An Interior Point Optimization Solver for Real Time Inter-Frame Collision Detection: Exploring Resource-Accuracy-Platform Tradeoffs

Brian Leung, Chih-Hung Wu, Seda Ogrenci Memik, Sanjay Mehrotra†

Department of Electrical Engineering and Computer Science
Department of Industrial Engineering†

Email Contact: ble973@eecs.northwestern.edu

Linear Programming (LP)

- Linear programming (LP) is an optimization technique for linear objective functions given certain linear equality constraints.
- The Simplex algorithm, developed by George Dantzig, is the oldest and most popular method for solving LPs.
- The most current generation interior-point software is based on the predictor-corrector method developed by Mehrotra.
  - Better performance is achieved by reusing the Cholesky decomposition (most computationally expensive) in each iteration to calculate a predictor and corrector to steer the algorithm in a direction to converge faster.

Linear Programming Applications

- LP has many applications ranging from:
  - Design and optimization of hardware systems, integrated circuits, and software.
    - Reda and Choudary used it to place and route VLSI circuits to memory.
    - Kandemir, Banerje, Choudhary, Ramanujam, and Ayguadé used it for code optimizations in compilers.

LP is also used for collision detection!

What is Collision Detection?

- Collision detection is the use of algorithms for detecting collisions (intersections) of two or more objects.
- Collision detection has many applications.
  - Auto body design.
  - High speed impact tests.
    - Bullet proof products, anti-air weapons, air/fluid modeling.
  - Movie productions and animations.
  - Video games.
- A fast and robust collision detection algorithm is necessary to ensure physical reality is translated to virtual reality.

*http://www.preparednessresource.com/insurance.ht

Contents of Talk

- Overview
- Related Work
- Collision Detection Algorithm
  - Linear Program Formulation
  - Solving the Linear Problem
- Experimental Setup
- Hardware Architecture/System Overview
- Results
  - Matrix Inversion
  - Floating Point VS Fixed Point
  - GPGPU VS FPGA
- Conclusions

Overview

- We implemented the Industrial Virtual Reality Institute Collision Detection (IVRI-CD) algorithm, which is based on the primal-dual predictor-corrector interior point algorithm, on a FPGA and GPGPU for real time collision detection.
  - IVRI-CD is an algorithm developed by Akgunduz and Mehrotra for collision detection based on LP formulation.
- We compare the platforms in terms of resource usage, data accuracy/precision, and system efficiency.

Overview

- We examine:
  - FPGA
    - 32-bit floating point implementation
    - 32-bit fixed point implementation
  - GPGPU
    - 32-bit floating point implementation
  - Trade-offs associated with different matrix manipulation methods on the FPGA.
    - Gaussian Elimination.
    - Shipley-Coleman method.
    - Cholesky Factorization.
  - Fixed point vs floating point implementations on a FPGA.
  - Benefits of using a GPGPU rather than a FPGA.

Related Work

- Most of the existing solutions have been in accelerating interior point based LPs in software.
  - Some include utilizing the concept of bounding boxes to manage computational complexity.
  - Some assume larger time steps, but more errors.
- Hardware implementations exist.
  - Simplex algorithm.
  - Lemke's algorithm.
  - Depth maps, which store distance and normal values, for collision detection.

Collision Detection Algorithm

- Our implemented method, IVRI-CD developed by Akgunduz and Mehrotra, is a linear-programming based approach for collision detection.
  - The Linear Program Formulation
  - Solving the Linear Problem

Linear Program Formulation

- The input is a frame containing 3-D convex objects.
  - Each Object is defined with 3 vectors v_i^x, v_i^y, and v_i^z
- The basic LP performs pair wise collision detection for all objects in the frame by representing them as convex hulls:

$$\sum_{j=1}^{n_1} v_j \alpha_j - \sum_{j=1}^{n_2} w_j \beta_j + u\varphi = 0$$

- Primal decision variables $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$
- $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$ denote the number of vertices of objects $\boldsymbol{v}$ and $\boldsymbol{w}$, respectively.
- $\boldsymbol{u\varphi}$ is equal to zero when objects $\boldsymbol{v}$ and $\boldsymbol{w}$ overlap and is positive when objects $\boldsymbol{v}$ and $\boldsymbol{w}$ do not overlap.

