# Amenability of Multigrid Computations to FPGA-Based Acceleration[*]

Yongfeng Gu Martin C. Herbordt
Department of Electrical and Computer Engineering
Boston University; Boston, MA 02215

## 1 Introduction

As operating frequencies of high-end processor chips continue to be limited by heat/power constraints, alternative architectures are receiving much attention. One of these, the FPGA, enables developers to create custom processors for their own particular applications. The challenge is that for an application to be amenable to FPGA acceleration, it must have certain characteristics. Their complete enumeration is beyond the scope of this abstract, but applications dominated by convolutions of large data sets (images, grid-based physical simulations) generally have them. The opposite is also true: other characteristics, such as a heavy reliance on high-precision floating point arithmetic, generally means the application will not do well with FPGAs.

We have recently studied an application, multigrid computation of Poisson's Equation [2], that has characteristics of both types: it is dominated by convolutions, but also heavily reliant on high-precision floating point. We were able to obtain a speed-up of roughly 5× over a tuned production serial version, but only after extensive optimization. We regard this result as generally neutral as an indication of the amenability of multigrid to FPGA acceleration.

Multigrid, however, is both an important and a diverse application; it could perhaps be better termed a *family* of applications, with many parameters whose variation has a substantial impact on performance. Our objective here is to study the impact of varying these parameters on FPGA acceleration.

## 2 Multigrid Applications

Many important computations can be executed by discretizing to a grid and iteratively performing operations in all neighborhoods (see, e.g. [5] for an introduction). Multigrid can improve the convergence rate by using a hierarchy of discretizations. The following procedure is commonly followed, as shown in Figures 1 and 2: (i) at each level down the hierarchy, solve the part of the problem appropriate for the frequency at that level and then anterpolate to the next coarser level, (ii) at the coarsest level, compute a direct solution, (iii) at each level back up the hierarchy, integrate the current solution with the correction previously computed at that level, and then interpolate to the next finer level.

A key point here is that multigrid computations span a wide range of computational complexity:

- How large is the overall system to be solved? This is important since the computation on finest grid is generally the bulk of the work. How large is the finest grid? How many dimensions does it have?
- What is the complexity of the computational kernel? Is it a simple multiply and add, or something substantially more complex?
- How large is the kernel?
- Is relaxation required? For how many iterations?
- What precision is required?
- Is floating point required?

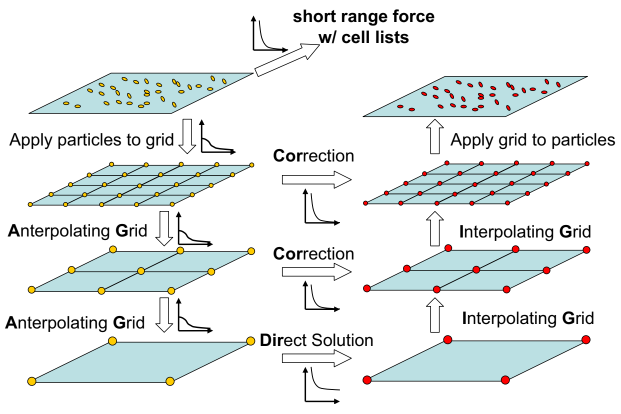

Figure 1: **Schematic of the application of multigrid for solving Poisson's equation to derive electrostatic forces from charge distributions.**

For solving Poisson's equation ($\bigtriangledown^2\Phi = \rho$), we use two levels of 3D grids, $28^3$ and $17^3$. Functions are computed with a $13^3$ kernel. Arithmetic precision is 35-bit; operations are a specially tuned floating point [3]. Because the solution is known, no relaxation steps need to be performed. In our implementation, we can perform 64 pipelined floating point operations simultaneously for a throughput of 6 GFLOPS and a speed-up of roughly 5×.

The problem of image diffusion [1, 4] can be expressed in a general form as follows: $A \cdot X = B$, where $X$ is the original image, $A$ is the diffusion effect, and $B$ is the corrupted image to be corrected. The major computation steps are shown in Figure 2: (i) on the finest grid, $\widehat{X}$, the initial guess of $X$, is relaxed for several iterations; (ii) the residue is passed to the next coarsest level to compute the error correction; (iii) on the following levels, the residue is relaxed with the diffusion operator $A$ and passed to the next coarsest level;

[*]This work was supported in part by the NIH through award #RR020209-01. Web: http://www.bu.edu/caadlab. Email: {maplegu|herbordt}@bu.edu

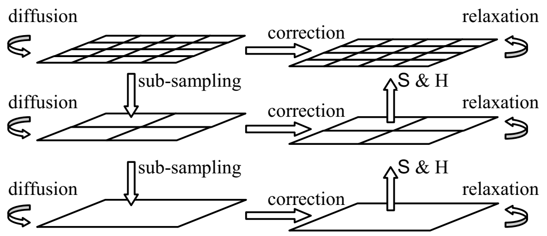

Figure 2: **Schematic of the application of multigrid to linear diffusion for processing images.**

(iv) at the coarsest level, the error correction is set to 0 and is interpolated back through the successively finer grid levels. The parameters depend on the application, but obviously this method can be applied to images of various sizes, dimension, and precision; and use kernels of various sizes.

