Accuracy of the Immersed Boundary Method in Fixed-Point Arithmetic

Gabor J. Ferencz, Jr.[1], **Eric R. Peskin**[1], **and Charles S. Peskin**[2]
[1]Electrical and Microelectronic Engineering Dept., Rochester Institute of Technology, Rochester, NY, USA
[2]Courant Institute of Mathematical Sciences, New York University, New York, NY, USA

Abstract— *The* immersed boundary *(IB) method is an algorithm for simulating elastic structures immersed in a fluid. The IB method can be used, for example, to simulate blood flow in the heart. Even running on supercomputers, software implementations require on the order of seven CPU-days to simulate one heart beat. The IB method has significant, inherent, fine-grain parallelism available. This parallelism makes it a good candidate for implementation in hardware, such as a* field-programmable gate array *(FPGA). While floating-point arithmetic is possible on FPGAs, fixed-point arithmetic is more efficient and takes less space to implement. This paper presents a study of the accuracy of a fixed-point implementation of the IB method.*

Keywords: Fixed-point arithmetic, round-off error, immersed boundary method, floating-point to fixed-point conversion.

1. Introduction

The *immersed boundary* (IB) method [1] models the interaction of a viscous, incompressible fluid with an immersed elastic boundary or structure. This method was originally introduced to study blood flow in the heart and has been used for numerous other applications, especially in but not limited to biofluid dynamics. In all of its applications, the IB method has been very demanding of computational resources, especially computer time.

One specific application of the IB method is prosthetic heart valve simulation. The IB method requires a significant amount of computation time for modeling even a single heartbeat. Current implementations can take days to perform the calculations needed for a single simulation, in spite of the fact that they are running on supercomputers. If the simulation runs faster, more heartbeats can be simulated in the same amount of computation time. The extra data can aid in detecting anomalies that might be missed in smaller simulations. Given that the analysis can indirectly aid in saving a life by creating a better prosthetic heart valve, a reduction in the time requirement for simulation would be of great benefit.

The question therefore arises whether a hardware implementation would provide significant speedup in comparison to the traditional software implementations that have been done until now. The first step in pursuing a hardware implementation is determining whether or not the algorithm will support a fixed-point implementation [2].

AccelDSP [3] is a tool that ports MATLAB code to a fixed-point *field-programmable gate array* (FPGA) implementation through an automated process. A number of other tools exist [4], [5] for automatically converting floating-point code into fixed-point code. Automated conversion works well for code written specifically for the tool, allowing caveats of the conversion tool to be addressed in advance. The floating-point code is not written with automatic conversion in mind, and would therefore require many changes to be effectively processed by an automatic tool. Thus, manually converting the algorithm and empirically analyzing the data is chosen in favor of automated tools.

Although the technique of manual conversion with empirical analysis is chosen, the conversion is not completely manual. MATLAB's Fixed-Point Toolbox [6] is employed to aid in the design of the fixed-point implementation. The Fixed-Point Toolbox implements some of the basic fixed-point functions. The numerical formats are manually chosen for top-level variables and some of the internal variables. Other internal variables are allowed to grow in word length, based on rules defined within the Fixed-Point Toolbox.

FPGAs are often used to exploit parallelism in order to improve performance [7]–[9]. In one such example, Gu and Herbordt [10] present an FPGA-based engine for molecular dynamics simulations. Their formulation of the problem has much in common with the IB method. In both methods, particles move in a continuous coordinate space, but must interact with computations that are done on a discrete Cartesian grid. The interaction is handled through interpolation using a weighted average. However, both the weighting function itself and also our approach to the computation of the weighting function differ in two main ways. First, Gu and Herbordt use *semi-floating point* [11] calculations. In contrast, a fixed-point implementation of the IB method is studied in present work. Second, in [10], the weighting function is evaluated arithmetically at run time. In contrast, the IB weighting function in the implementation described here uses table look-up. The motivation for both fixed-point representation and table look-up is to simplify the hardware and improve performance. Table look-up also makes it easy to change the weighting function.

