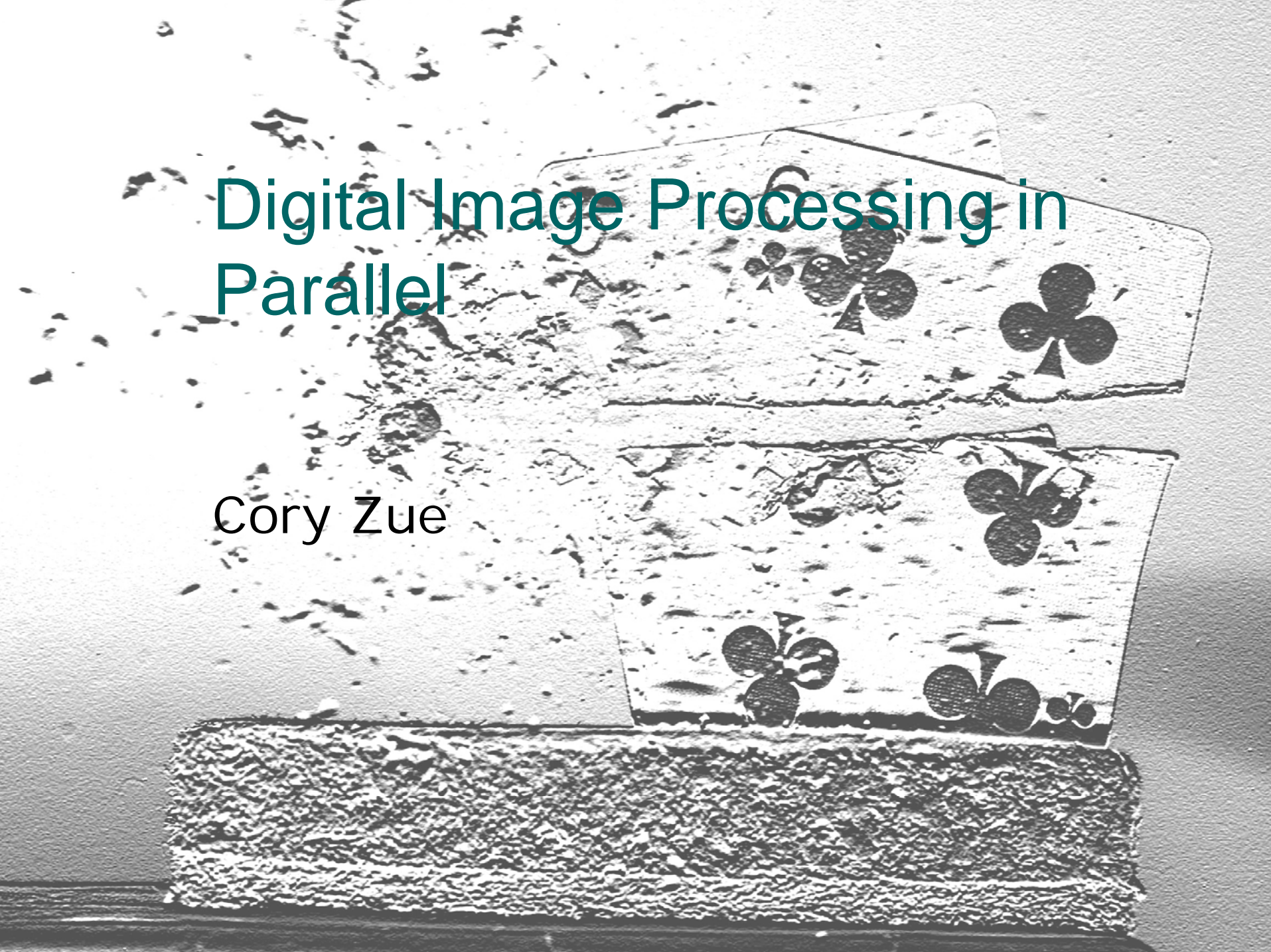


# Digital Image Processing in Parallel

Cory Zue





# What is It?

---

- Apply a filter to an image
- Produce a new image that is better/useful/interesting in some way

## How it works (briefly)

---

- Translate image into a matrix of numbers (i.e. a bitmap)
- Create a convolution filter (different processing effects use different filters)
- Apply filter by locally convolving at each location

$$y[r, c] = \frac{1}{\sum_{i,j} h[i, j]} \sum_{j=0}^{M-1} \sum_{i=0}^{M-1} h[j, i] g x[r - j, c - i]$$



# Why do it in parallel?

---

- Images are getting large
  - Satellite pictures taken of the entire earth on a daily basis
  - Many applications require fast, real-time processing
- It adapts well to parallel
  - Images are basically matrices of bits
  - Filtering usually only depends on your neighbors

# Some Filters

---

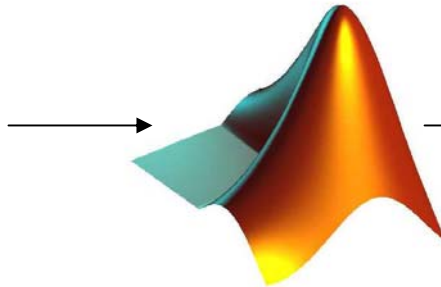
$$\frac{1}{9} \left\{ \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right\}$$

- Average (blurring) filter

$$\left\{ \begin{array}{ccc} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{array} \right\}$$

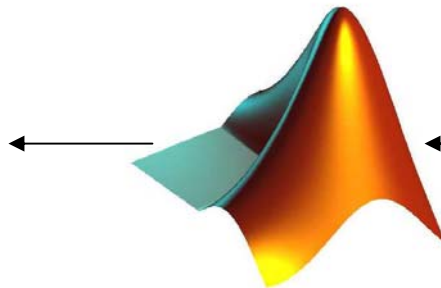
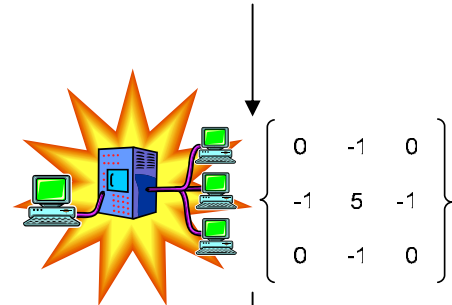
- Sharpening Filter

