Amenability of Multigrid Computations to FPGA-Based Acceleration*

Yongfeng Gu **Martin Herbordt**

Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University
http://www.bu.edu/caadlab

*This work supported in part by the U.S. NIH/NCRR

BOSTON UNIVERSITY

Motivation …

Historically …

FPGAs get fantastic performance because …

- Massive potential parallelism (100x … 1000x … 10000x)
- High utilization (50% and up …)

For applications with certain characteristics …

- Small, regular cores
- Modest data types

If you are willing to deal with high sensitivity of performance to quality of implementation …

- Consider underlying hardware when mapping problem
- Program in non-standard languages

Result → frequent reports of speed-ups in the hundreds

Motivation …

Lately …

Reported speed-ups have become more modest, with ***low single digits frequently being reported …***

Why? Some hypotheses →

- More ambitious applications
  - Large codes in established systems
  - HPC: large, complex, data types
- More realistic reporting
  - end-to-end numbers
  - production reference codes
- More "ambitious" development tools
- FPGA stagnation for two generations (4 years)
  - Smaller chips
  - Fewer "hard" components: Block RAMs, multipliers

The Questions …

As …

- Hardware offers more modest potential acceleration
- Applications get more realistic

1. What applications are amenable to FPGA acceleration?
   - *Metric: at least 5x, with 50x preferred*

2. What are the characteristics of these applications?

Motivation for this study …

We've recently implemented **multigrid** to solve Poisson's equation for MD force computation …

Application characteristics:

- Depends heavily on convolutions 😊
- Requires significant precision ☹

Result → Borderline cost-effective performance

Observation → Multigrid is a complex family of applications with a large parameter space

Question: What parts of the multigrid application space are amenable to FPGA acceleration?

Outline

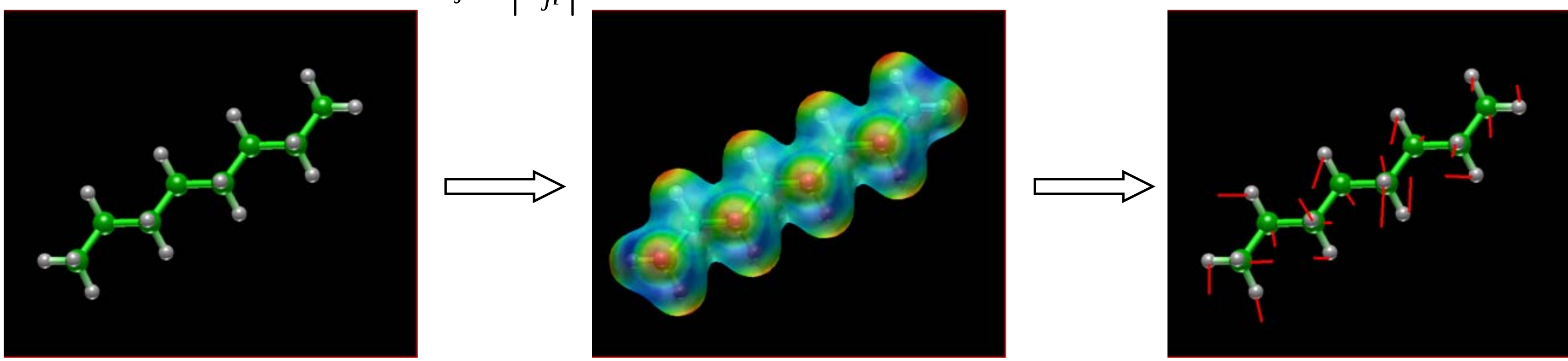
- Introduction
- **Multigrid: Motivation and overview**
- Multigrid for computing the Coulomb force
- Multigrid for computing linear diffusion
- Discussion