Gu and Herbordt demonstrate an $11\times$ speedup [11] with-

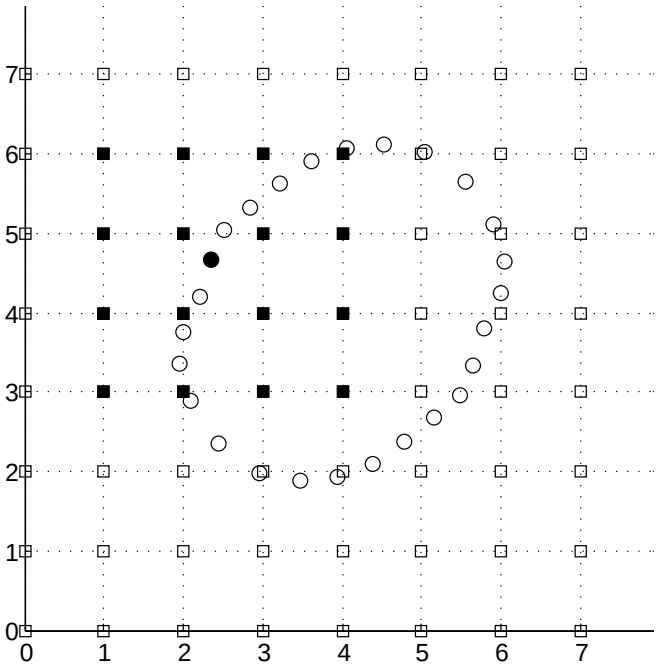

Figure 1: A fiber immersed in a fluid ($N = 8$, $N_b = 26$).

out fixed-point representation or table look-up. Provided that the accuracy can be sufficiently maintained with fixed-point representation, fixed-point arithmetic, and table lookup, a speedup of greater than $11\times$ appears to be feasible for an FPGA-based IB method implementation.

In this paper, a fixed-point implementation of the IB algorithm is presented. The implementation is written in MATLAB, using the Fixed-Point Toolbox [6]. The accuracy of the fixed-point implementation is compared to that of the double-precision floating-point version of the IB method applied to the same physical problem.

Section 2 further describes the IB method. Section 3 explains calculation changes that serve to facilitate an FPGA implementation. Section 4 describes the experimental setup and presents the results. Finally, Section 5 draws conclusions and outlines directions for future work.

2. The Immersed Boundary Method

The IB method models the fluid as a regular Cartesian grid of fluid points. The grid has finite extent, with N points in each dimension. The grid is treated as periodic, such that the last grid point in each row is treated as being adjacent to the first grid point in that row, and similarly in each dimension. The velocity $\mathbf{u}$ of the fluid is stored at each point on the grid. In principle, the grid can have any number of dimensions. Figure 1 depicts a two-dimensional case with $N = 8$. The fluid grid points are indicated by the small squares at the intersections of the grid lines.

The immersed boundary is modeled as a collection of N_b discrete fiber points, each of which stores its current position $\mathbf{X}$. The circles in Figure 1 represent fiber points. A fiber point is *not* restricted to be at the same position as a fluid grid point. The position of a fiber point is stored with significantly greater precision than the fluid grid.

Herein lies the central problem within the immersed boundary method. The velocity of any fiber point should be equal to the velocity of the fluid *at that position*. However, the fluid velocity is only stored on the discrete Cartesian grid. Unless a fiber point happens to be at exactly the same position as a fluid grid point, the fluid velocity at the position of the fiber point is undefined. The solution is to *interpolate* the velocities of the fluid at grid points in a *neighborhood* of each fiber point's position. At each time step, the velocity of each fiber point is set to a weighted average of the velocities of the fluid grid points in its neighborhood. The weights are determined by the fiber point's relative position within the neighborhood. Figure 1 highlights one of the fiber points (the dark disc) and its four-by-four neighborhood of fluid points (the dark squares).

Similarly, the elastic forces exerted on a given fiber point by its neighbors should be imparted to the fluid *at the position of the given fiber point*. The same problem arises that the position of a fiber point may not be equal to that of any given fluid point. Therefore, the forces on fibers are *spread* to the fluid points in the neighborhood of that fiber point. The same neighborhood and the same weights are used as in the interpolation of velocity discussed above.

Figure 2 shows how the state variables of the IB method, the position $\mathbf{X}(t)$ and the velocity $\mathbf{u}(t)$, are updated from time t to time $t + \Delta t$. A second-order accurate *Runge-Kutta* type time-stepping scheme is used. The basic strategy is to define intermediate quantities at time $t + \frac{\Delta t}{2}$ and then to use those intermediate quantities in the update of the state variables from time t to time $t + \Delta t$.

The first step is to *interpolate* the fluid velocity $\mathbf{u}(t)$ around the positions $\mathbf{X}(t)$ of the immersed boundary points to obtain the immersed boundary velocities $\mathbf{U}(t)$, which are then used to update the immersed boundary configuration from $\mathbf{X}(t)$ to $\mathbf{X}(t + \frac{\Delta t}{2})$. This intermediate configuration of the immersed boundary is then used to compute the elastic force density $\mathbf{F}(t + \frac{\Delta t}{2})$ that is applied by the immersed boundary to the fluid. Note that $\mathbf{F}(t + \frac{\Delta t}{2})$ is defined only on the immersed boundary itself.

The next step is to *spread* the force density out onto the fluid grid to obtain the grid-based force density $\mathbf{f}(t + \frac{\Delta t}{2})$. Note that the force-spreading operation is the transpose, or adjoint, of the velocity-interpolation operation.