# Implementation



```

156 159 158 155 158 156 159 158 157 158 158 159
160 164 160 165 163 161 163 161 164 165 162 161
167 165 165 164 170 166 167 167 170 168 169 166
171 169 171 170 169 173 173 172 171 173 176 174
181 177 180 178 179 180 180 180 178 184 181 182
185 184 181 181 183 184 183 184 183 186 186 187
186 182 184 187 185 188 185 185 186 184 184 182
185 183 180 181 182 181 185 182 186 182 181 183
180 181 178 178 180 179 180 177 179 180 178 176
176 172 172 172 176 174 177 173 173 169 172 169
168 168 169 166 169 165 166 166 166 164 166 169
165 166 166 164 163 164 159 164 164 158 162 162
160 155 155 154 155 153 156 155 158 156 155 155
    
```



```

156 159 158 155 158 156 159 158 157 158 158 159
160 164 160 165 163 161 163 161 164 165 162 161
167 165 165 164 170 166 167 167 170 168 169 166
171 169 171 170 169 173 173 172 171 173 176 174
181 177 180 178 179 180 180 180 178 184 181 182
185 184 181 181 183 184 183 184 183 186 186 187
186 182 184 187 185 188 185 185 186 184 184 182
185 183 180 181 182 181 185 182 186 182 181 183
180 181 178 178 180 179 180 177 179 180 178 176
176 172 172 172 176 174 177 173 173 169 172 169
168 168 169 166 169 165 166 166 166 164 166 169
165 166 166 164 163 164 159 164 164 158 162 162
160 155 155 154 155 153 156 155 158 156 155 155
    
```



# Performance Experiments

---

- 2 Images
  - 1 small (256x256)
  - 1 large (2400x3200)
- 4 Filter Sizes
  - 3x3
  - 5x5
  - 15x15
  - 25x25