Coulomb Force Computation in Molecular Dynamics

1. Sum charge contributions to get Potential Field V^{CL}

$$V_i^{CL} = \sum_{j \neq i} \frac{q_j}{|r_{ji}|}$$

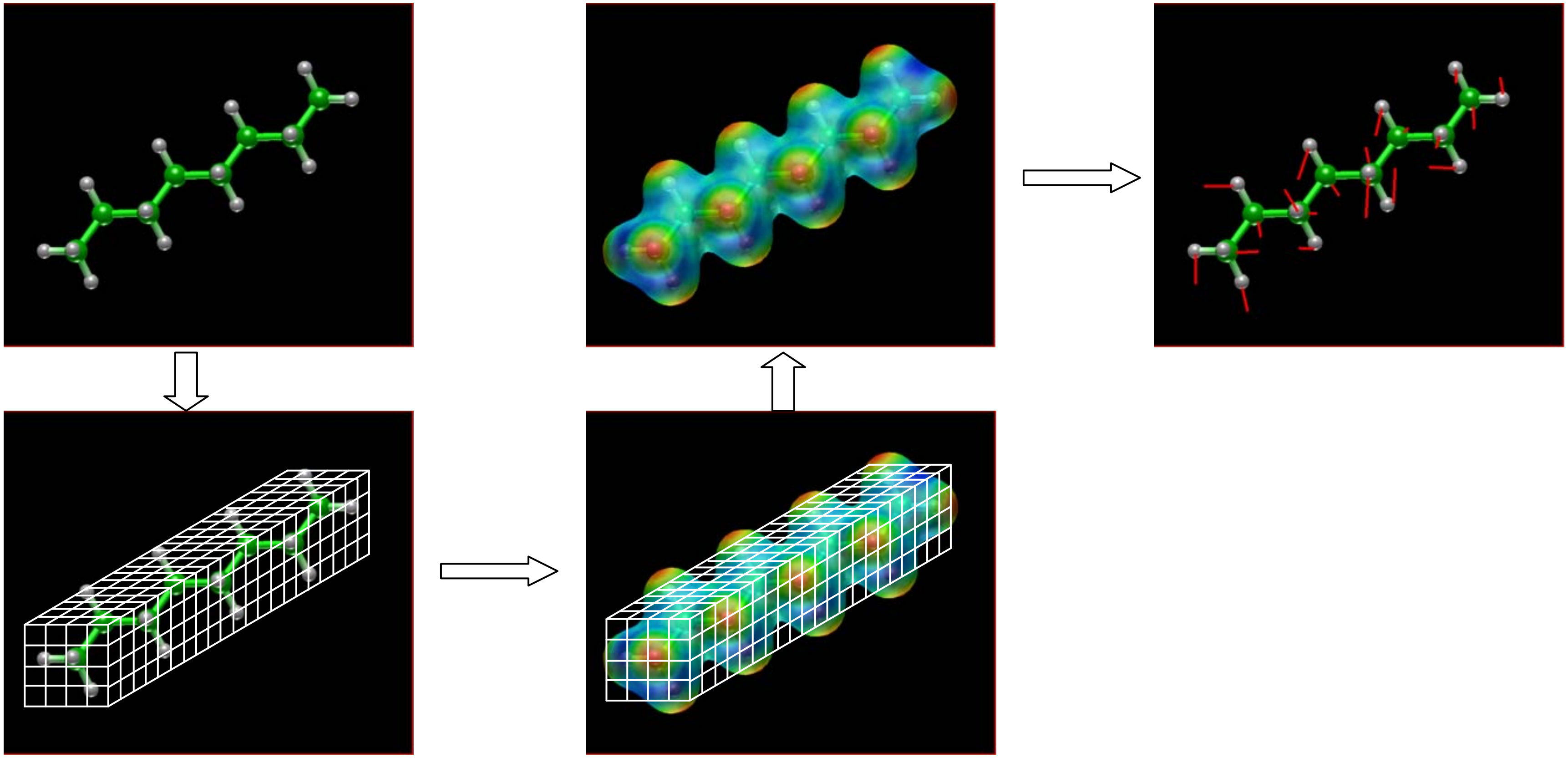
2. Apply Potential Field to particles to derive forces



Picture source: http://core.ecu.edu/phys/flurchickk/AtomicMolecularSystems/octaneReplacement/octaneReplacement.html

Problem: summing the charges is an all-to-all operation

Compute Coulomb Force with 3D grids



Good news: Applying force from 3D grid to particles is O(N)!

Bad news: … as the grid size goes to ∞ !!