The next step is to update the fluid velocity under the influence of $\mathbf{f}(t + \frac{\Delta t}{2})$. This is done in two stages, thus generating $\mathbf{u}(t + \frac{\Delta t}{2})$ and then $\mathbf{u}(t + \Delta t)$. The intermediate velocity $\mathbf{u}(t + \frac{\Delta t}{2})$ is then interpolated at the intermediate positions $\mathbf{X}(t + \frac{\Delta t}{2})$ of the immersed boundary points to

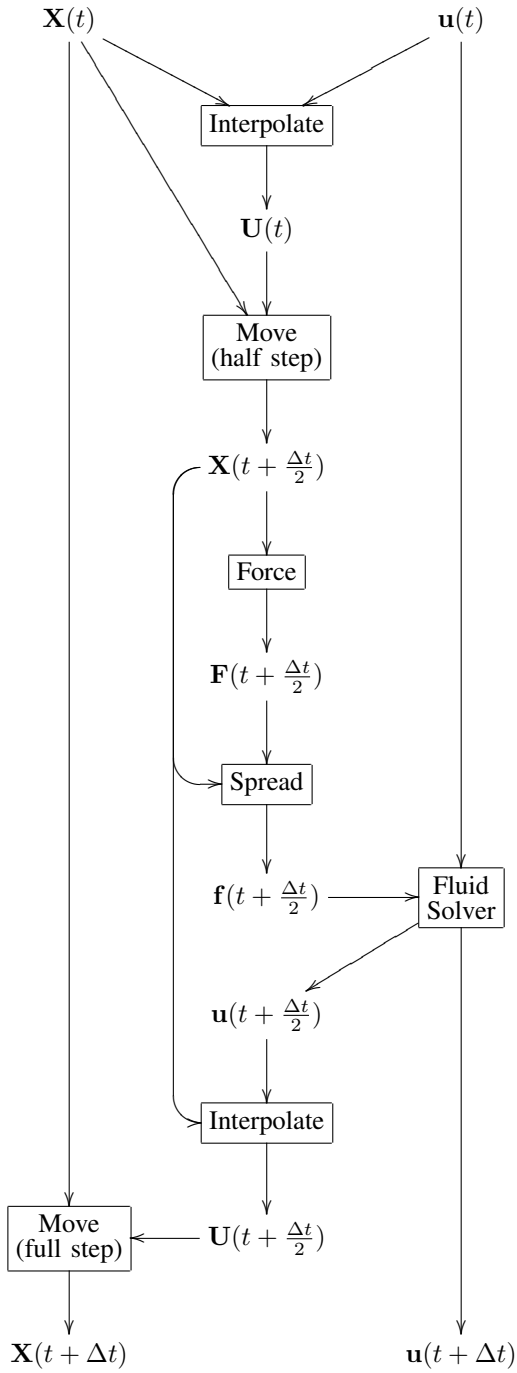

Figure 2: Data flow within one iteration of the IB method.

obtain the intermediate boundary velocity $\mathbf{U}(t+\frac{\Delta t}{2})$, which is used to update the immersed boundary configuration from $\mathbf{X}(t)$ to $\mathbf{X}(t+\Delta t)$. Since both of the state variables have been updated, the time step is complete.

In the remainder of this paper, we use double letters to refer to the intermediate quantities. Thus, for example, the notation $\mathbf{uu}$ will refer to the velocity field at an intermediate time $t+\frac{\Delta t}{2}$, where t is an integer multiple of Δt.

3. FPGA-Centric Optimizations

Several changes are made to the code in order to more readily support an optimized FPGA implementation. These changes also provide benefit for the fixed-point implementation, as they take away operations not directly supported by the Fixed-Point Toolbox, such as square roots and exponential calculations.

3.1 Program Units

The floating-point implementation of the IB method uses physical units. In contrast, the fixed-point implementation uses *program units* as follows. The unit of distance h is the distance between adjacent fluid grid points. The unit of time Δt is one time step of the simulation. The unit of mass m is the mass of the fluid contained in one fluid grid box. This choice of units causes the numerical value of several constant IB parameters to become 1. This is especially convenient for an FPGA, since the original IB equations frequently multiply and divide by such constants. The use of program units thus eliminates several multipliers and dividers.

