Fisheye Lens Distortion Correction on Multicore and Hardware Accelerator Platforms

Konstantis Daloukas\textsuperscript{1} Christos D. Antonopoulos\textsuperscript{1} Nikolaos Bellas\textsuperscript{1} Sek M. Chai\textsuperscript{2}

\textsuperscript{1}Department of Computer and Communications Engineering
University of Thessaly
Volos, Greece

\textsuperscript{2}Motorola Inc.
Schaumburg, IL, USA
Introduction

Wide-angle lenses (a.k.a. fisheye lenses) are traditionally used to enlarge the field of view in photography.

A. Conventional rectilinear lens

B. Full-frame fisheye lens 98 degrees horizontal by 147 degrees vertical

C. Full circular fisheye lens 180 degrees horizontal and vertical
Introduction

• Main Applications
  – Meteorology
  – Astronomy
  – Robot Navigation
  – Video Surveillance
  – Video Conferencing
  – Digital Cameras

• The incoming rays are mapped onto a spherical surface

• Such mapping introduces barrel distortion
Motivation

• Explore the mapping of the algorithm’s inherent parallelism on three contemporary platforms:
  – x86 Chip Multiprocessor (Core 2 Quad)
  – Cell B.E. processor
  – Virtex-4 FPGA

• Present a detailed characterization of the performance using both high- and low-level metrics
Outline

• Introduction

• **Wide-angle Lenses Distortion Correction Algorithm**

• Description of Target Platforms

• Algorithm Optimizations

• Performance Evaluation

• Conclusions
Wide-angle Lenses
Distortion Correction

Transformation of the distorted wide-angle images back to the central perspective space.
Projection Model of Wide-angle Lenses

Central Perspective Projection

Wide-angle Projection

\[ \alpha_1 = \beta_1 \quad \alpha_2 = \beta_2 \]

\[ \frac{\alpha_1}{d_1} = \frac{\alpha_2}{d_2} \]
Algorithmic Flow (A)

- **Inverse Mapping**: Maps each image point \((i, j)\) to the corresponding point \((x, y)\) in the wide-angle space.

\[
\begin{bmatrix}
X_c \\
Y_c \\
Z_c
\end{bmatrix} = \begin{bmatrix}
r_{11} & r_{12} & r_{13} \\
r_{21} & r_{22} & r_{23} \\
r_{31} & r_{32} & r_{33}
\end{bmatrix} \begin{bmatrix}
i \\
j \\
1
\end{bmatrix}
\]

\[
x = \frac{2R}{\pi} \tan \left[ \frac{\sqrt{(X_c)^2 + (Y_c)^2}}{Z_c} \right] + d_x + x_h
\]

\[
y = \frac{2R}{\pi} \tan \left[ \frac{\sqrt{(X_c)^2 + (Y_c)^2}}{Z_c} \right] + d_y + y_h
\]
Algorithmic Flow (A)

- Need to approximate the value of fractional positions in the fisheye space
- Complex memory access pattern
Algorithmic Flow (B)

- **Bicubic Interpolation**: uses a 4x4 window of pixels to approximate intermediate points

![Diagram showing the algorithmic flow and bicubic interpolation process.](image-url)
Algorithmic Flow (B)

- Bicubic interpolation is broken into horizontal and vertical 1D interpolation
- \( C_i \) are the pixel values

\[
g(x) = C_1*U_1(s) + C_2*U_2(s) + C_3*U_3(s) + C_4*U_4(s)
\]

\[
U_1(s) = (-s^3 + 2s^2 - s)/2
\]

\[
U_2(s) = (3s^3 - 5s^2 + 2)/2
\]

\[
U_3(s) = (-3s^3 + 4s^2 + s)/2
\]

\[
U_4(s) = (s^3 - s^2)/2
\]

\[
G(x,y) = g_1(x)*V_1(t) + g_2(x)*V_2(t) + g_3(x)*V_3(t) + g_4(x)*V_4(t)
\]

\[
V_1(t) = (-t^3 + 2t^2 - t)/2
\]

\[
V_2(t) = (3t^3 - 5t^2 + 2)/2
\]

\[
V_3(t) = (-3t^3 + 4t^2 + t)/2
\]

\[
V_4(t) = (t^3 - t^2)/2
\]
Complete Algorithm

For each pixel \((i, j)\) in the central perspective space \{
  Apply \textit{inverse mapping} to find fractional coordinates \((x, y)\) in the wide-angle space.
  Use \textit{bicubic interpolation} to approximate the pixel value at \((x,y)\).
\}

Apply a 2D low pass filter and downscale output image to VGA resolution \((640x480)\)
Outline

• Introduction
• Wide-angle Lenses Distortion Correction Algorithm
• Description of Target Platforms
• Algorithm Optimizations
• Performance Evaluation
• Conclusions
Intel Core 2 Quad

- A mainstream homogeneous multicore system
- 2.5 GHz operating frequency
- 1.3 GHz FSB
- Organized as two independent dual core processor blocks
- 3MB L2 cache for each block
- 64KB L1 cache for each processor
- Supports the SSE 4.1 vector instruction set
Cell Broadband Engine

