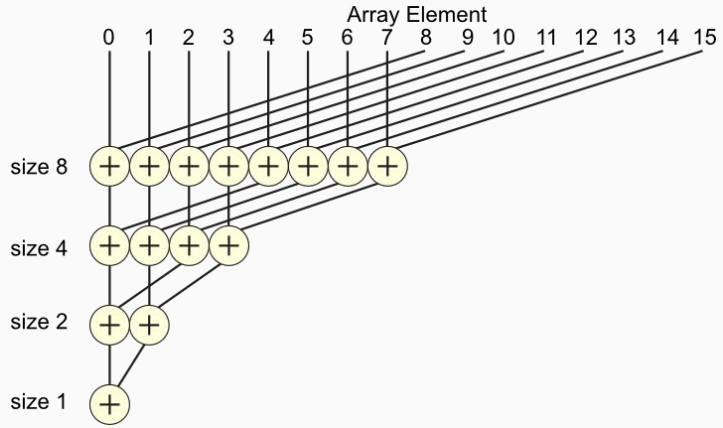
Pixel data is loaded once to the GPU

K-Means is computed entirely on the card

Kernel arguments:

```
__kernel void Kmeans(
      __global float3* pixels,
      __global float4* centroidScratch,
      __global float3* output,
      __global float3* GLOBALcentroids
)
```

K-Means “Improved”



centroidScratch

- Allows the GPU to move centroids in parallel
- Each pixel has its own record of each centroid
- Threads reduce centroid records to accurate values

K-Means “Improved”

Move Centroid w/ Reduction:

Each pixel fills out initial values:

```
//prepare scratch
for(i=0;i<CENTROID_COUNT;i++){
	centroidScratch[id*CENTROID_COUNT+i] = (float4)(0.0f,0.0f,0.0f,0.0f);
}
centroidScratch[id*CENTROID_COUNT+label] = (float4)(currPix.x,currPix.y,currPix.z,1.0f);
```

K-Means “Improved”

Move Centroid w/ Reduction:

Pixel values are summed and reduced:

```
for(halfSize=numThreads/2;halfSize>0;halfSize>>=1){
	if(id<halfSize){
		for(i=0;i<CENTROID_COUNT;i++){
			centroidScratch[id*CENTROID_COUNT+i] += centroidScratch[(id+halfSize)*CENTROID_COUNT+i];
		}
	}
}
```

K-Means “Improved”

Move Centroid w/ Reduction:

Global centroid values are calculated:

```
if(id<CENTROID_COUNT){
	temp = centroidScratch[id];
	temp.x /= temp.w;
	temp.y /= temp.w;
	temp.z /= temp.w;
	centroidScratch[id] = temp;
}
```

K-Means “Improved”

Move Centroid w/ Reduction:

Global centroid values are copied to local:

```
if(lid<CENTROID_COUNT){
	temp = centroidScratch[lid];
	centroids[lid] = (float3)(temp.x,temp.y,temp.z);
}
```

K-Means “Improved”

Convergence Check

- Each local thread group runs this
- After copying global centroids to local

```
bool conv(__local float3* c1, __local float3*c2){
	float acc = 0;
	int i;
	for(i=0;i<5;i++){
		acc += fabs(c1[i].x-c2[i].x);
		acc += fabs(c1[i].y-c2[i].y);
		acc += fabs(c1[i].z-c2[i].z);
	}
	return acc < 0.000001;
}
```

K-Means “Improved”

Pros and Cons:

- Pro: Data is sent to the GPU only once
- Con: The GPU is occasionally very underutilized
  - Ex: reduction and convergence

Results

Algorithm	300x300 Time	1000x1000 Time
naive single	135	1604
pinnacle of single	58	1408
omp	123	1475
c++11	260	961
Open-CL naive	74	1265
Open-CL complex	84	1875

Questions?

Nope.