3.2 Implementing ϕ as a Table

For the two-dimensional case, in program units, the equation for interpolation is given in (1).

$$\mathbf{U}_k = \sum_{\mathbf{x}\in n_k} \mathbf{u}(\mathbf{x})\phi(x_1 - X_{k,1})\phi(x_2 - X_{k,2}) \tag{1}$$

$$n_k = n_{k,1} \times n_{k,2} \tag{2}$$

$$n_{k,i} = \{\lfloor X_{k,i} \rfloor - 1, \ldots, \lfloor X_{k,i} \rfloor + 2\} \tag{3}$$

where:

- $\mathbf{U}_k$ is the velocity of fiber point k.
- $\mathbf{x} = (x_1, x_2)$ is the position of a fluid grid point.
- $\mathbf{X}_k = (X_{k,1}, X_{k,2})$ is the position of fiber point k.
- n_k is the four-by-four neighborhood of fiber point k.
- $\mathbf{u}(\mathbf{x})$ is the velocity of the fluid at position $\mathbf{x}$.
- $\phi(r)$ is a weighting function.

The interpolation function ϕ is given by (4) [1].

$$\phi(r) = \begin{cases} \frac{1}{8}\left(3 - 2|r| + \sqrt{1 + 4|r| - 4r^2}\right), & |r| \le 1 \\ \frac{1}{8}\left(5 - 2|r| - \sqrt{-7 + 12|r| - 4r^2}\right), & 1 \le |r| \le 2 \end{cases} \tag{4}$$

The ϕ function, as defined in (4), involves square roots which are costly to implement in FPGAs and not fully

supported within the Fixed-Point Toolbox. To avoid the square roots, ϕ can instead be pre-calculated and accessed using table look-up. Note that the mathematical dual of interpolate, *i.e.*, the spreading function, uses the same ϕ function. Thus the savings in implementing ϕ as a lookup table are doubled.

Let $s_i = X_i - \lfloor X_i \rfloor$. Thus, s_i, the fractional part of X_i, represents the relative position of fiber point k within its current fluid grid box, along dimension i, in program units. For each dimension i, the following weights must be computed:

$$w_{1,i} = \phi(-1 - s_i) = \frac{1}{2} - \phi(1 - s_i) \tag{5}$$

$$w_{2,i} = \phi(-s_i) = \phi(s_i) \tag{6}$$

$$w_{3,i} = \phi(1 - s_i) \tag{7}$$

$$w_{4,i} = \phi(2 - s_i) = \frac{1}{2} - \phi(-s_i) = \frac{1}{2} - \phi(s_i) \tag{8}$$

(6) and (8) rely on the fact that ϕ is an even function: $\phi(-r) = \phi(r)$. (5) and (8) rely on the following property of ϕ [1]:

$$\sum_{j \text{ even}} \phi(r - j) = \sum_{j \text{ odd}} \phi(r - j) = \frac{1}{2} \tag{9}$$

which ensures that shifting the argument of ϕ by 2 complements its output with respect to $\frac{1}{2}$, provided that the argument remains within the support of ϕ. This observation is used to reduce the memory required for table look-up as follows.

$w_{2,i} = \phi(s_i)$ and $w_{3,i} = \phi(1 - s_i)$ are stored in tables. These two tables can be accessed in parallel. Then $w_{1,i}$ and $w_{4,i}$ are computed as follows: $w_{1,i} = \frac{1}{2} - w_{3,i}$ and $w_{4,i} = \frac{1}{2} - w_{2,i}$. This ensures that the property of (9) is preserved exactly, regardless of the precision of the tables. This method also cuts the storage requirements for ϕ in half, because separate tables are not required for $w_{1,i}$ and $w_{4,i}$.

The table index s_i is simply the fractional part of the fiber-point position X_i. The values stored in the tables are also quantized. Because the range of the function ϕ is $\left[0, \frac{1}{2}\right]$, the table values are unsigned and have no integer part. The current implementation uses the same number of bits for the fractional part of each entry in the table for ϕ, as are used for the fractional part of velocity.

3.3 FFT/IFFT Twiddle Factor Table Lookup

At its core, the fluid solver uses *fast Fourier transforms* (FFTs) and *inverse fast Fourier transforms* (IFFTs). Twiddle factors [12] are the constant complex coefficients used in the FFT and IFFT. These values come at high computational cost, as they are complex exponentials. Since the inputs to the FFT and IFFT are of a constant size $N \times N$, these twiddle factors can be pre-computed and accessed through a lookup table, as done for ϕ in Section 3.2. This is common practice in hardware implementations and has significant performance benefits [13]. The twiddle factors for the FFT versus the IFFT are simply the complex conjugates of one another, but are stored in two separate tables in the current implementation to avoid computing the complex conjugate on each table access during the IFFT.

4. Test Methodology and Results

A two-dimensional slice of a simple generalized cylinder immersed in a fluid is simulated. The fiber-point positions are started in a circular ring configuration. An initial shear fluid-grid velocity is applied to perturb the system, causing the ring to oscillate and rotate. The simulation runs for 400 times steps, which allows the system to return to equilibrium.

