

Project Name: Image Histogram by barraCUDA team

Team Members: Vincent Vilda, Jacob Osorio

Demo and Presentation Video: <https://youtu.be/7NwBPWKMiY0>

<https://youtube.com/shorts/jcrfhiBUHvQ?si=eBhAU3TMIZPtjviE>

Project Idea / Overview

The main idea of this project is to perform image histogram calculations using a GPU in order to increase performance. Four different methods of calculating the histogram are performed. Histogram on the CPU is conducted by using an OpenCV histogram function, while three different kernels will be utilized on the GPU with an incrementing method of optimization. The elapsed times are taken for comparison for each of the four methods.

Project Description: The goal of this project is to develop an image histogram computation using CUDA and compare its performance to a similar implementation using OpenCV. The histogram of an image is a graphical representation of the tonal distribution in a digital image. By utilizing GPU acceleration through CUDA, we aim to significantly speed up the computation process compared to OpenCV histogram function, which typically runs on a CPU. The first method used on the GPU is using a kernel with a block dimension of 1x1. The second method is using a block dimension of 32x32. The last method is by using a kernel with a block dimension of 32x32 and shared local bins.

How is the GPU used to accelerate the application?

Parallel Algorithm/CUDA Reduction Algorithm: A reduction algorithm will be implemented to sum up pixel values efficiently.

Problem Space Partitioning: Threads and Thread Blocks: The image will be divided into chunks (tiling), with each chunk assigned a number of threads. Multiple threads will be grouped into thread blocks to take advantage of the GPU architecture.

Data Parallelism: The task of calculating the histogram will be parallelized across the image's pixels, allowing for simultaneous computation.

Data Loading: Load image data into GPU memory.

Kernel Execution: Execute CUDA kernels for reduction and histogram computation.

Result Aggregation: Combine results atomically (atomicAdd) from each thread block into the final histogram. Shared memory for local binning was also used.

Comparison: Execute a similar pipeline in openCV for performance comparison against GPU computing.

Implementation Details

Data Transfer: Transfer image data to GPU memory using cudaMemcpy.

Kernel Functions: Write CUDA kernels for reduction and histogram computation. The first method uses a 1x1 block dimension. The second method uses a 32x32 block dimension. The third method uses a shared memory for local bin and also a 32x32 block dimension.

Memory Management: Efficiently manage GPU memory to avoid bottlenecks. Reduced use of _syncthread().

openCV Implementation: The Image is uploaded using OpenCV. The image is uploaded as black and white. Image features such as, dimension, and channel can be used. The image is then processed for histogram calculation using an OpenCV function calcHist().

Image Processing Toolbox: Utilize openCV built-in functions calcHist() for histogram computation.

Performance Measurement: Time the execution of the histogram computation.

Comparison: The time execution of the OpenCV on the CPU was much faster than the ones calculated by the GPU. This could be due to the size of the image. The larger the image the better the result would be for the GPU. The bigger block dimension on the GPU also demonstrated faster time.

Metrics: Measure execution time and compare the performance between CUDA and openCV implementations.

Analysis: Analyze the reasons for performance differences. The poorer performance of the GPU is due to its overhead cost. For it to perform better the input should have a much larger element count.

Documentation

[git@github.com:VincentVilda/Final-Project.git](https://github.com/VincentVilda/Final-Project.git)

Evaluation/Results

Performance Metrics: Time taken for histogram computation in both CUDA and openCV . Used three histogram kernels for testing trials with the amount of time elapsed.

Analysis: Time elapsed was slower for a GPU histogram than with a CPU(OpenCV histogram).

Visualization: Graphs comparing execution times for different image sizes and types.

Problems Faced

CUDA - Issues such as debugging parallel code, managing memory efficiently.

openCV - Issues: Optimizing openCV code for comparison and using proper library functions.

General Issues: Handling non square images was not tested, however the kernel with a dimension block of 1x1 would be able to perform on samples that are not square. Ensuring accurate performance measurement required the use of precise timing. Initially the timer.h was used but was conflicting with the results. A support code written by the Professor was used to accurately get the elapsed time for each performance.