Linear Program Formulation

- From the previous equation, ***A*** is a 5 x (n_1+n_2+1)

$$A = \begin{bmatrix} v_1^x & \dots & v_{n_1}^x & -w_1^x & \dots & -w_{n_2}^x & u^x \\ v_1^y & \dots & v_{n_1}^y & -w_1^y & \dots & -w_{n_2}^y & u^y \\ v_1^z & \dots & v_{n_1}^z & -w_1^z & \dots & -w_{n_2}^z & u^z \\ 1 & \dots & 1 & 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 & 0 \end{bmatrix}$$

- Where u is:

$$u = \frac{1}{n_2}\left(\sum_{j=1}^{n_2} w_j\right) - \frac{1}{n_1}\left(\sum_{j=1}^{n_1} v_j\right)$$

Linear Program Formulation

- Primal decision variable vector:

$$\gamma = [\alpha_1 \ldots \alpha_{n1}, \ \beta_1 \ldots \beta_{n2}, \ \varphi \ ]$$

where $\alpha_i = 1/n_1$ and $\beta_j = 1/n_2$

Linear Program Formulation

- Given the decision variables and constraints previously defined:
  - Primal Model: Min $\boldsymbol{\mu\varphi}$ where $\boldsymbol{\mu} = \max\{abs(u)\}$
  - Dual Model: Max $(\boldsymbol{\pi}_4 + \boldsymbol{\pi}_5)$ subject to the following constraints.
    - $\mathbf{A}^T\boldsymbol{\pi} + \mathbf{s} = \mathbf{c}$
    - $\mathbf{s} \geq 0$
    - $\boldsymbol{\pi} = [\, 0\ 0\ 0\ \boldsymbol{\eta}\ \boldsymbol{\eta}\,]$
- $\boldsymbol{\eta}$ is the maximum absolute value among all x, y, and z vertex coordinates from two objects.
- $\mathbf{c}$ is then defined as a vector with *(n+1)* elements, containing *n* zeros and the value of μ as its last element.
- $\boldsymbol{s}$ is a *(n+1)* vector containing the slack variables.
- $\boldsymbol{\pi}$ is a vector with five elements, where the first three elements are zero and the last two are of value $\boldsymbol{\eta}$.

Solving the Linear Problem

- Initialize the matrices containing the dual and primal variables ***X***, ***S***, and ***D***.
  - ***X*** and ***S*** are diagonal matrices containing the elements of vectors **π** and ***s***.
  - ***D*** is defined as $D = S^{-1} * X$.
- Compute the primal dual affined scaling trajectory using Cholesky factorization, the column vector of search directions for slack variables, and a column vector of search dire$\wp_\pi = -(ADA^T)^{-1}b$il va$\wp_s = -A^T\wp_\pi \qquad \wp_\gamma = \gamma - D\wp_s$
- ($f_\gamma = 0.95 * \min_i \left\{ \frac{\gamma_i^t}{\wp_{\gamma_i}} \middle| \wp_{\gamma_i} \phi\ 0 \right\}$lual s$f_s = 0.95 * \min_i \left\{ \frac{s_i^t}{\wp_{s_i}} \middle| \wp_{s_i} \phi\ 0 \right\}$

Solving the Linear Problem

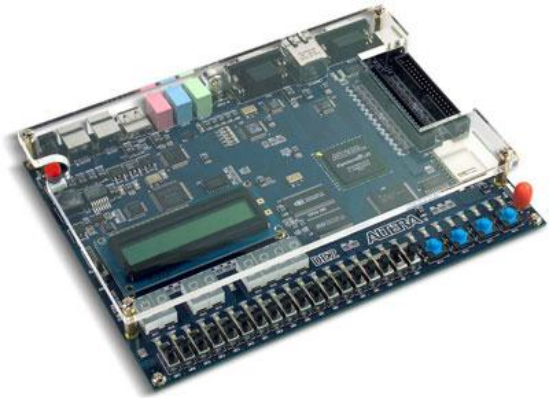
- Create trial points: the variables computed above established a search direction for generating the trial points.

$$\gamma^{t+1} = \gamma^{t} - f_{\gamma}\wp_{\gamma} \qquad s^{t+1} = s^{t} - f_{s}\wp_{s} \qquad \pi^{t+1} = \pi^{t} - f_{s}\wp_{\pi}$$

- Check for collision
  - If ($\boldsymbol{u\varphi} < 10^{-6}$) => STOP! Collision!
  - Else if ($\boldsymbol{\pi}_4 + \boldsymbol{\pi}_5 > 0$) => STOP! No collision!
  - Else => Repeat algorithm.
- Output:
  - 00 = no collision detected.
  - 11 = collision detected
  - 10 = undetermined output, reiterate