While this simulation is less complex than a production-level implementation, the key issues and calculation difficulties of the more complex problem remain intact. The simpler simulation is chosen to reduce the amount of time each simulation requires, as a large number of simulation runs are necessary to provide more meaningful results.

Several stages of experimentation are used in order to find acceptable bit-widths for this problem. At each stage, results are compared to results of the original floating-point implementation stored in a MATLAB data file. The state of the system in this IB computation is represented by an N_b-length vector of fiber-point positions and an $N \times N$ matrix of fluid-grid velocities. All tests are performed with $N = 64$ and $N_b = 202$.

The overall accuracy of the fixed-point implementation is measured by the absolute value of the difference between the fixed-point results and the floating-point results. Acceptable quantitative error depends largely on the application. One percent of a fluid-grid box is chosen as a desired maximum error in this implementation. Qualitatively, acceptable error is determined by observing the output of the simulation for the floating-point and fixed-point implementations on the same graph. When there are far too few bits, the ring is destroyed by the end of the simulation. Bit widths close to the acceptable range show deformities in the ring, but the fiber point positions still roughly resemble a ring. Bit widths equal to, or greater than the acceptable error are a very close visual match to that of the floating-point implementation. The maximum, average and *root mean squared* (RMS) error of these absolute differences is measured for each time step, as well as across an entire simulation run of 400 time steps. These errors are examined in the fiber-point positions, as well as the fluid-grid velocities. The errors in the internal variables are examined as required for debugging and development purposes.

4.1 Operating Range of Variables

The first stage in moving to a fixed-point implementation involves examining the operating range that each of the variables takes throughout the simulation. The goals of this stage are to decide if the quantities in an IB computation

Table 1: Operating range of top-level IB method variables.

Variable	Absolute Non-Zero Minimum	Absolute Maximum	Signed or Unsigned	Units
$\mathbf{X}$	10.7223	53.2777	Unsigned	h
$\mathbf{XX}$	10.7208	53.2792	Unsigned	h
$\mathbf{U}$	1.0483×10^{-06}	0.6385	Signed	$\frac{h}{\Delta t}$
$\mathbf{UU}$	1.8766×10^{-06}	0.6379	Signed	$\frac{h}{\Delta t}$
$\mathbf{FF}$	4.6670×10^{-06}	0.3917	Signed	$\frac{m}{(\Delta t)^2 link}$
$\mathbf{ff}$	1.3307×10^{-14}	0.3895	Signed	$\frac{m}{(h\Delta t)^2}$
$\mathbf{uu}$	1.0956×10^{-18}	0.6943	Signed	$\frac{h}{\Delta t}$
$\mathbf{u}$	1.8725×10^{-19}	0.6944	Signed	$\frac{h}{\Delta t}$

are good candidates for fixed-point representation, and if so, to determine a suitable fixed-point numerical format for each of the variables. We do this first from theoretical considerations for the position and velocity variables. Then we use empirical observations to check these bounds and also extend them to force density variables.

The position $\mathbf{X} = (X_1, X_2)$ of any given fiber point is restricted such that $0 \leq X_1 < Nh$ and $0 \leq X_2 < Nh$. In program units, $h = 1$, and the integer portion of each component of fiber position must lie in the set $\{0, 1, \ldots, N-1\}$. Thus, X_1 and X_2 are unsigned quantities, and each requires $\lceil \log_2 N \rceil$ bits for the integer part. The use of a fixed-point value for position essentially defines a finer grid for fiber points, embedded within the coarse fluid grid. Fiber points are restricted to positions on the finer grid. In program units, the integer portion of position serves as an index into the containing fluid grid box. The fractional portion of position identifies the relative position within that box. Supporting M subdivisions per fluid grid box along each dimension requires $\lceil \log_2 M \rceil$ bits of fractional part for position.

For stability, the IB method requires that a given fiber point cannot skip over any entire fluid grid box in one time step. Thus, the magnitude of each component of the velocity of each fiber point must be less than one fluid grid box per time step: $|U_i| < \frac{h}{\Delta t}$. Since fiber velocity is interpolated from fluid velocity, the same restriction applies to the fluid velocity: $|u_i| < \frac{h}{\Delta t}$. In program units, $h = 1$ and $\Delta t = 1$. Thus, any of U_1, U_2, u_1, u_2 can be represented as a signed, fixed-point number, with no integer part. $\mathbf{U}$ is used to update $\mathbf{X}$ at each time step.

The signedness, the absolute (non-zero) minimum, and the absolute maximum values of each of the variables within the IB method are observed for the complete simulation. It is shown in Table 1 that the important IB quantities have upper bounds in known, moderate ranges. The lower bounds indicate that without a large number of bits, underflows cannot be prevented, particularly for $\mathbf{u}$, $\mathbf{uu}$, and $\mathbf{ff}$. Using the numbers obtained in this experiment, a portion of which are seen in Table 1, it is decided that the numbers are indeed well-suited for a fixed-point implementation.