Computing the Coulomb Force w/ 3D Grids – Intuition

1. Apply charges (arbitrarily distributed in 3-space) to a 3D grid
   - To apply each charge to the entire grid is impractical, but required by finite spacing, so …
   - apply to as many points as practical initially, and then correct in step 2.
     - E.g., to surrounding 8 grid points in circumscribing cube, to surrounding 64 grid points for larger cube, …

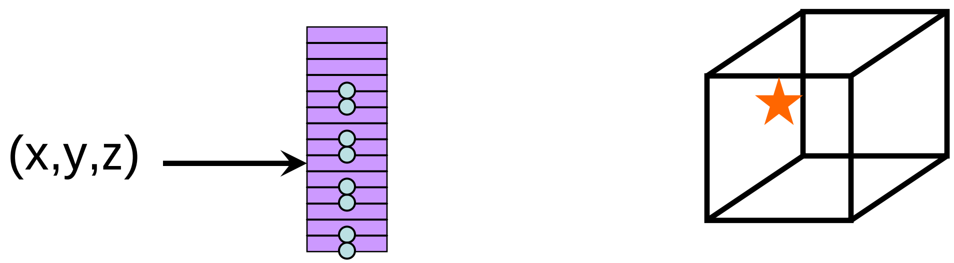
2. Convert *charge density grid* to *potential energy grid*
   - Solve Poisson's equation … $\nabla^2 \Phi = \rho$

3. Convert potential on 3D grid to forces on particles (arbitrarily distributed in 3-space)

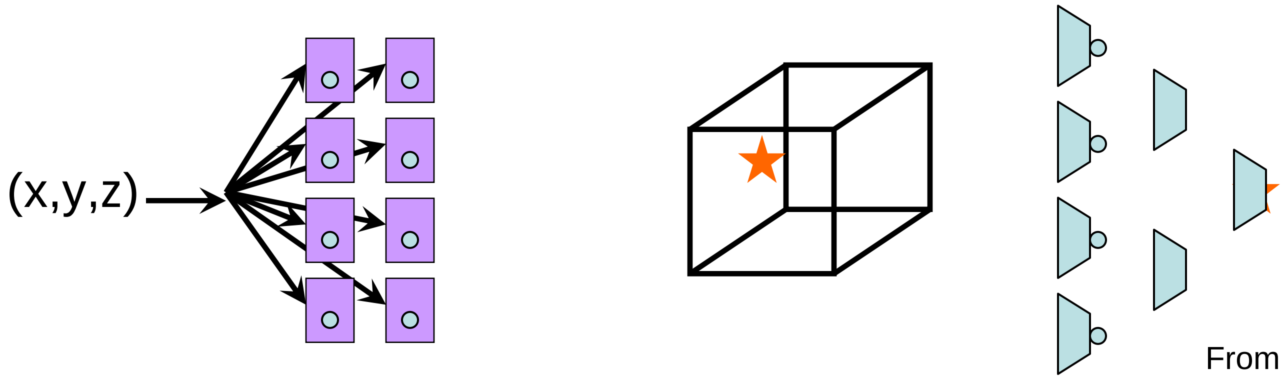
Particle-Grid (1) & Grid-Particle (3) Map Really Well to FPGAs …