Experimental Setup

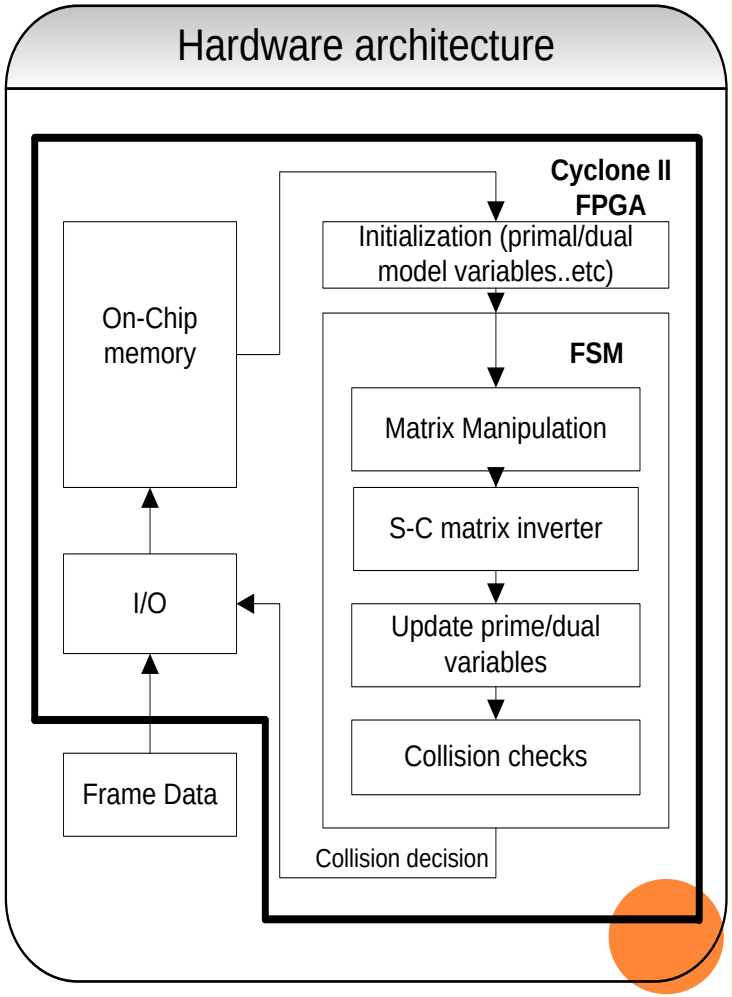
- In our experiments, we used the Altera DE2 board.
  - Megawizard tool for floating point arithmetic.
  - For fixed point arithmetic, we used the VHDL-200x packages under IEEE754 for our system
  - Six 32x32 RAM blocks for storing data.
  - NIOS II processor used for data transfer between PC and FPGA.
- We used a NVIDIA GeForce 8800 GT graphics card with CUDA v2.0 for the GPGPU experiments.
- Two objects of equally sized pentagonal prisms with ten frames are used for simulation.



*http://knightvision.calvin.edu/bbcswebdav/orgs/ENGR/senior-projects/2007-08/Team03/images/de2-dev-board.jpg

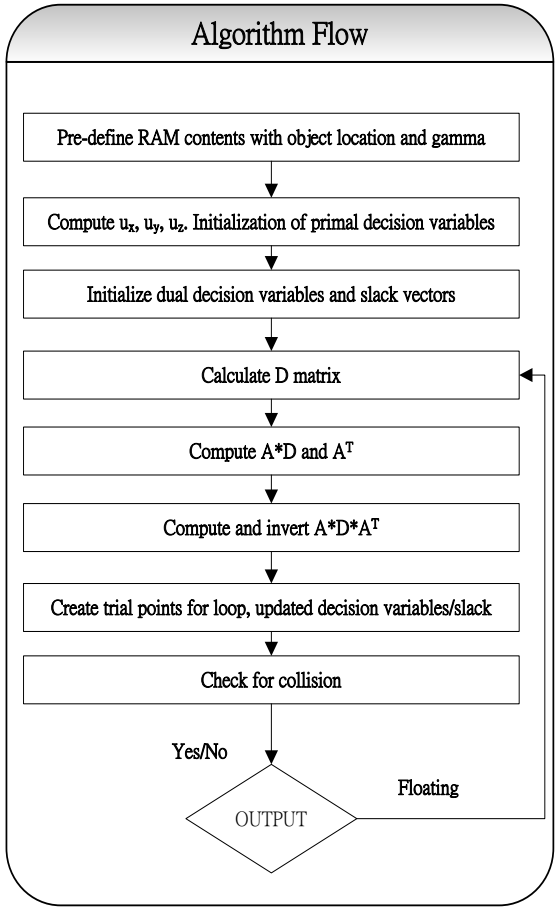
Hardware Architecture/System Overview

- Overall system can be viewed as a large FSM with 38 states.
- I/O
  - NIOS II processor instantiated to input and output via USB blaster.
  - Inputs:
    - coordinates of vertices v_j and w_j for objects v and w
    - a primal decision variable vector
    - a reset signal
  - Output:
    - 2-bit collision signal
    - error signals.
- On-Chip memory
  - Instantiated M4K memory blocks for data storage.
- Initialization
  - Derive in parallel initial pair wise distances, u_x, u_y, and u_z, of vertices in frame.
- FSM
  - Solves LP formulation to detect collision.