Task	GPU histogram	CPU (OpenCV histogram Function)
Three Histogram Kernels used for Comparison	Time elapsed (s)	Time elapsed (s)
Kernel with 1x1 Block Dim	0.022059	0.000786
Kernel with 32x32 Block Dim	0.011056	0.000668
Kernel with 32x32 Block Dim and Share Memory Local Bin	0.011214	0.000847

The figure below shows a kernel with 1x1 block dimension used for gpu histogram. The result shows that the CPU is still much faster in producing the histogram.

```
112 //Initial Code with block dimension equal to 1//
113 __global__ void kernel(unsigned char* Im, int* Hist);
114
115 void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){
116
117     dim3 dimGrid(Width, Height);
118     kernel<<<dimGrid, 1 >>>(I_dev, bin_dev);
119 }
120
121
122
123
124 __global__ void kernel(unsigned char* Im, int* Hist){
125     int x = blockIdx.x;
126     int y = blockIdx.y;
127
128     int Im_Idx = x + y * gridDim.x;
129
130     atomicAdd(&Hist[Im_Idx], 1);
131 }
132
133
134
135
136 //Kernel with optimized block dimension//
137 // __global__ void kernel(unsigned char* Im, int* Hist, int Height, int Width);
138
139 // void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){
140 //     dim3 blockDim(32, 32);
141 //     dim3 gridDim((Width + blockDim.x - 1) / blockDim.x, (Height + blockDim.y - 1) / blockDim.y);
142 // }
```

make: warning: Clock skew detected. Your build may be incomplete.

• bender /home/ceemj/vvilda/new \$./histogram
Height= 256, Width= 256, Channels= 1
Running CPU histogram0.000786 s
Running GPU Histogram0.022859 s
bin[0]: CUDA = 102 , CPU = 102
bin[1]: CUDA = 42 , CPU = 42
bin[2]: CUDA = 60 , CPU = 60
bin[3]: CUDA = 87 , CPU = 87
bin[4]: CUDA = 76 , CPU = 76
bin[5]: CUDA = 127 , CPU = 127
bin[6]: CUDA = 181 , CPU = 181
bin[7]: CUDA = 261 , CPU = 261

The figure below shows a kernel with 32x32 block dimension used for gpu histogram. The result shows that the CPU is still much faster in producing the histogram. The result also shows that this is faster than the kernel with a block dimension of only 1x1.

```
116 // atomicAdd(&Hist[Im[Idx]], 1);
117 // }
118 ///////////////////////////////////////////////////
119
120
121
122 ///////////////////////////////////////////////////Kernel with optimized block dimension////////////////////////////////////
123 __global__ void kernel(unsigned char* Im, int* Hist, int Height, int Width);
124
125 void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){
126     dim3 blockDim(32, 32);
127     dim3 gridDim((Width + blockDim.x - 1) / blockDim.x, (Height + blockDim.y - 1) / blockDim.y);
128
129     kernel<<<gridDim, blockDim>>>(I_dev, bin_dev, Height, Width);
130 }
131
132
133 __global__ void kernel(unsigned char* Im, int* Hist, int Height, int Width){
134     int x = blockIdx.x * blockDim.x + threadIdx.x;
135     int y = blockIdx.y * blockDim.y + threadIdx.y;
136
137     if (x < Width && y < Height) {
138         int Idx = x + y * Width;
139         atomicAdd(&Hist[Idx], 1);
140     }
141 }
142
143
144 ///////////////////////////////////////////////////
145
146 ///////////////////////////////////////////////////Kernel with optimized dim block and shared memory bins////////////////////////////////////
147
```

```
bash
Height= 256, Width= 256, Channels= 1
Running CPU histogram0.000668 s
Running GPU Histogram0.011056 s
bin[0]: CUDA = 102 , CPU = 102
bin[1]: CUDA = 42 , CPU = 42
bin[2]: CUDA = 60 , CPU = 60
bin[3]: CUDA = 87 , CPU = 87
bin[4]: CUDA = 76 , CPU = 76
bin[5]: CUDA = 127 , CPU = 127
bin[6]: CUDA = 181 , CPU = 181
bin[7]: CUDA = 261 , CPU = 261
bin[8]: CUDA = 471 , CPU = 471
```