Example: Trilinear Interpolation

- SW style: Sequential RAM access



- HW style: App-specific interleaving

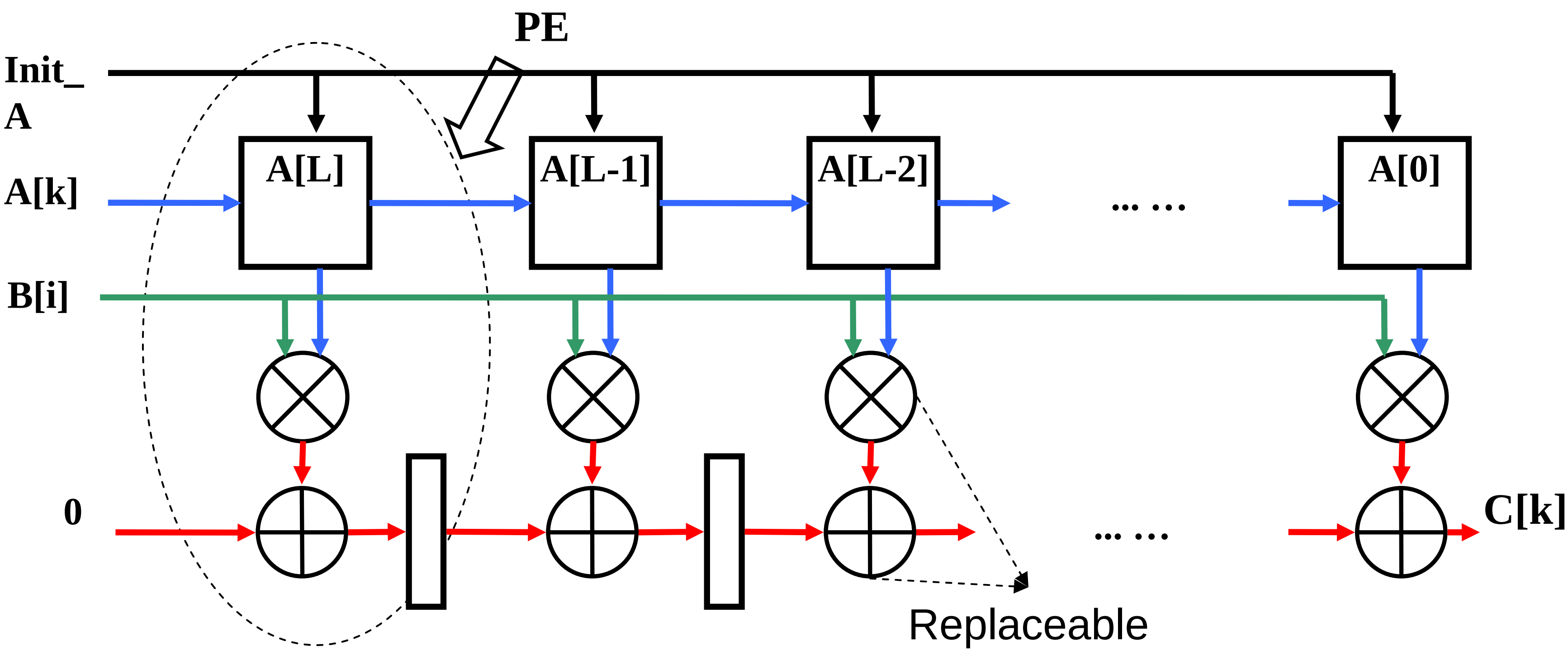


From VanCourt, *et al.* FPL06

3D Grid-Grid (2) also maps really well to FPGAs …

- Operations on grid are mostly convolutions.
- MAC can be replaced with arbitrary operations

1D Convolution Systolic Array (well-known structure)

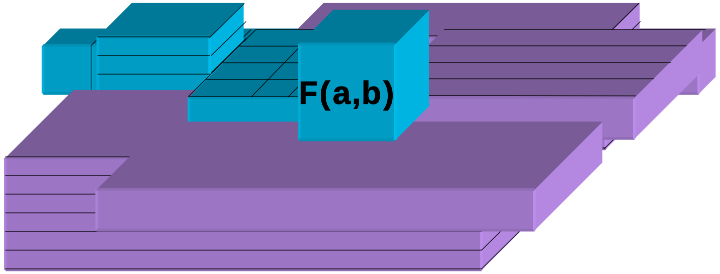


BOSTON UNIVERSITY

3D Grid-Grid (2) also maps really well to FPGAs …

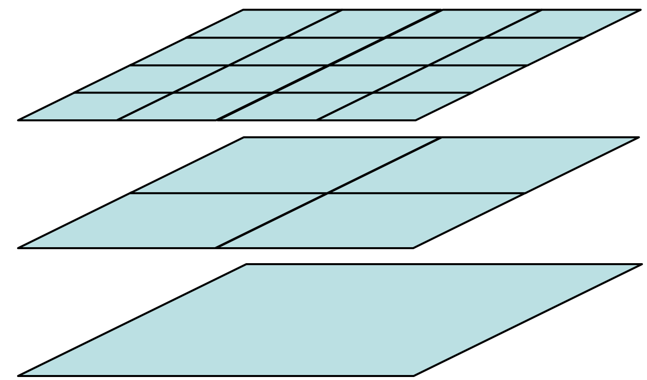
Example: 3D Correlation

- Serial processor: Fourier transform $\mathcal{F}$
  - $A \otimes B = \mathcal{F}^{-1}(\mathcal{F}(A) \times \mathcal{F}(B))$
- FPGA: Direct summation
  - RAM FIFO