## 3 Methods and Results

For our experiments, serial reference codes were run on a base system consisting of a 2.4GHz Xeon CPU; code was compiled with Microsoft Visual C++ using full optimization. The accelerated versions were written in VHDL, synthesized using Synplicity tools, and run on a Xilinx VirtexII Pro 100 FPGA.

The common settings for linear image diffusion are as follows: the image size is $256 \times 256$ (or $256^3$ for 3D images); the size of the relaxation operator, such as Gauss-Seidel is $3 \times 3$ (or $3^3$ for 3D); relaxation iterates twice per level; there are 3 grid levels. We simulate a single pass; although multiple passes would be performed in a production application, this normalizes identically to the accelerated code and so can be ignored. The major remaining design parameters are data type and size of the diffusion operator. Except for the larger 3D cases (described below), we use a single basic design. As a consequence, some of the smaller designs show less speed-up than could be obtained if multiple copies were run in parallel.

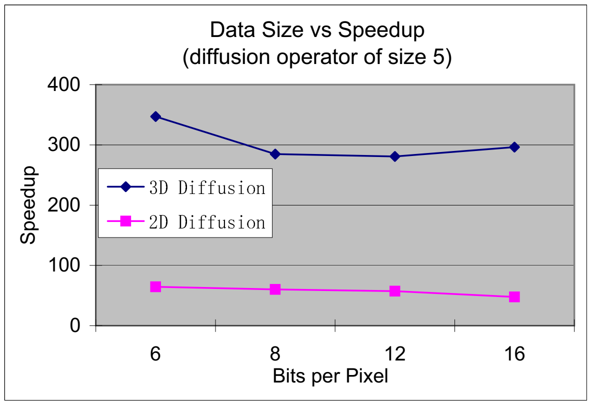

Figure 3: **Graph shows speed-up versus data type size.**

The first result (see Figures 3 and 4) is that the speed-ups, 50× to 1200×, are substantially larger than were obtained by the floating point dominated Poisson solver. There is no doubt that FPGAs are cost effective in all cases here. The second observation is the relationship between speed-up and problem size (2D versus 3D): for the larger problem size, there is more potential parallelism, which clearly can be exploited efficiently.

Figure 3 shows the relationship between speedup and data size. The 2D case shows little variation in operating frequency and the speed-up is almost constant. In the 3D case, the larger data types cause high enough chip utilization to affect the critical path delay.

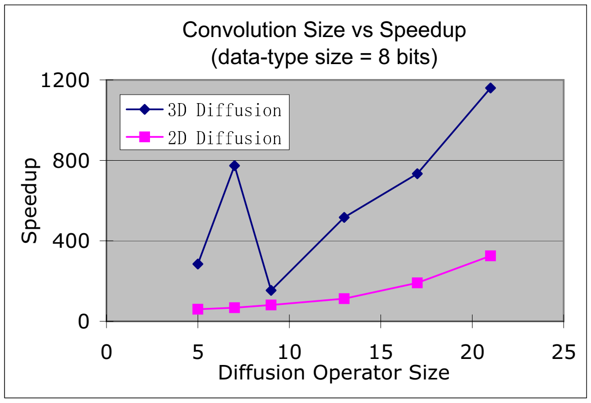

Figure 4: **Graph shows speed-up versus diffusion operator size.**

Figure 4 shows the relationship between speedup and the size of the diffusion operator. In the 2D cases, as the diffusion operator size increases, more parallelism can be applied and speedup increases accordingly. In the 3D cases, the same trend occurs for the first two data points, where we implement $5^3$ and $7^3$ convolvers, respectively. For larger operators, however, the convolvers no longer fit on chip. We therefore use the less efficient procedure of performing 2D convolutions and assembling the results. By the time we reach the last data point, $21^3$, we again run out of chip resources (hard multipliers). More partitioning would be required for still larger operators.

## References

[1] Acton, S. Multigrid anisotropic diffusion. *IEEE Transactions on Image Processing 7*, 3 (1998), 280–291.

[2] Gu, Y., and Herbordt, M. C. FPGA-based multigrid computations for molecular dynamics simulations. In *Field Prog. Custom Computing Machines* (2007).

[3] Gu, Y., VanCourt, T., and Herbordt, M. C. Improved interpolation and system integration for FPGA-based molecular dynamics simulations. In *Proc. Field Programmable Logic and Applications* (2006), pp. 21–28.

[4] Snyder, W., and Qi, H. *Machine Vision*. Cambridge University Press, 2004.

[5] Yavneh, I. Why multigrid methods are so efficient. *Computing in Science and Engineering 8* (2006), 12–22.