Optimizing a Mobile Robot Control System using GPU Acceleration

Nat Tuck, Michael McGuinness, Fred Martin
University of Massachusetts Lowell
Presentation Outline

- Control Program Design
- Simulation
- Profiling
- GPU Optimization
- Conclusion
Nav Challenge Control Program

- Control Program for IGVC Navigation Challenge
  - Subset of Autonomous Challenge Design

- Sensor Modules
  - Position Estimation, Laser Rangefinder, Stereo Vision

- Control Modules
  - Waypoint Navigation, Mapping, Pathfinding, Obstacle Avoidance

- Built using the Robot Operating System (ROS)
Gazebo Simulation
Performance Problems

- Our control program did not run in real time.
- Saturated two cores.
- Map didn't stay up to date.
  - Slow stereo vision?
  - Slow point cloud processing?
  - But fast laser scan didn't seem to help.
- Robot couldn't figure out where to go.
Needs to run on GPU

- Quad Core i5: 50 GFLOPS
- Radeon 6850: 1500 GFLOPS
Profiling Techniques

• Polling process CPU usage %
  • Script recorded time at 10 Hz
  • Not the highest quality measurement
  • Found candidates for deeper consideration

• Wrapped expensive function calls in timers.
  • Better measurement
  • Required code modification
Hot Spots

• Expected:
  • Path Finding
    • Runs A* path finding as fast as possible.
  • Stereo Vision
  • Point Cloud Processing

• Surprising
  • Mapping
    • Applies laser scans to occupancy grids, which are then merged.
Bacon

- Programming language / runtime system
- Generates specialized OpenCL based on runtime parameters.

Features:
- Runtime-size aware loop unrolling
- Runtime “static” array allocation
- Multi-dimensional array syntax
- Automatic generation of OpenCL C++ wrapper
GPU Implementation: Path Finder

- No obvious way to parallelize A*
  - Basically serial, needs values for one cell before the next cell can be expanded.

- Not a real performance bottleneck, just eats CPU time.
Mapping and Map Merging
GPU Mapper

- Parallel mapping was pretty straightforward:
  - Apply laser hits to grid in parallel
  - Apply laser misses to grid in parallel
  - Ignore collisions
    - Technically not defined behavior.
    - In practice, just fails to perform duplicate updates.

- Map merging was even simpler:
  - Merge cells in parallel
GPU Stereo Disparity

- Started with highly optimized OpenCV routine.
- Porting to GPU was very difficult.
  - Had to understand optimized C++ code.
  - Optimizations were CPU-specific
    e.g. Serial cache access pattern
GPU Stereo Disparity

- Started with highly optimized OpenCV routine.
- Porting to GPU was very difficult.
  - Had to understand optimized C++ code.
  - Optimizations were CPU-specific e.g. Serial cache access pattern
- Result was 50% slower.
GPU Point Cloud Processing

• Based on our results for Stereo Disparity, we decided not to try to beat the Point Cloud Library (PCL).
## Speedups

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td>0.039</td>
<td>0.006</td>
</tr>
<tr>
<td>Map Merging</td>
<td>1.150</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Kernel execution times in seconds.

- Robot ran much nicer in simulation.
CPU Usage - Comparison

Before

- 58%
- 14%
- 14%
- 7%

After

- 39%
- 14%
- 7%
- 7%
Conclusions

- Simple GPU version beats simple CPU version
  - If the problem can be easily parallelized

- Well optimized code is hard to beat, even with theoretically much faster hardware
  - May not be worth the effort
Thank You

Bacon is available under an open source license from

http://www.cs.uml.edu/~ntuck