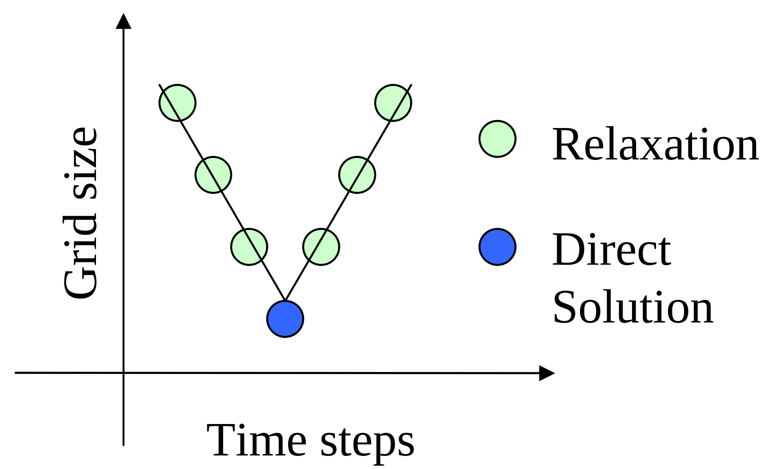


Multigrid Method

- Basic Ideas
  - Computation in discrete grid space is easier than in continuous space
  - Solution at each frequency can be found in a small number of steps per grid point
  - Successively lower frequency components require geometrically fewer computations



- V-Cycle
  - The down and up traversal of the grid hierarchy is called a V-cycle



Multigrid Method, cont.

– The V-Cycle is constructed by a series of recursive calls on every grid level, from the finest to the coarsest.

– On every level (l), there are 9 steps:

1. If this is coarsest grid,

   solve $L_l * u_l = q_l$ and return u_l
2. $u_l = \text{Relax0}(u_l, q_l, l)$
3. $r_l = q_l - L_l * u_l$
4. $q_l+1 = A^{l+1}_l * r_l$
5. $u_l+1 = 0$
6. $u_l+1 = \text{V-Cycle}(u_{l+1}, q_{l+1}, l+1)$
7. $u_l = u_l + I_l^{l+1} * u_{l+1}$
8. $u_l = \text{Relax1}(u_l, q_l, l)$
9. Return u_l

Initial Guess

After Relaxation

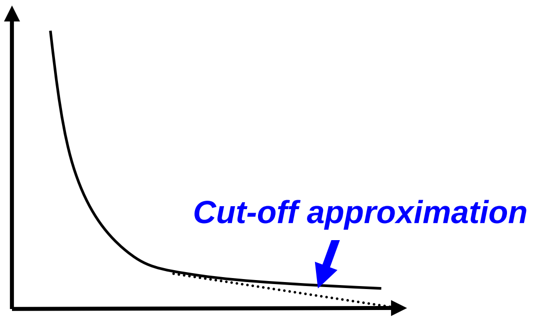
After Correction

BOSTON UNIVERSITY

Outline

- Introduction
- Multigrid: Motivation and overview
- **Multigrid for computing the Coulomb force**
- Multigrid for computing linear diffusion
- Discussion

Multigrid for Coulomb Force



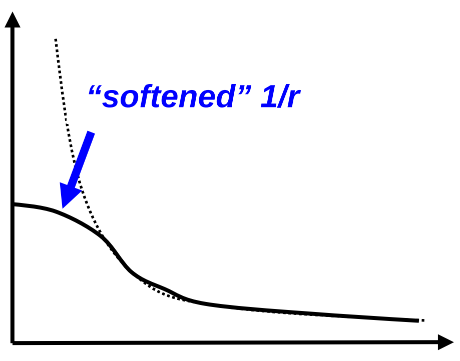
Difficulties with Coulomb force:

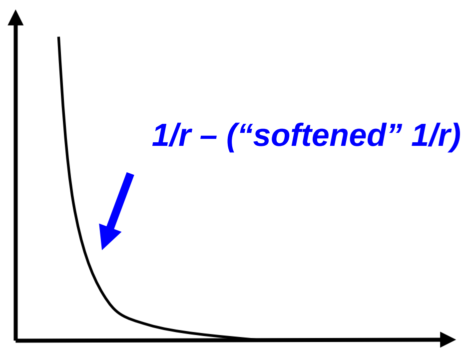
- converges too slowly to use cell lists
- cut-off is not highly accurate

Idea:

- split force into two components →
  - fast converging part that can be solved locally
  - the rest, a.k.a. "the softened part"

doesn't this just put off the problem?





