Parallelizing a Beat Detection Algorithm

The Seahawks

The Beat Detection Algorithm

- Named “Sound Energy Algorithm” by Marco Ziccardi
- Divides samples, or frequencies, of a song into blocks
- 1 block = 1,024 samples
- The instant energy of a block j:

$$E_j = \sum_{i=0}^{1023} left[i]^2 + right[i]^2$$

The Beat Detection Algorithm cont.

- 1 second = 43 blocks (= 44,100 total samples)
- Average energy of a window of blocks:

$$avg(E) = \frac{1}{43} \sum_{j=0}^{42} E_j$$

- Energy variance of a window of blocks:

$$var(E) = \frac{1}{43} \sum_{j=0}^{42} (avg(E) - E_j)^2$$

The Beat Detection Algorithm cont. 2

- Linear Regression of the energy variance:

$$C = -0.0000015 \cdot var(E) + 1.5142857$$

A peak is detected if instant energy E_j of a block is greater than $C \cdot avg(E)$

4 consecutive peaks is considered a beat

What We Parallelized

- The summation to calculate the instant energy of a block.

$$E_j = \sum_{i=0}^{1023} left[i]^2 + right[i]^2 \qquad E_j = \sum_{i=0}^{1023} channel[i]^2 * 2$$

```
__global__ void getInstantEnergies(float * frequencies, float * energy,
        int samples) {
    gpuSquared(frequencies, samples);
    __syncthreads();
    gpuCalcInstantEnergies(frequencies, energy);
    __syncthreads();
}
```

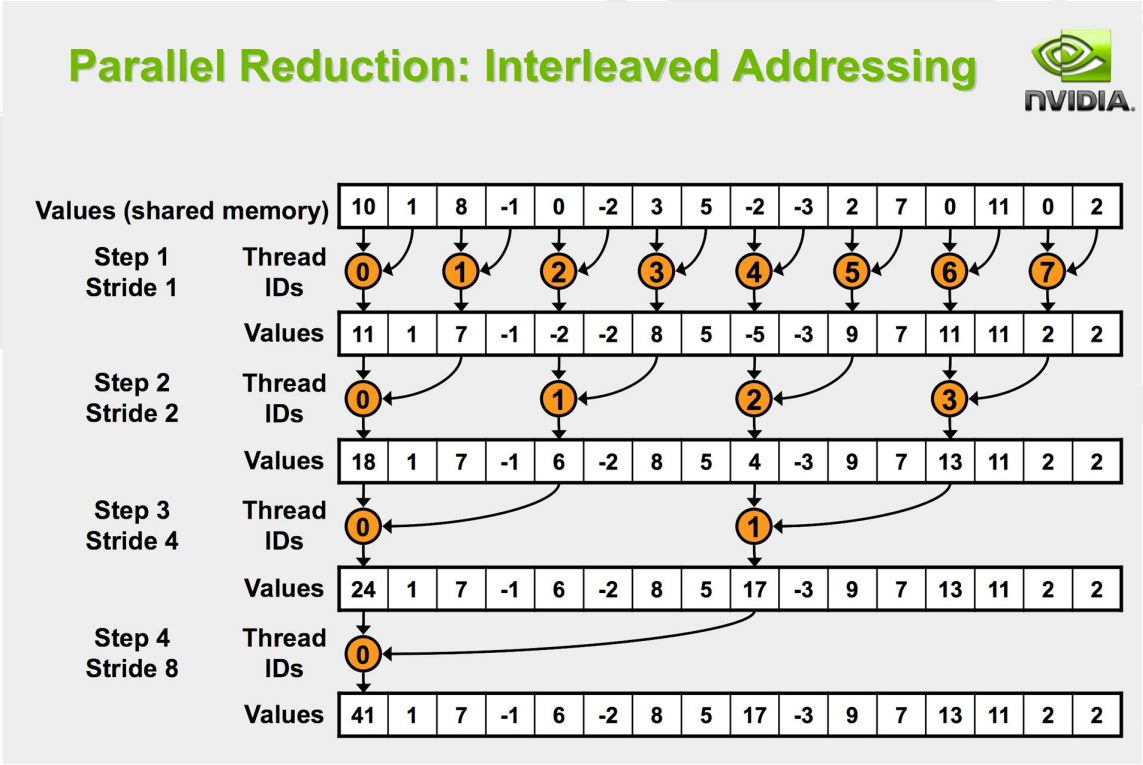
Without Parallelization

```
float * calculateSample(float sampleArray[], long samples){
    long i = samples - SAMPLES_PER_SECOND;

    for(; i < samples; i++){
        sampleArray[i] = (sampleArray[i] * sampleArray[i]) * 2;
    }

    return sampleArray;
}
```

With Parallelization

```
__device__ void gpuSquared(float frequency[], int totalFrequencies) {
    int element = blockIdx.x * blockDim.x + threadIdx.x;
    if (element < totalFrequencies)
        frequency[element] = 2 * (frequency[element] * frequency[element]);
}
```

Reduction Algorithm



Without Parallelization

```
float * getBlocks(float ej[], float sampleArray[], long samplesPerBlock, long samples){
    long i = samples - SAMPLES_PER_SECOND;
    long j = 0;
    float sum = 0.0;

    for(j = 0; j < 43; j++){
        for(; i < samplesPerBlock; i++){
            sum += sampleArray[i];
        }
        samplesPerBlock += SAMPLES_PER_BLOCK;
        ej[j] = sum;
        sum = 0;
    }
    return ej;
}
```

With Parallelization

```
__device__ void gpuCalcInstantEnergies(float frequency[], float instantEnergy[]) {
    unsigned int tid = threadIdx.x;
    unsigned int element = blockIdx.x * blockDim.x + tid;

    //the last 68 samples of a second don't get computed
    unsigned int offset = blockIdx.x / BLOCKS_PER_SECOND;
    offset *= UNCALCULATED_SAMPS;

    instantEnergy[element] = frequency[element];
    __syncthreads();

    for (unsigned int s = 1; s < SAMPLES_PER_BLOCK; s *= 2) {
        if (tid % (2 * s) == 0) {
            instantEnergy[element + offset] += instantEnergy[element + s
                    + offset];
        }
        __syncthreads();
    }

    if (tid == 0) {
        frequency[blockIdx.x] = instantEnergy[element + offset];
    }
}
```

Demonstration

Comparing Software to Our Code


```
STARSHIP (~): nvcc gpubpm.cu
STARSHIP (~): ./a.out techno.txt
Starting
BPM = 135
Time taken 5 seconds 651 milliseconds
STARSHIP (~):
```

Results

- The algorithm does not work for all music due to it assuming the length of beat.

- It is not efficient to use the GPU to detect BPM.