# But First Some Pictures...

---











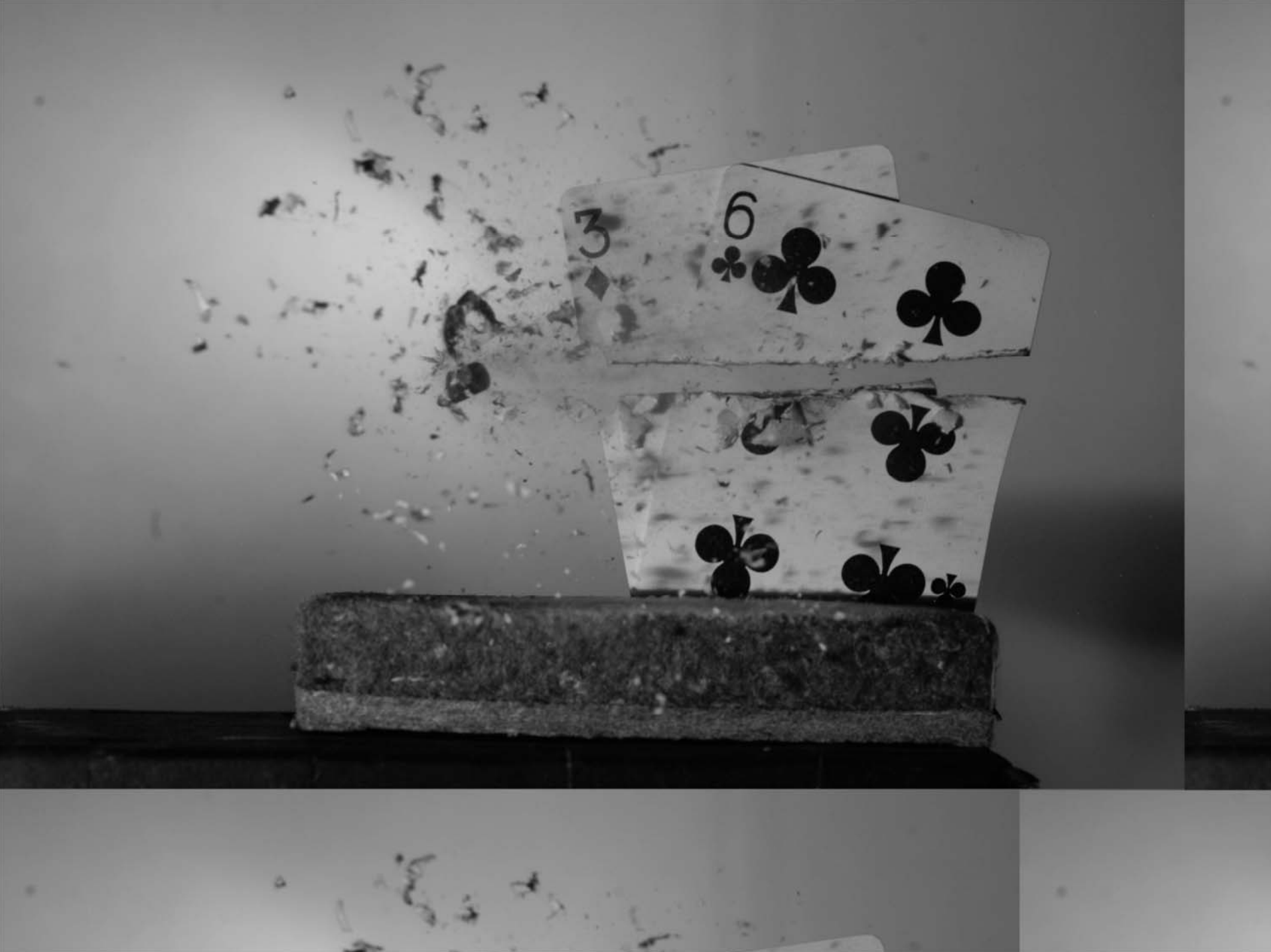




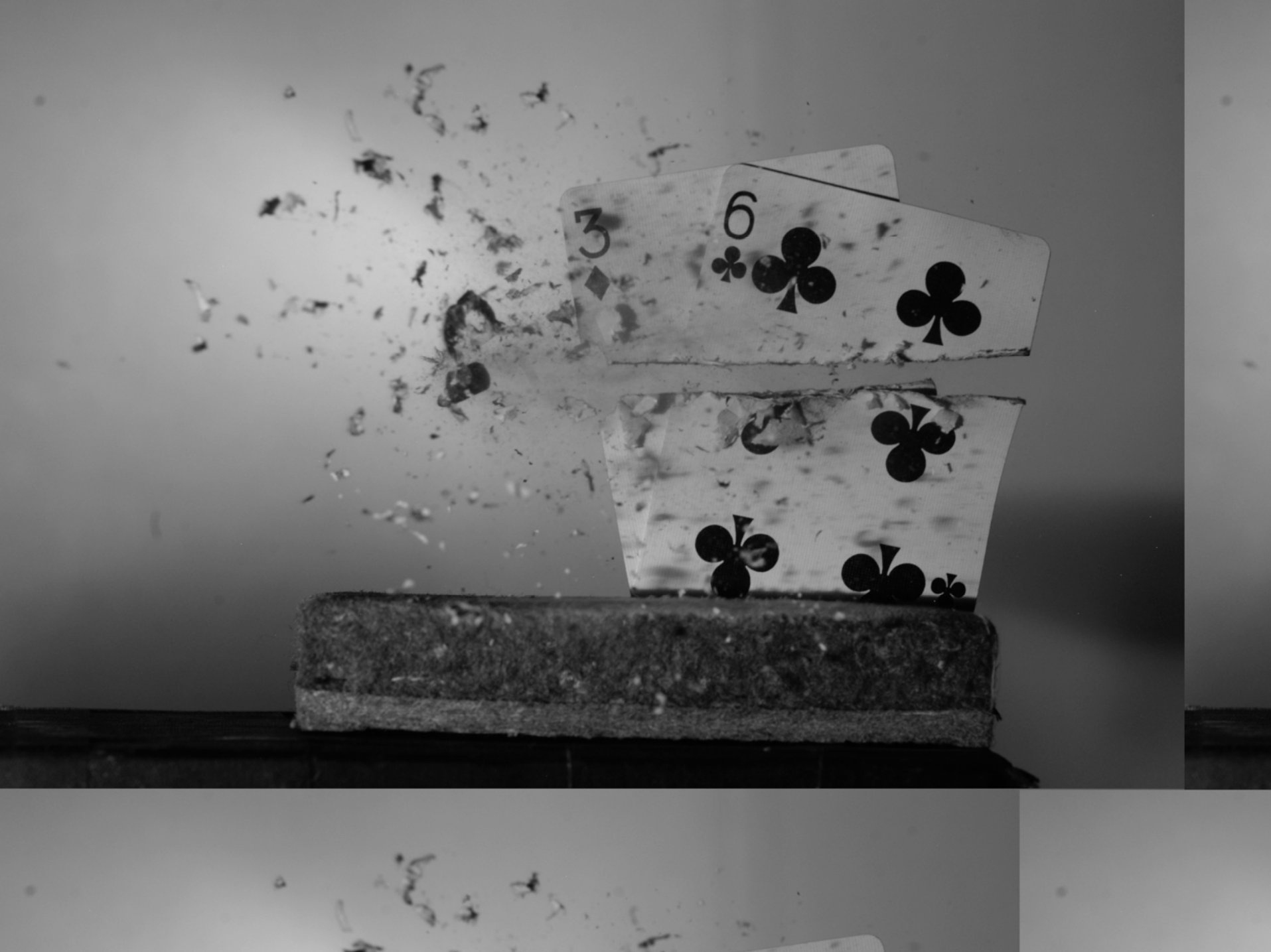


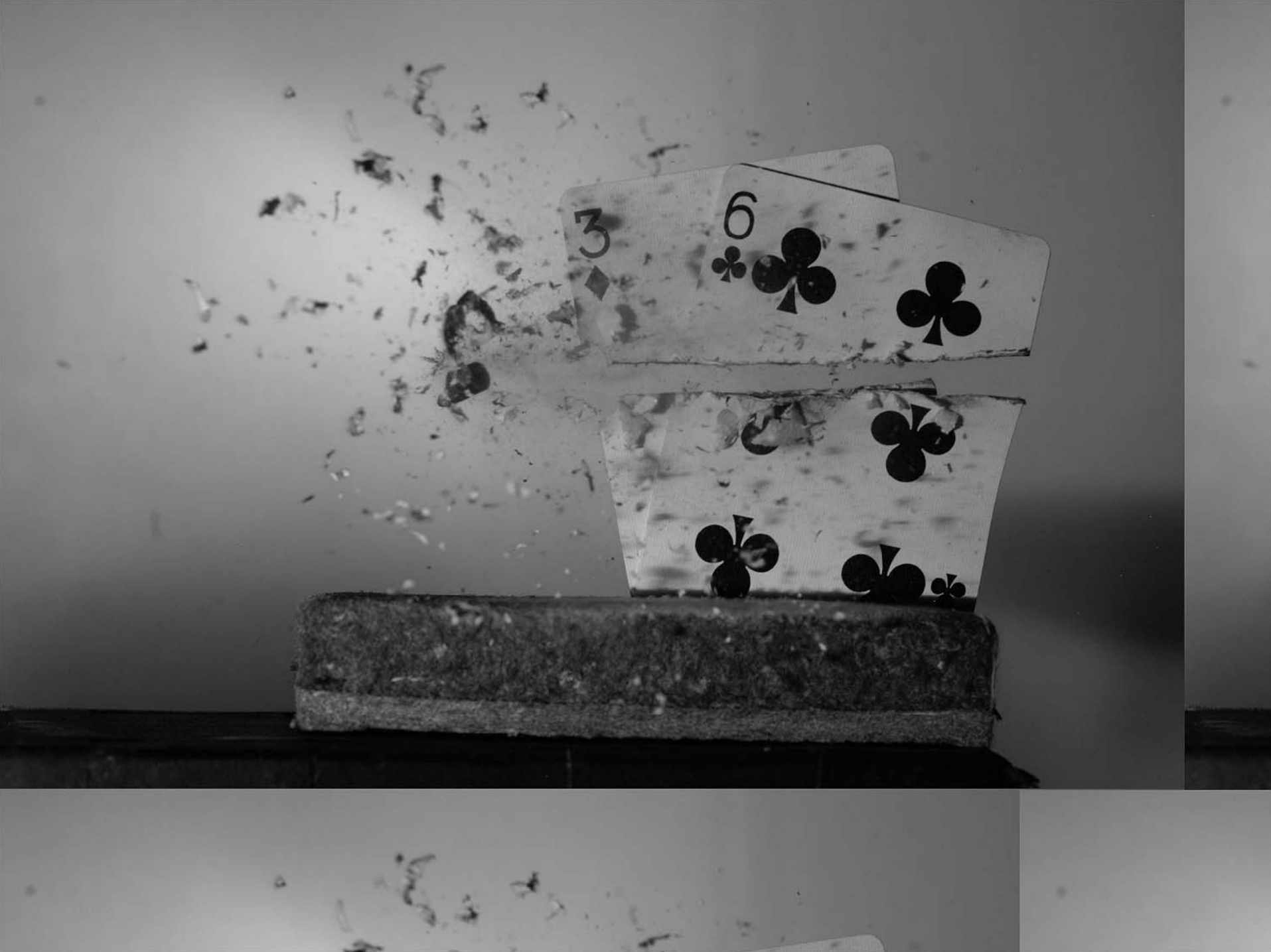