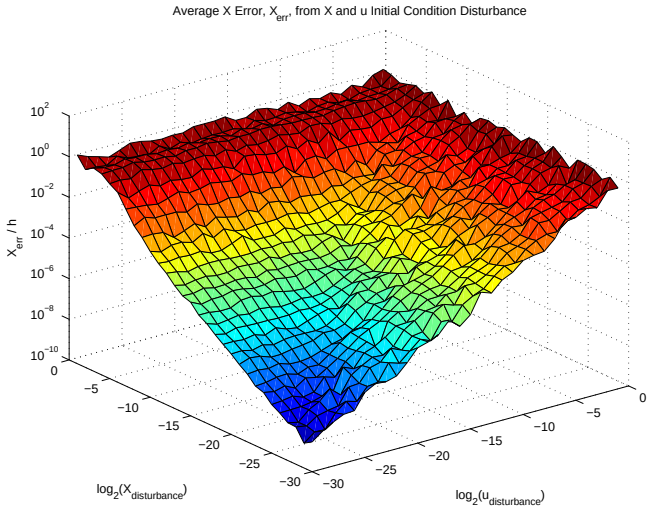

Figure 3: Average $\mathbf{X}$ error from $\mathbf{X}$ and $\mathbf{u}$ initial condition disturbance.

4.2 Initial Condition Disturbance

The second stage in moving to a fixed-point implementation involves examining the response to a perturbation in the initial condition. In this stage, which again takes place within the floating-point implementation, a random disturbance of varying weight is applied to the input state and the propagation of this disturbance throughout a simulation run is observed. The weights applied are in powers of two in order to represent quantization error that will later be introduced by the fixed-point implementation. The goal of this stage is to examine how small errors propagate through the closed-loop system. It also serves as a way to determine the least amount of error that should be expected, given state variables quantized at fractional bit lengths corresponding to the weight of the input disturbance.

It can be seen in Figure 3 that disturbing the initial conditions of the state variables with decreasing fractional bit lengths leads, for the most part, to an increase in overall error. There are instances where increasing the disturbance (which models decreasing the number of available fractional bits) leads to a decrease in error. The non-monotonic nature of the error in response to initial condition disturbance is due to feedback through the closed loop system. Other than at the very high disturbances, these figures do not indicate a significant advantage in choosing a different number of bits for $\mathbf{X}$ and $\mathbf{u}$. The higher disturbances, *i.e.*, those that correspond to using fewer than ten bits, show errors in position greater than 10^{-4} h. While this magnitude of error is not unreasonable, it is high given that the noise is only injected into the initial conditions.

Table 2: Average maximum error for $\mathbf{X}$ and $\mathbf{u}$ under different top-level rounding modes.

Rounding Mode	Average Maximum Error in $\mathbf{X}/h$	$\mathbf{u}/(\frac{h}{\Delta t})$
Nearest	0.0337	0.2535
Round	0.0337	0.2535
Convergent	0.0337	0.2535
Random	0.0743	0.3558
Floor	0.0851	0.2036
Fix	0.1602	0.3625
Ceiling	0.1938	0.5143

4.3 Top-Level Quantization

The third stage in moving to a fixed-point implementation involves top-level quantization of the state variables. This stage quantizes the state variables at each time step, but performs all calculations in floating point. Given that the calculations in this step are performed with much higher resolution, a problem with a particular bit-width at the top level would not likely work in the full fixed-point implementation. Thus this experiment tests the lower boundary of applicable bit widths.

An important observation taken from this step is which rounding mode to use in later testing. In this particular test, the fractional part of $\mathbf{X}$ runs from one to twenty-two bits. The number of bits in $\mathbf{u}$ is equal to the number of bits in $\mathbf{X}$ plus additional bits, ranging from zero to fifteen. The results for each of the rounding methods can be seen in Table 2.

All of the rounding modes listed in Table 2 are documented in [6], with the exception of random roundoff. Random roundoff rounds up with probability equal to the fractional part of the number that is being rounded [14], [15].

The results indicate that there is a modest advantage in the *nearest*, *round*, and *convergent* modes of rounding in comparison to the other methods. Within these three rounding modes, there is no clear advantage to any one of them. The *nearest* mode is therefore chosen for subsequent studies within this paper because of its lower implementation cost [6].

4.4 Full Fixed-Point Implementation

The first three stages took place within the framework of the floating-point implementation. The final stage involves implementing the computation in fixed-point arithmetic. This stage provides the framework within which data can be collected for different bit widths to aid in the design of a future hardware implementation.