The figure below shows a kernel with shared local bin and 32x32 block dimension used for gpu histogram. The result shows that the CPU is still much faster in producing the histogram. The result also shows that this is faster than the kernel with a block dimension of only 1x1, and almost similar in speed as to the kernel with 32x32 blockdimension and no shared local bin.

```
148 //////////////////////////////////////////////////Kernel with optimized dim block and shared memory bins////////////////////////////////////
149 global__ void kernel(unsigned char* Im, int* Histogram, int Height, int Width){
150     __shared__ int local_hist[256]; // Shared memory for local histograms
151     int x = blockIdx.x * blockDim.x + threadIdx.x;
152     int y = blockIdx.y * blockDim.y + threadIdx.y;
153
154     // Initialize local histogram
155     for (int i = threadIdx.x; i < 256; i += blockDim.x) {
156         local_hist[i] = 0;
157     }
158     __syncthreads();
159
160     if (x < Width && y < Height) {
161         int Im_idx = x + y * Width;
162         atomicAdd(&local_hist[Im_idx], 1); // Update local histogram atomically
163     }
164     __syncthreads();
165
166     // Accumulate local histograms into global histogram
167     for (int i = threadIdx.x; i < 256; i += blockDim.x) {
168         atomicAdd(&Histogram[i], local_hist[i]);
169     }
170 }
171
172 void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){
173     dim3 blockDim(32, 32); // Choose appropriate block size
174     dim3 gridDim((Width + blockDim.x - 1) / blockDim.x, (Height + blockDim.y - 1) / blockDim.y);
175
176     // Allocate mem for histogram on device
177     cudaMemset(bin_dev, 0, 256 * sizeof(int));
178
179     kernel<<<gridDim, blockDim>>>(I_dev, bin_dev, Height, Width);
180 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Height= 256, Width= 256, Channels= 1
Running CPU histogram0.000456 s
Running GPU Histogram0.011214 s
bin[0]: CUDA = 102 , CPU = 102
bin[1]: CUDA = 42 , CPU = 42
bin[2]: CUDA = 60 , CPU = 60
bin[3]: CUDA = 87 , CPU = 87
bin[4]: CUDA = 76 , CPU = 76
bin[5]: CUDA = 127 , CPU = 127
bin[6]: CUDA = 181 , CPU = 181
bin[7]: CUDA = 261 , CPU = 261
```

The two images below show the result of the histogram computed by the GPU as compared to the one that is calculated by the OpenCV function in the CPU.

new [SSH: bender.engr.ucr.edu]

EXPLORER

- NEW [SSH: BENDER.ENG...]
- cameraman.png
- histogram
- Histogram_Image.png
- Lenna.png
- main.cu
- main.o
- Makefile
- support.cu
- support.h
- support.o