Hardware Architecture/System Overview

- The FSM hardware performs the operations shown in the figure on the right.
  - ***A*** is a *5 x (2*n+1)* matrix,
  - ***D*** is a *(2*n+1) x (2*n+1)* diagonal matrix.
  - Modified addresses in memory to avoid transpose function.
- Matrix Inversion is the most computationally expensive portion of the algorithm.
- A separate transpose module is avoided since it is just different addressing of a matrix stored in memory.



Results (Matrix Inversion)

- The most resource hungry portion of algorithm is the matrix inversion block, which requires inversion of a 5x5 matrix.
- A comparison between two in-place inversion approaches and Cholesky Factorization for a 5x5 matrix.
- We decided to go with the Shipley-Coleman approach.
  - Slightly slower, but less resource usage.
  - Requires 5 divide operations and no square root function.

	Shipley-Coleman	**Gauss-Jordan**	**Cholesky Factorization**
Execution Time	52.098 µs	45.024 µs	63.012 µs
Resource usage	20 %	35 %	31%

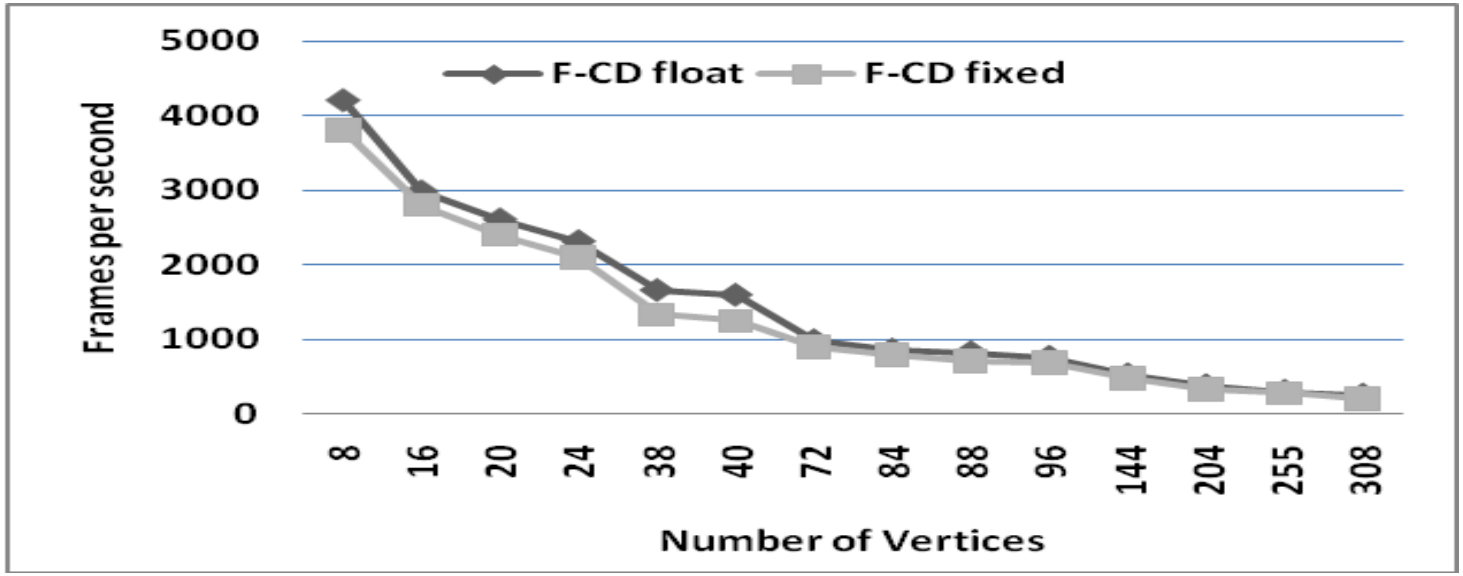
Results: Floating Point VS Fixed Point

- Floating point:
  - 27,903 cycles are needed for one iteration of algorithm.
  - Maxes out resources at 93% for objects with 45 vertices.
- Fixed point:
  - For the same 93% resource usage, fixed point can accommodate 50 vertices.
- Fixed point has better resource utilization and can provide ~11% increased object resolution over floating point.
- Precision is affected in fixed point and on average requires 1-2 additional iterations of the algorithm.