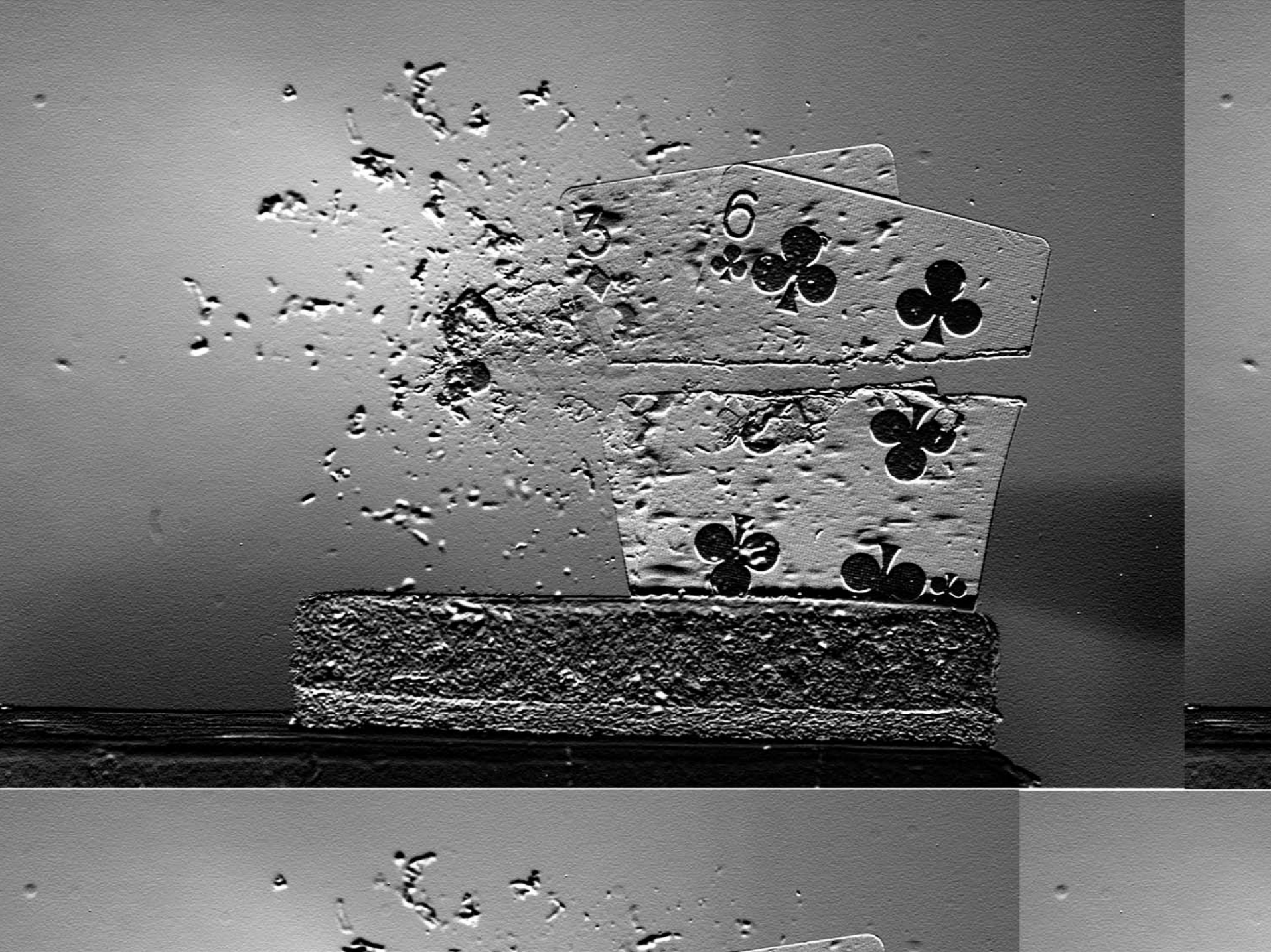










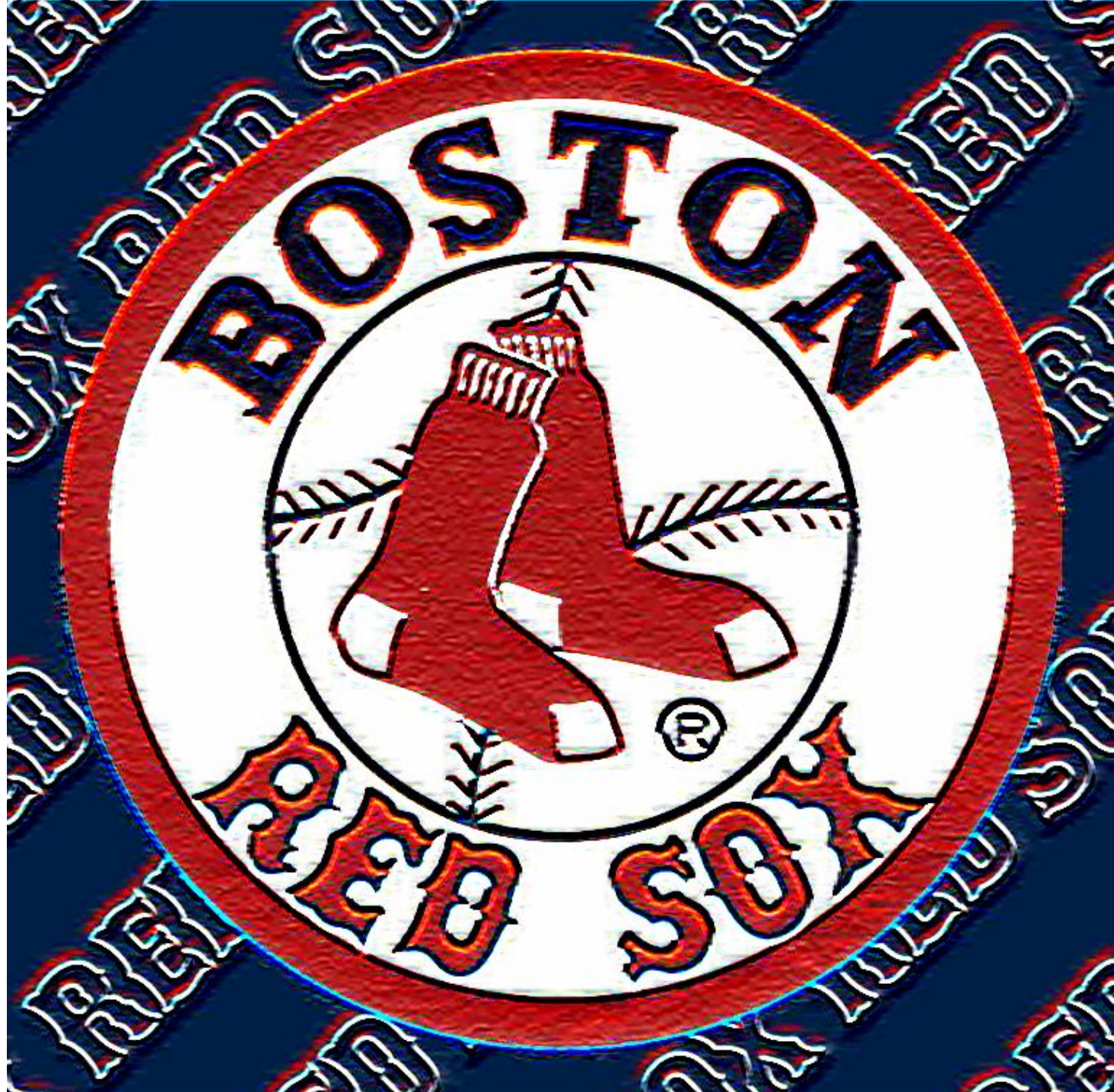












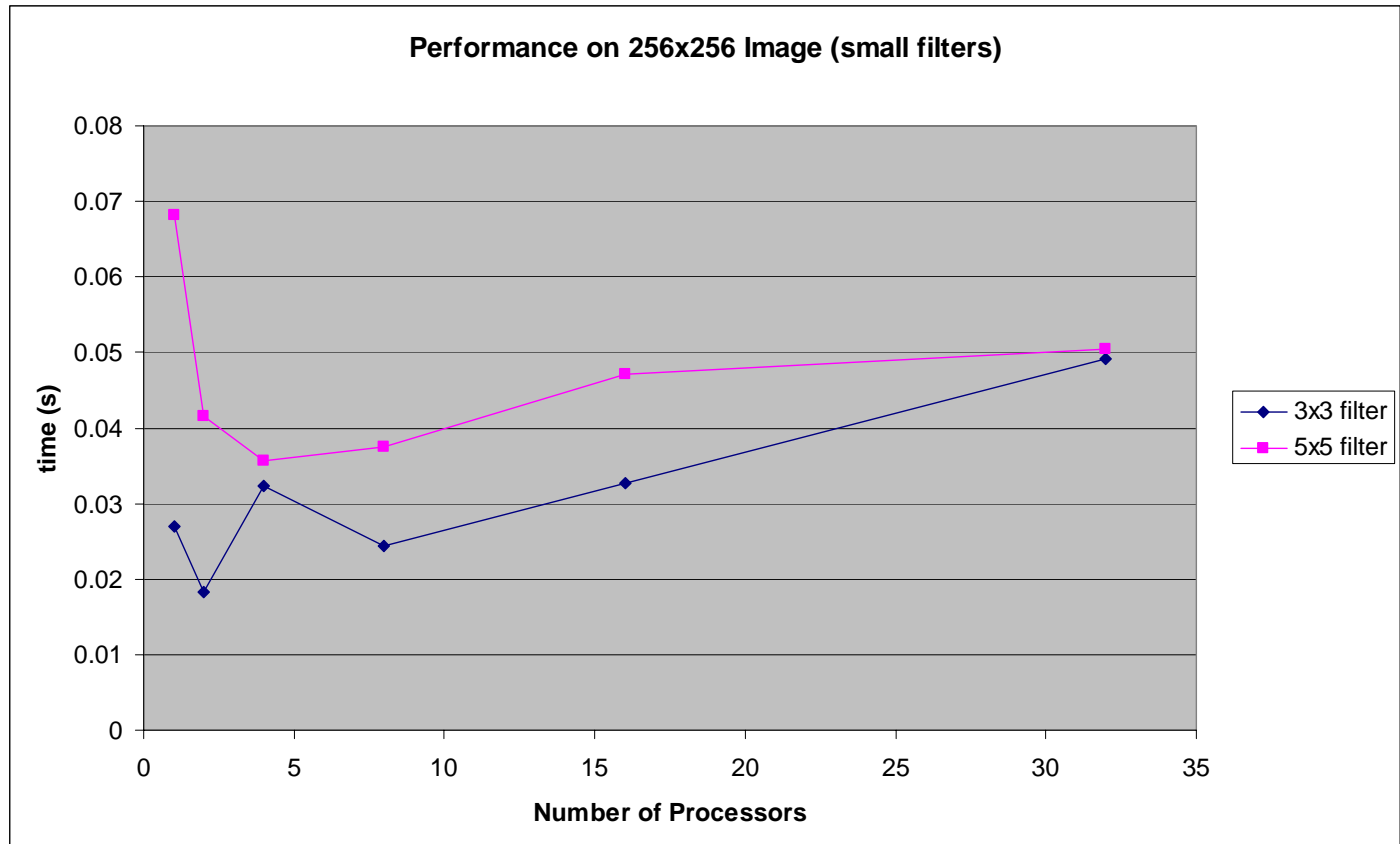