Another Idea:

- *pass "the rest" to the next (coarser) level (!)*
- *keep doing this until the grid is coarse enough to solve directly (!!)*

Multigrid for Coulomb Force

- Potential is split into two parts with a smoothing function $g_a(r)$:

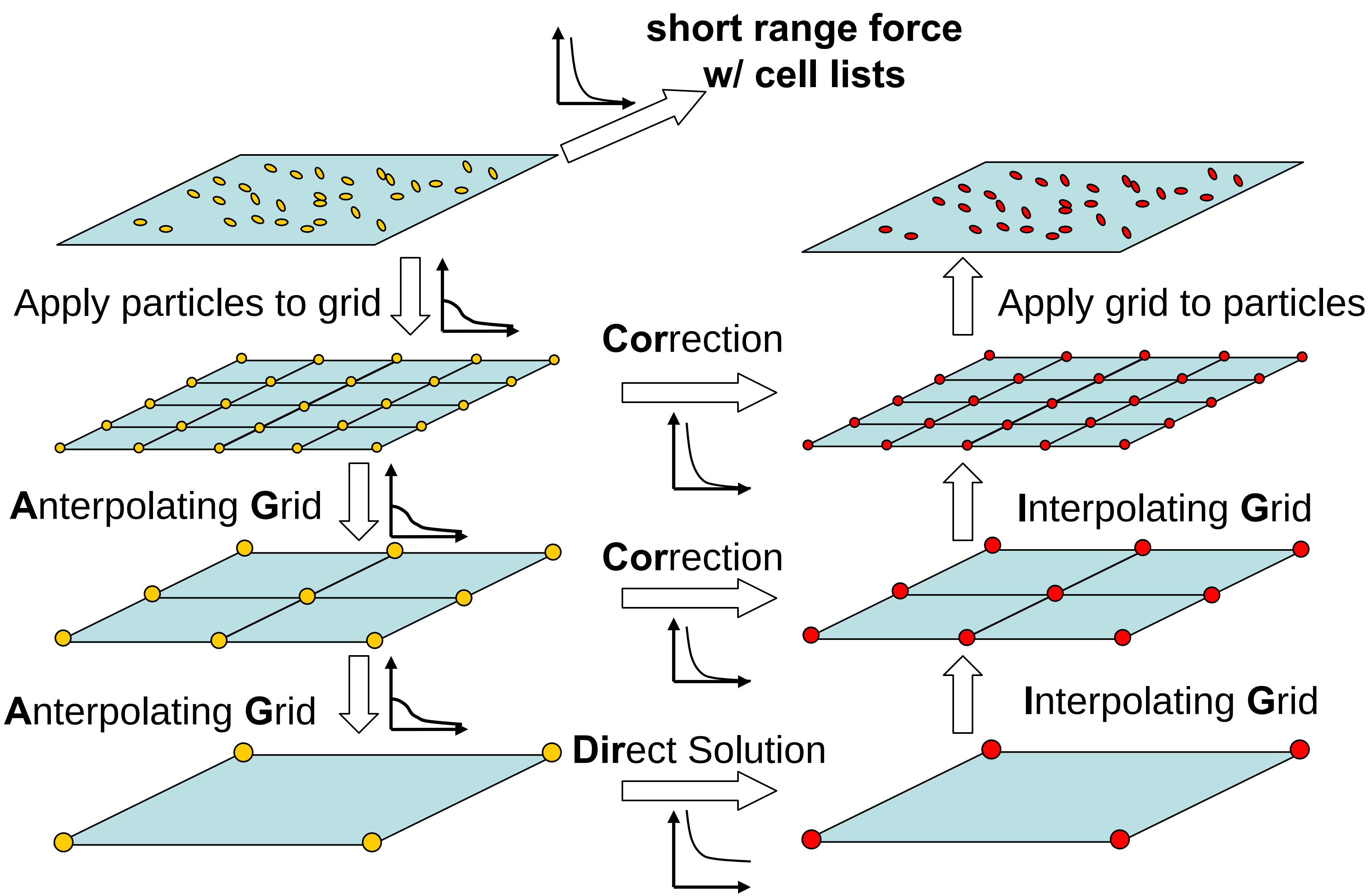
$$V_i^{CL} = \sum_{j \neq i} \frac{q_j}{|r_{ji}|} \quad \Longrightarrow \quad \frac{1}{r} = (\frac{1}{r} - g_a(r)) + g_a(r)$$