With the large potential for accumulation of error in the fluid solver, which relies heavily on FFT and IFFT, a study is performed to determine whether more bits in the fluid solver would aid in decreasing the error accumulation. As seen in Figure 4, adding extra bits to the fluid solver, despite introducing a much more non-monotonic graph, does indeed

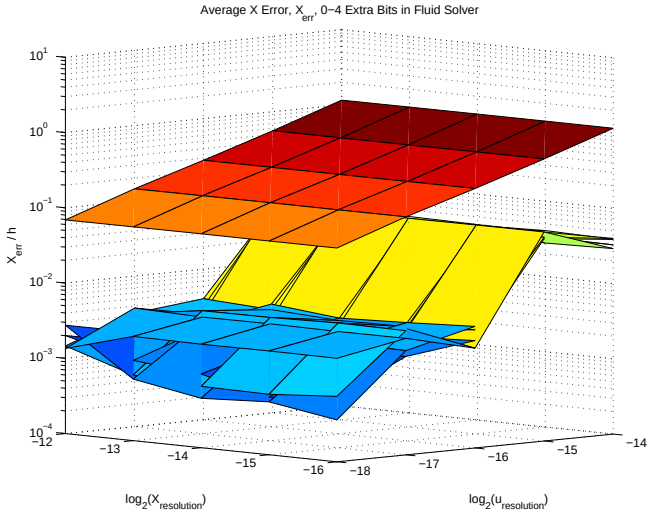

Figure 4: Average $\mathbf{X}$ error for zero through four extra bits in the fluid solver.

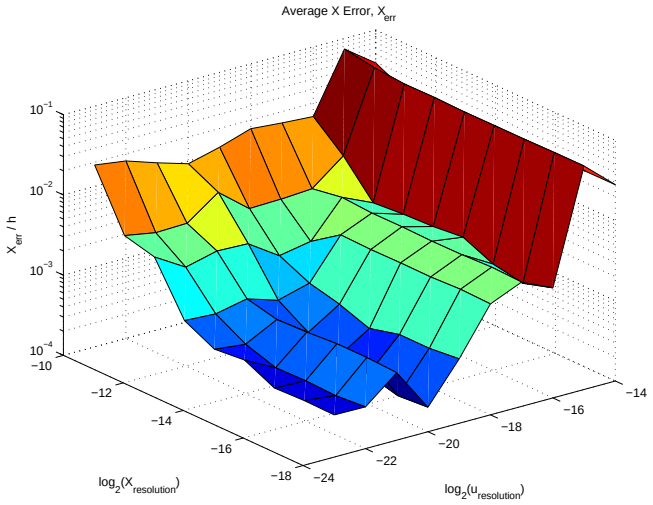

Figure 5: Average $\mathbf{X}$ error in the fixed-point implementation.

aid in decreasing error accumulation. It is also worth noting that most of the benefit comes from the first extra bit; adding more than one additional extra bit does not yield further significant improvement. (The lower three surfaces in Figure 4 are nearly coincident.)

Figure 5 and Figure 6 show the accumulated errors of $\mathbf{X}$ and $\mathbf{u}$, respectively, that occur upon limiting the number of fractional bits of the $\mathbf{X}$-like and $\mathbf{u}$-like variables. The $\mathbf{X}$-like variables are those that are always positive and bounded between by the domain, zero to N, *i.e.*, the position variables. The $\mathbf{u}$-like variables are those that are signed numbers, bounded between negative one and one, *i.e.*, the velocity and force density variables. These tests are run using one extra bit within the fluid solver. Lower bit widths than

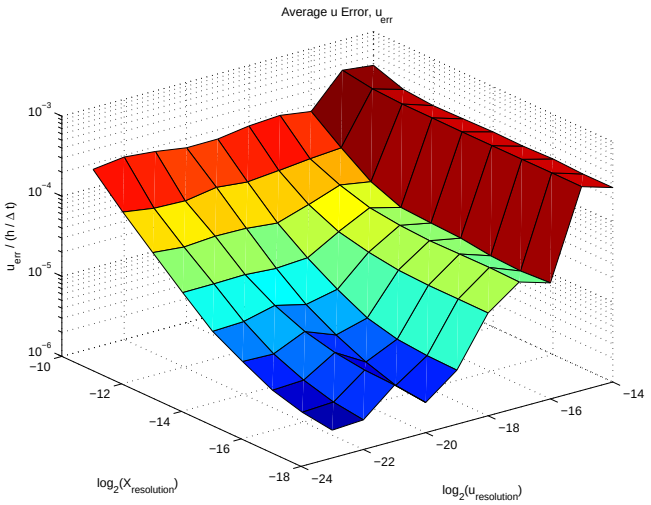

Figure 6: Average $\mathbf{u}$ error in the fixed-point implementation.

those shown are not practical, as they result in overflows that quickly lead to catastrophic error.