# How big is the problem?

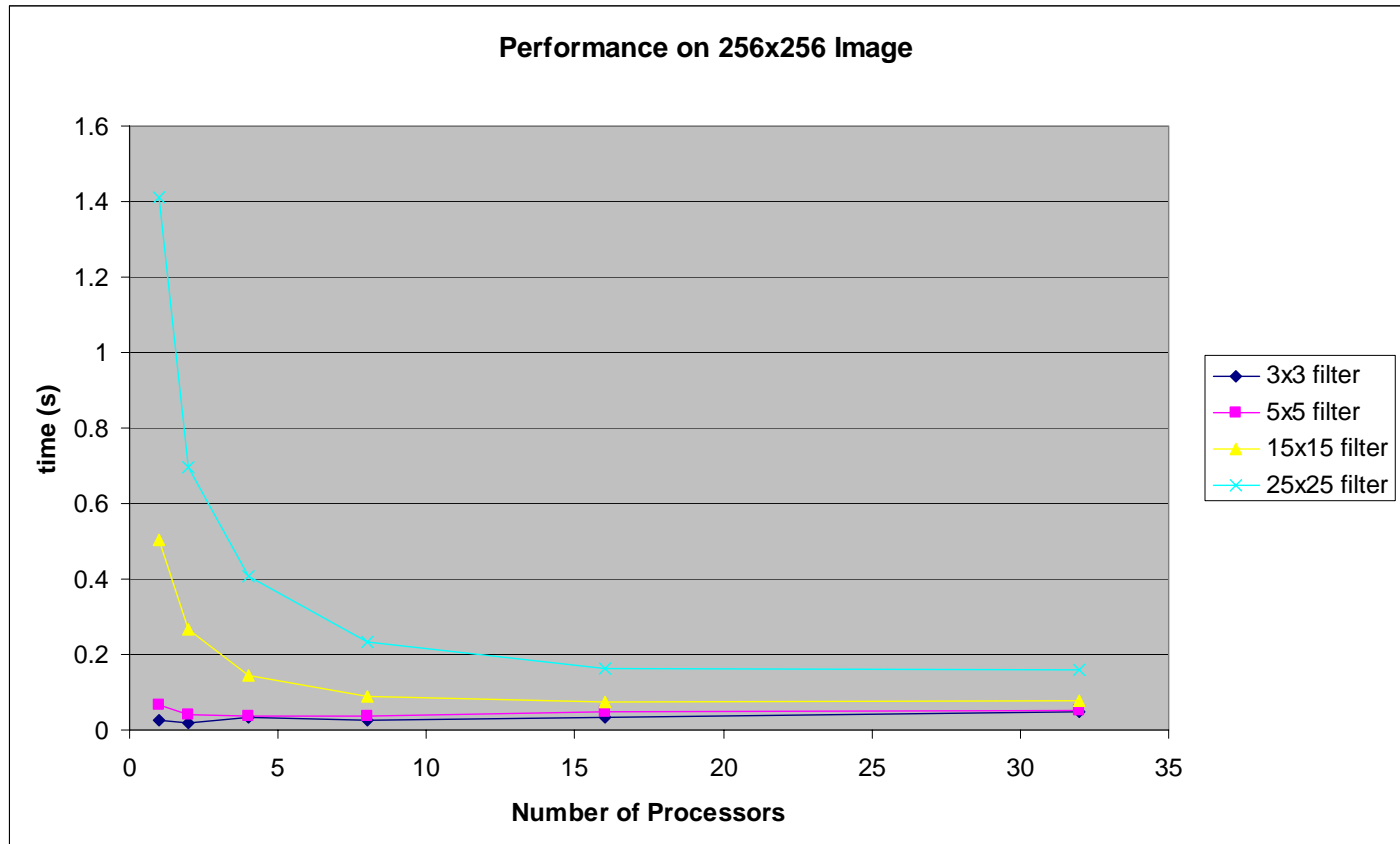
---

- With an  $m \times n$  image and  $f \times f$  filter (with  $m, n \gg f$ )
- Memory usage is  $O(m \cdot n)$ 
  - Parallelism reduces to  $\sim (m \cdot n) / np$  for each processor
- Runtime is  $O(m \cdot n \cdot f^2)$ 
  - Parallelism reduces to  $\sim (m \cdot n \cdot f^2) / np$