- A heterogeneous multicore processor
- Integrates a 2-way SMT PPC and 8 SPEs
- 3.2 GHz operating frequency
- Each SPE contains:
  - A 128-bit wide SIMD execution engine
  - 256KB private Local Store
- On-chip network (EIB) with 307.2 GBps peak perf.
- Peak Performance:
  - 204.8 GFlops for single-precision
  - 14.63 GFlops for double-precision
Virtex-4 LX80 FPGA

- Arrays of uncommitted logic blocks
- Flexibility in tailoring the architecture to match the application
- High power efficiency
- Virtex-4 LX80:
  - 80,640 logic cells
  - 62.5 MHz operating frequency
- Main drawbacks:
  - Programmed primarily with HDLs
  - Low clock frequency
- Correction module generated using the *Proteus* architectural synthesis tool
Proteus

• Produces hardware accelerators that follow the streaming paradigm
  – Produces several load/store units and the datapath as well
• The application is expressed using an assembly-like streaming DFG
• Source code is modulo-scheduled with II = 2
• Generate 100K lines of synthesizable Verilog from 800 lines of code
Outline

• Introduction
• Wide-angle Lenses Distortion Correction Algorithm
• Description of Target Platforms
• Algorithm Optimizations
• Performance Evaluation
• Conclusions
High-Level Optimizations

• **Block Tiling**
  – Partition the output image in blocks and correct a block of pixels at a time
  – Alleviates the problem of prefetching
  – Facilitates efficient data partitioning (x86 and Cell) and task-level pipelining (FPGA)
Low-Level Optimizations

- x86 and Cell:
  - SIMD Optimization
  - Explicit loop unrolling
  - Eliminate pipeline stalls from data dependencies
Low-Level Optimizations

• x86 and Cell:
  – Inverse-mapping amortization

• Cell-specific:
  – Manual instruction scheduling

• FPGA
  – Modulo scheduling with II = 2
  – 400 sDFG operations in all pipeline stages
Outline

- Introduction
- Wide-angle Lenses Distortion Correction Algorithm
- Description of Target Platforms
- Algorithm Optimizations
- Performance Evaluation
- Conclusions
Performance and Scalability Analysis

- Inverse Mapping Amortization
- HL+LL optimizations
- HL optimizations

<table>
<thead>
<tr>
<th>Device</th>
<th>Only PPE</th>
<th>1 SPE</th>
<th>2 SPE</th>
<th>4 SPE</th>
<th>8 SPE</th>
<th>1T</th>
<th>2T</th>
<th>4T</th>
<th>Virtex-4 LX80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Speed (Frames/Sec)</td>
<td>0.55 fps</td>
<td>3.83 fps</td>
<td>7.86 fps</td>
<td>14.95 fps</td>
<td>29.94 fps</td>
<td>3.70 fps</td>
<td>8.01 fps</td>
<td>15.82 fps</td>
<td>22.28 fps</td>
</tr>
</tbody>
</table>

- Cell
- Core 2 Quad
- FPGA
Performance and Scalability Analysis

Module Runtime Breakdown

- Inverse Mapping
- Bicubic Interpolation
- Low Pass Filter

Cell

- Only PPE
- HL, 1 SPE
- HL, 2 SPE
- HL, 4 SPE
- HL, 8 SPE
- HL+LL, 1 SPE
- HL+LL, 2 SPE
- HL+LL, 4 SPE
- HL+LL, 8 SPE
- IMA, 1 SPE
- IMA, 2 SPE
- IMA, 4 SPE
- IMA, 8 SPE

Core 2 Quad

- HL, 1T
- HL, 2T
- HL, 4T
- HL+LL, 1T
- HL+LL, 2T
- HL+LL, 4T

FPGA

- Virtex-4 LX80

April 20, 2010  IPDPS 2010  24
Memory Performance

Average Off-Chip Bandwidth

<table>
<thead>
<tr>
<th></th>
<th>8 threads</th>
<th>4 threads</th>
<th>2 threads</th>
<th>1 thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core2 Quad</td>
<td>HL optimizations</td>
<td>HL + LL optimizations</td>
<td>IMA</td>
<td></td>
</tr>
<tr>
<td>Cell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core2 Quad</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtex-4 LX 80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MBytes/sec

April 20, 2010

IPDPS 2010
Development Cost

- A significant factor that must be considered
  - One aspect in the comparison of programming models in the three platforms
  - Use Lines-of-Code (LOC) as the primary metric

- Initial single-threaded version: 800 lines

- Fully-optimized version for x86: extra 500 LOC

- Fully-optimized version for Cell: extra 1500 LOC

- FPGA Implementation: 800 assembly-like LOC
  - Requires multiple time-consuming synthesis and Place & Route iterations
Outline

• Introduction
• Wide-angle Lenses Distortion Correction Algorithm
• Description of Target Platforms
• Algorithm Optimizations
• Performance Evaluation
• Conclusions
Conclusions

• Presented the implementation of a real-time image warping algorithm
  – Analyzed and characterized the performance on all underlying architectures
  – Applied a series of optimizations and identified their effect
• Commercially available general purpose multi-cores not capable of handling real-time distortion correction
• Exotic architectures such as Cell or FPGAs offer the necessary computational power
  – Significantly higher development cost
  – Advanced tools, development models and support environments can alleviate this effort
Acknowledgements

• We would like to thank *Barcelona Supercomputing Center* for providing us with access to their IBM QS20 blade

• This project is partially supported by the EC Marie Curie International Reintegration Grant (IRG) 223819