main.cu

```
148 //////////////////////////////////Kernel with optimized dim block and shared memory bins////////////////////////////////////
149 __global__ void kernel(unsigned char* I, int* Histogram, int Height, int Width){
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Running CPU histogram0.000456 s
Running GPU Histogram0.011214 s

bin[0]:	CUDA = 102 ,	CPU = 102
bin[1]:	CUDA = 42 ,	CPU = 42
bin[2]:	CUDA = 66 ,	CPU = 66
bin[3]:	CUDA = 87 ,	CPU = 87
bin[4]:	CUDA = 76 ,	CPU = 76
bin[5]:	CUDA = 127 ,	CPU = 127
bin[6]:	CUDA = 181 ,	CPU = 181
bin[7]:	CUDA = 261 ,	CPU = 261
bin[8]:	CUDA = 471 ,	CPU = 471
bin[9]:	CUDA = 769 ,	CPU = 769
bin[10]:	CUDA = 875 ,	CPU = 875
bin[11]:	CUDA = 941 ,	CPU = 941
bin[12]:	CUDA = 1422 ,	CPU = 1422
bin[13]:	CUDA = 1681 ,	CPU = 1681
bin[14]:	CUDA = 1596 ,	CPU = 1596
bin[15]:	CUDA = 1485 ,	CPU = 1485
bin[16]:	CUDA = 858 ,	CPU = 858
bin[17]:	CUDA = 479 ,	CPU = 479
bin[18]:	CUDA = 325 ,	CPU = 325
bin[19]:	CUDA = 238 ,	CPU = 238
bin[20]:	CUDA = 219 ,	CPU = 219
bin[21]:	CUDA = 153 ,	CPU = 153
bin[22]:	CUDA = 175 ,	CPU = 175
bin[23]:	CUDA = 131 ,	CPU = 131
bin[24]:	CUDA = 129 ,	CPU = 129
bin[25]:	CUDA = 111 ,	CPU = 111
bin[26]:	CUDA = 132 ,	CPU = 132
bin[27]:	CUDA = 125 ,	CPU = 125
bin[28]:	CUDA = 99 ,	CPU = 99
bin[29]:	CUDA = 104 ,	CPU = 104
bin[30]:	CUDA = 117 ,	CPU = 117
bin[31]:	CUDA = 103 ,	CPU = 103
bin[32]:	CUDA = 76 ,	CPU = 76
bin[33]:	CUDA = 104 ,	CPU = 104
bin[34]:	CUDA = 103 ,	CPU = 103
bin[35]:	CUDA = 106 ,	CPU = 106
bin[36]:	CUDA = 83 ,	CPU = 83
bin[37]:	CUDA = 85 ,	CPU = 85
bin[38]:	CUDA = 95 ,	CPU = 95
bin[39]:	CUDA = 97 ,	CPU = 97
bin[40]:	CUDA = 77 ,	CPU = 77
bin[41]:	CUDA = 81 ,	CPU = 81
bin[42]:	CUDA = 94 ,	CPU = 94
bin[43]:	CUDA = 84 ,	CPU = 84
bin[44]:	CUDA = 82 ,	CPU = 82
bin[45]:	CUDA = 69 ,	CPU = 69
bin[46]:	CUDA = 69 ,	CPU = 69
bin[47]:	CUDA = 65 ,	CPU = 65
bin[48]:	CUDA = 82 ,	CPU = 82

SSH: bender.engr.ucr.edu 0 0 0 0

Ln 148, Col 102 Spaces: 4 UTF-8 LF CUDA C++

new [SSH: bender.engr.ucr.edu]

08

EXPLORER

NEW [SSH: BENDER.ENG...]

cameraman.png

histogram

Histogram_Image.png

Lenna.png

main.cu

main.o

Makefile

support.cu

support.h

support.o

OUTLINE

TIMELINE

main.cu

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

bin[119]: CUDA = 360 , CPU = 360

bin[120]: CUDA = 334 , CPU = 334

bin[121]: CUDA = 382 , CPU = 382

bin[122]: CUDA = 354 , CPU = 354

bin[123]: CUDA = 353 , CPU = 353

bin[124]: CUDA = 364 , CPU = 364

bin[125]: CUDA = 352 , CPU = 352

bin[126]: CUDA = 347 , CPU = 347

bin[127]: CUDA = 364 , CPU = 364

bin[128]: CUDA = 399 , CPU = 399

bin[129]: CUDA = 458 , CPU = 458

bin[130]: CUDA = 396 , CPU = 396

bin[131]: CUDA = 392 , CPU = 392

bin[132]: CUDA = 417 , CPU = 417

bin[133]: CUDA = 418 , CPU = 418

bin[134]: CUDA = 412 , CPU = 412

bin[135]: CUDA = 402 , CPU = 402

bin[136]: CUDA = 398 , CPU = 398

bin[137]: CUDA = 377 , CPU = 377

bin[138]: CUDA = 385 , CPU = 385

bin[139]: CUDA = 359 , CPU = 359

bin[140]: CUDA = 335 , CPU = 335

bin[141]: CUDA = 332 , CPU = 332

bin[142]: CUDA = 322 , CPU = 322

bin[143]: CUDA = 394 , CPU = 394

bin[144]: CUDA = 371 , CPU = 371

bin[145]: CUDA = 443 , CPU = 443

bin[146]: CUDA = 486 , CPU = 486

bin[147]: CUDA = 497 , CPU = 497

bin[148]: CUDA = 554 , CPU = 554

bin[149]: CUDA = 582 , CPU = 582

bin[150]: CUDA = 538 , CPU = 538

bin[151]: CUDA = 607 , CPU = 607

bin[152]: CUDA = 521 , CPU = 521

bin[153]: CUDA = 524 , CPU = 524

bin[154]: CUDA = 540 , CPU = 540

bin[155]: CUDA = 607 , CPU = 607

bin[156]: CUDA = 723 , CPU = 723

bin[157]: CUDA = 674 , CPU = 674

bin[158]: CUDA = 695 , CPU = 695

bin[159]: CUDA = 752 , CPU = 752

bin[160]: CUDA = 822 , CPU = 822

bin[161]: CUDA = 974 , CPU = 974

bin[162]: CUDA = 1250 , CPU = 1250

bin[163]: CUDA = 1423 , CPU = 1423

bin[164]: CUDA = 1197 , CPU = 1197

bin[165]: CUDA = 1164 , CPU = 1164

bin[166]: CUDA = 1079 , CPU = 1079

bin[167]: CUDA = 987 , CPU = 987

bin[168]: CUDA = 975 , CPU = 975

bin[169]: CUDA = 910 , CPU = 910

SSH: bender.engr.ucr.edu

0 0 0

Ln 148, Col 102 Spaces: 4 UTF-8 LF CUDA C++