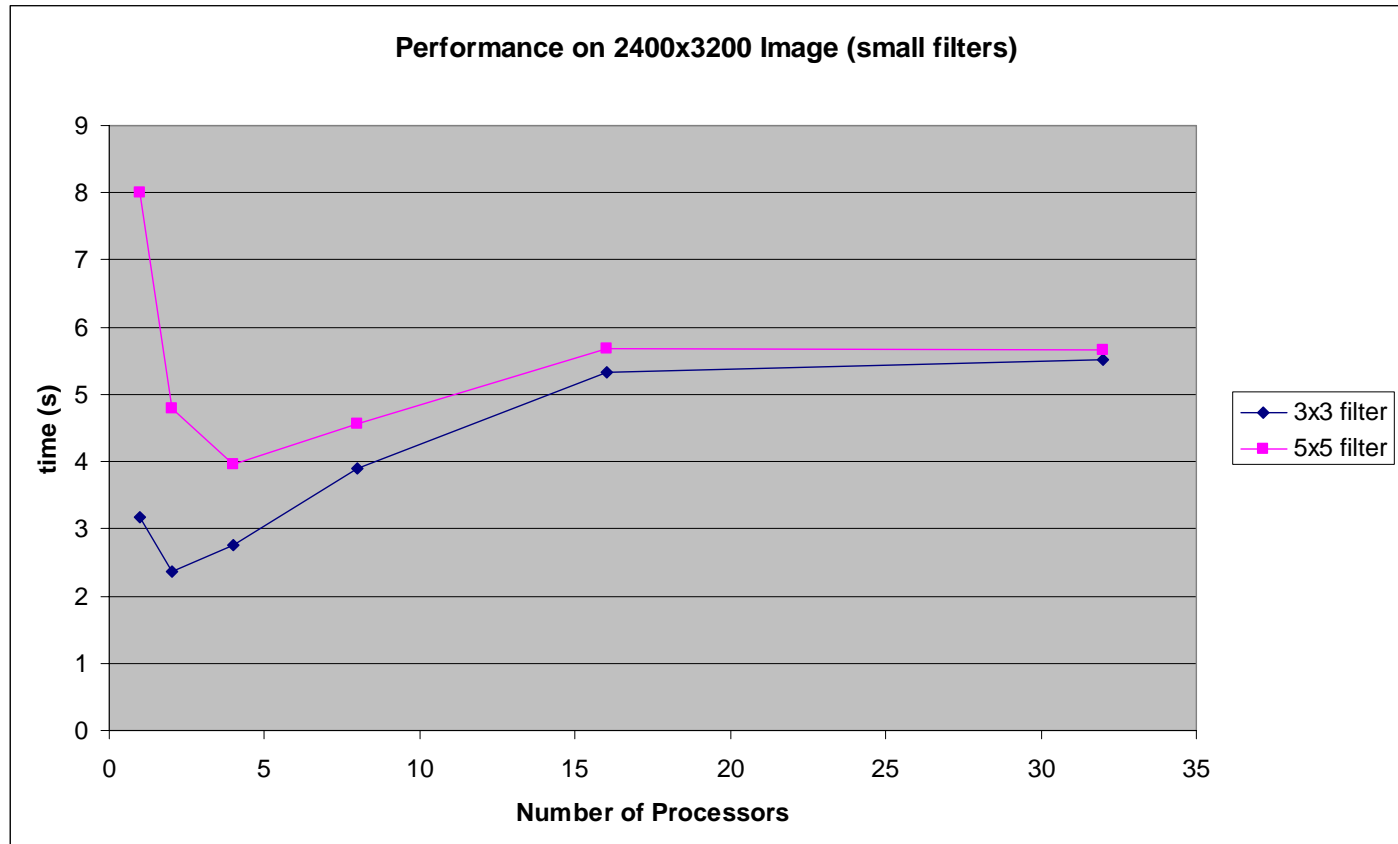
# Performance on the Small Image



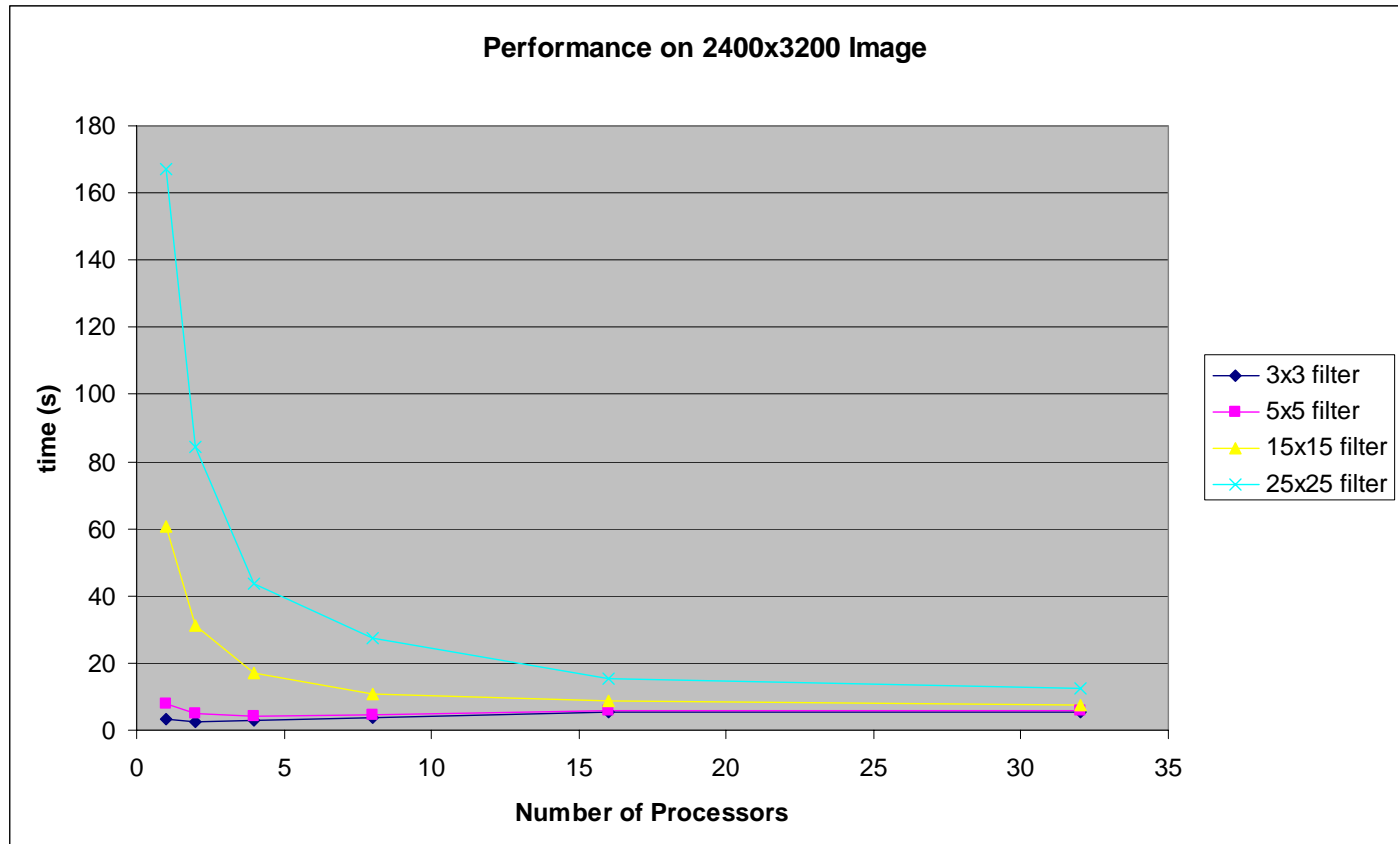
# Performance on the Small Image



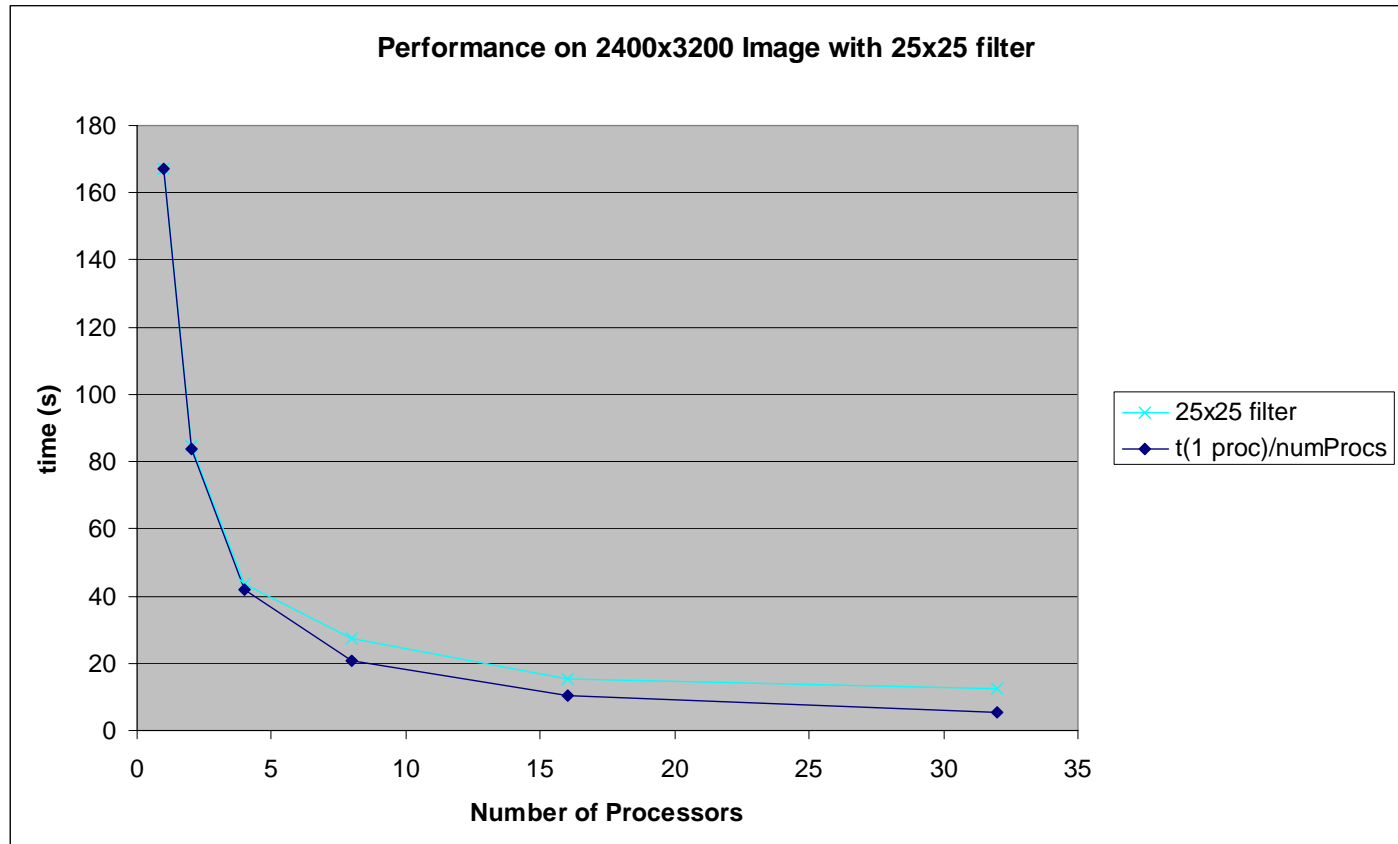
# Performance on the Large Image



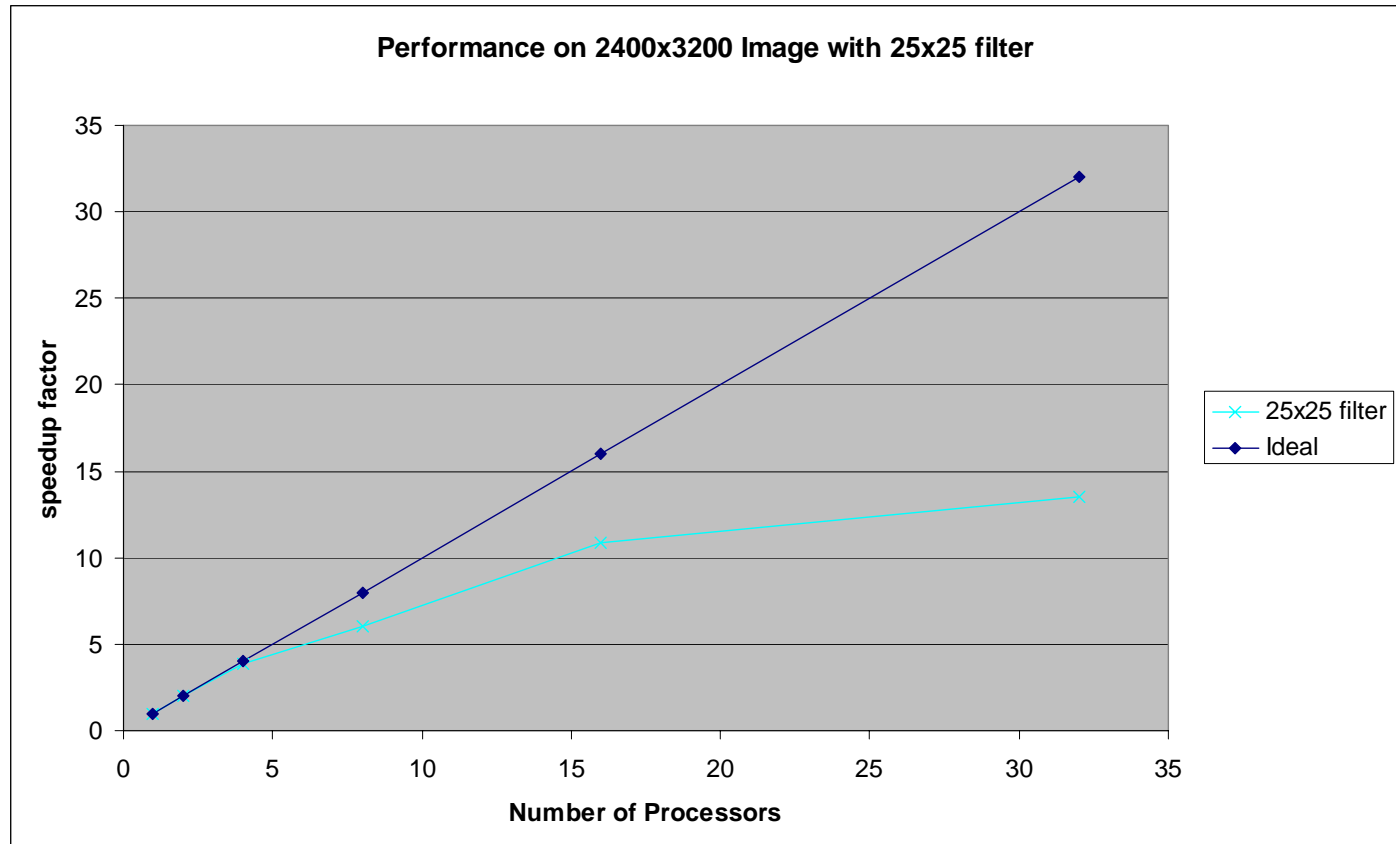
# Performance on the Large Image



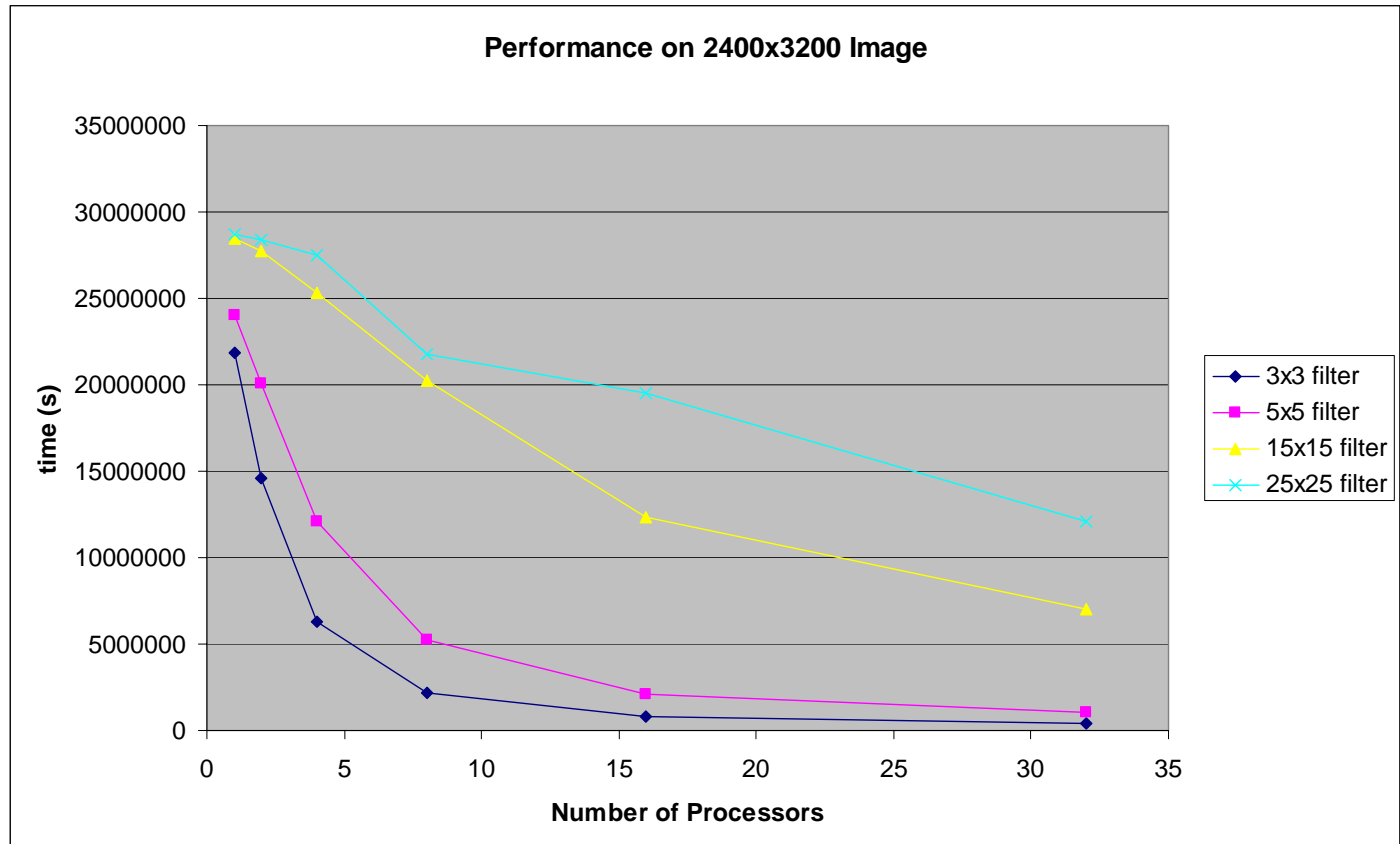
# How close to “perfect”?



# How close to “perfect”?



# Another Measure of Efficiency







# Conclusions

---

- Image processing can greatly benefit from parallelism
- Larger problems (especially filters) result in larger improvements
- Diminishing returns with additional processors



# Questions?

---