Resource	Floating point	Fixed Point
Max object resolution	45	50
Max clock frequency	88.2MHz	56.7MHz
Throughput (frames/sec)	1562 F/s	1000 F/s

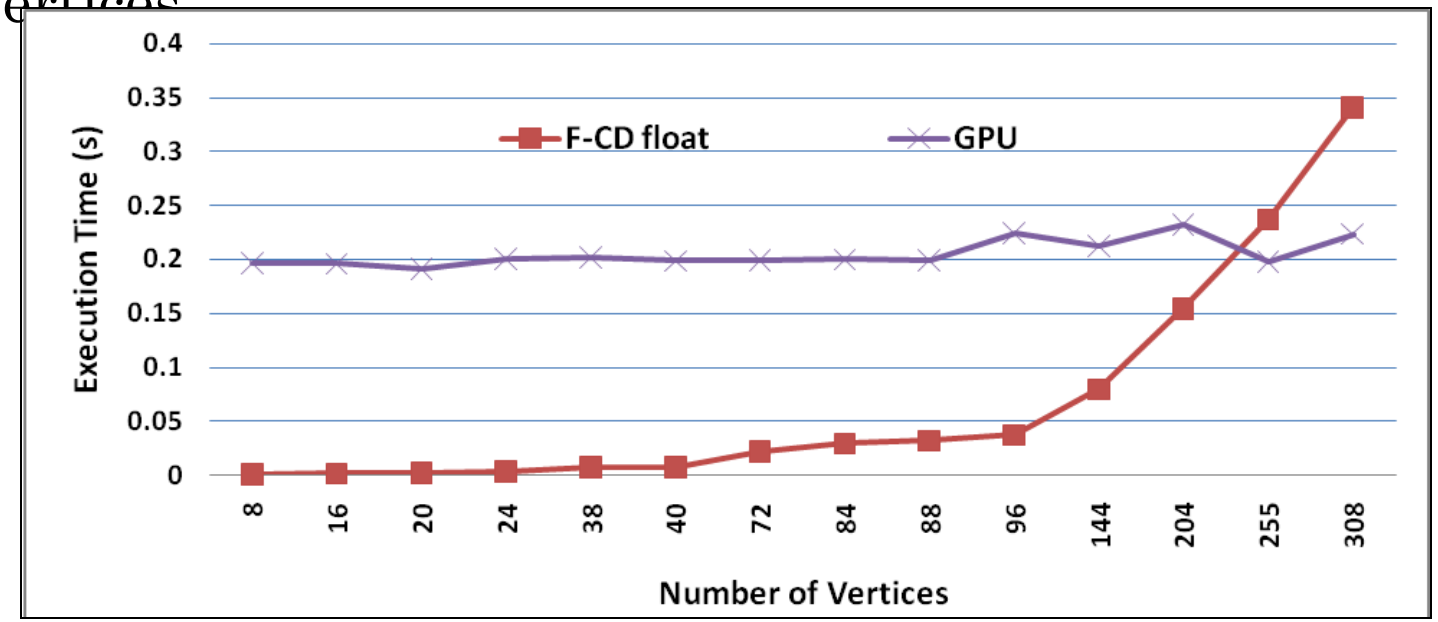
Results: Floating Point VS Fixed Point

- Fixed point and floating point comparison for the FPGA.
- Note that floating points maxed out its resources at 93% with 45 vertices; hence, results beyond 45 vertices were extrapolated.
- At 45 vertices, floating point processes 1562 frames per second, and fixed point processes 1321 frames per second.



Results: GPGPU VS FPGA

- For a small number of vertices, GPGPU is not as efficient.
  - For 45 vertices, FPGA has 11x speedup over GPGPU.
  - At 242 vertices, the GPGPU is on par with the FPGA.
  - At 308, the GPGPU has overtaken then FPGA.
- Note that the GPGPU does not have the same limitation of fitting the design at more than 45 vertices



Conclusions

- We presented two FPGA and one GPGPU implementations of a collision detection algorithm based on an interior-point linear programming approach.
- We explored resource, accuracy, and platform tradeoffs of:
  - Different techniques for matrix inversion and Cholesky Factorization (most computationally expensive portion of algorithm).
  - Fixed point vs floating point on an FPGA.
  - GPGPU vs FPGA.

Thank you!