5. Conclusions and Future Work

The end result of this study is the discovery that the IB method is indeed well-suited for a fixed-point implementation, at least in terms of accuracy. From the tests in Section 4.4, it is determined that at least ten bits are required for the fractional part of the $\mathbf{X}$-like variables, and at least fourteen bits are required for the fractional part of the $\mathbf{u}$-like variables. There is a large improvement of error in moving to twelve bits for the $\mathbf{X}$-like variables, and sixteen bits for the $\mathbf{u}$-like variables. Under these bit widths, the average $\mathbf{X}$ error after 400 time steps does not exceed one percent of a fluid grid box.

As noted in Section 3.2, the ϕ lookup table in the current implementation uses all of the bits of the fractional part of the fiber-point position to index into the table. This table quickly grows in size as the fractional bit width is increased, as each additional bit doubles the size of the table. These bits may not all be necessary to determine the position weight to a sufficiently accurate value. A future study is intended to determine whether the lookup table can be further compacted without unreasonable loss of accuracy.

Ongoing research seeks to implement the IB method on reconfigurable hardware, specifically, an FPGA. Moving forward, optimization and parallelization will need to be used in order to obtain a reasonable performance comparison. With a parallelized implementation of the IB method meaningful performance comparisons can be made to production IB code — including existing, parallel, FORTRAN and C++ implementations that run on supercomputers. A performance-based study can use the results of this paper as both a starting point and a way to check results. Once a complete hardware implementation is established, evaluation on the trade-offs in cost and speed can be studied. Using such comparisons as a guide, optimizations can also be explored that further adapt the IB method to take advantage of the unique opportunities of reconfigurable computing.

References

[1] C. S. Peskin, "The immersed boundary method," *Acta Numerica*, vol. 11, pp. 479–517, 2002.

[2] C. Inacio and D. Ombres, "The DSP decision: fixed point or floating?" *IEEE Spectr.*, vol. 33, no. 9, pp. 72–74, 1996.

[3] P. Banerjee, M. Haldar, A. Nayak, V. Kim, V. Saxena, S. Parkes, D. Bagchi, S. Pal, N. Tripathi, D. Zaretsky, R. Anderson, and J. R. Uribe, "Overview of a compiler for synthesizing MATLAB programs onto FPGAs," *IEEE Trans. VLSI Syst.*, vol. 12, no. 3, pp. 312–324, 2004.

[4] D. Menard, D. Chillet, F. Charot, and O. Sentieys, "Automatic floating-point to fixed-point conversion for DSP code generation," in *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*. New York, NY, USA: ACM, 2002, pp. 270–276.

[5] C. Shi and R. W. Brodersen, "Automated fixed-point data-type optimization tool for signal processing and communication systems," in *DAC '04: Proceedings of the 41st annual Design Automation Conference*. New York, NY, USA: ACM, 2004, pp. 478–483.

[6] *Fixed-Point Toolbox User's Guide*, 3rd ed., The MathWorks, Inc., Sep. 2009. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/pdf_doc/fixedpoint/FPTUG.pdf

[7] K. Sano, O. Pell, W. Luk, and S. Yamamoto, "FPGA-based streaming computation for Lattice Boltzmann method," in *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, 2007, pp. 233–236.

[8] S. R. Alam, P. K. Agarwal, M. C. Smith, J. S. Vetter, and D. Caliga, "Using FPGA devices to accelerate biomolecular simulations," *IEEE Computer*, vol. 40, no. 3, pp. 66–73, 2007.

[9] L. Zhuo and V. K. Prasanna, "High performance linear algebra operations on reconfigurable systems," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 2.

[10] Y. Gu and M. C. Herbordt, "FPGA-based multigrid computation for molecular dynamics simulations," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*. IEEE Computer Society, 2007, pp. 117–126.

[11] Y. Gu, T. Vancourt, and M. C. Herbordt, "Explicit design of FPGA-based coprocessors for short-range force computations in molecular dynamics simulations," *Parallel Comput.*, vol. 34, no. 4-5, pp. 261–277, 2008.

[12] W. M. Gentleman and G. Sande, "Fast Fourier transforms: for fun and profit," in *AFIPS '66 (Fall): Proceedings of the fall joint computer conference*. New York, NY, USA: ACM, Nov. 1966, pp. 563–578.

[13] I. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing," *IEE Proceedings - Vision, Image, and Signal Processing*, vol. 152, no. 3, pp. 283–296, 2005.

[14] G. E. Forsythe, "Reprint of a note on rounding-off errors," *SIAM Review*, vol. 1, no. 1, pp. 66–67, 1959.

[15] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, "Programmable active memories: Reconfigurable systems come of age," *IEEE Trans. VLSI Syst.*, vol. 4, no. 1, pp. 56–69, 1996.