- Only the long range part $g_a(r)$ is computed with Multigrid Method
- $g_a(r)$ is recursively approximated with another smoothing function $g_{2a}(r)$:

$$g_a(r) = (g_a(r) - g_{2a}(r)) + g_{2a}(r)$$

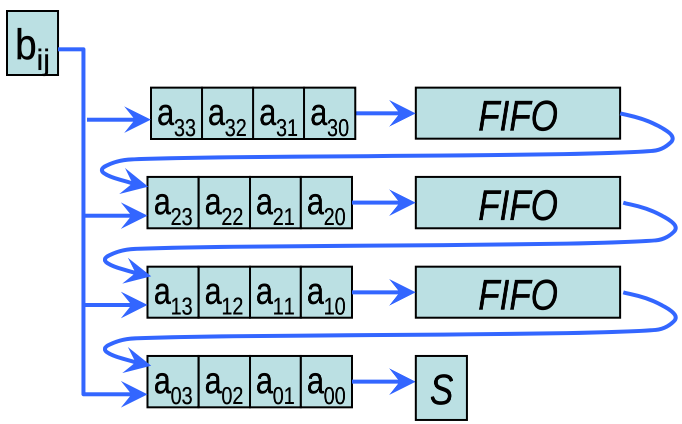
- $g_a(r) - g_{2a}(r)$, the correction, is calculated on the current level grid,
- $g_{2a}(r)$ is approximated on coarser grids.
- If the grid is small enough, $g_a(r)$ is computed directly

Multigrid for Coulomb Force

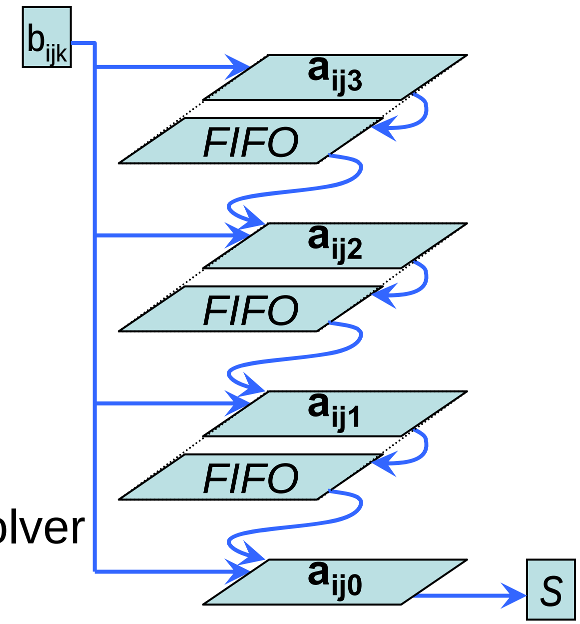


Systolic Convolver

- Grid-Grid Convolver
  - Systolic array provides huge bandwidth and parallelism.
  - BRAMs are efficient to construct FIFOs.
  - The critical resource in this convolver is hardware multipliers required in every PE.



2D Convolver



3D Convolver

The 3D convolver is constructed with 2D and 1D convolvers in series.

Grid-Grid Details

- For the models studied, the following configuration has good accuracy:
  - 2 Grids: Fine ➔ 28 x 28 x 28
    Coarse ➔ 17 x 17 x 17
  - Grids convolved with 10 x 10 x 10 kernels
  - Coarse grid solved directly, i.e. grid charges are integrated to obtain grid potentials (all-to-all)

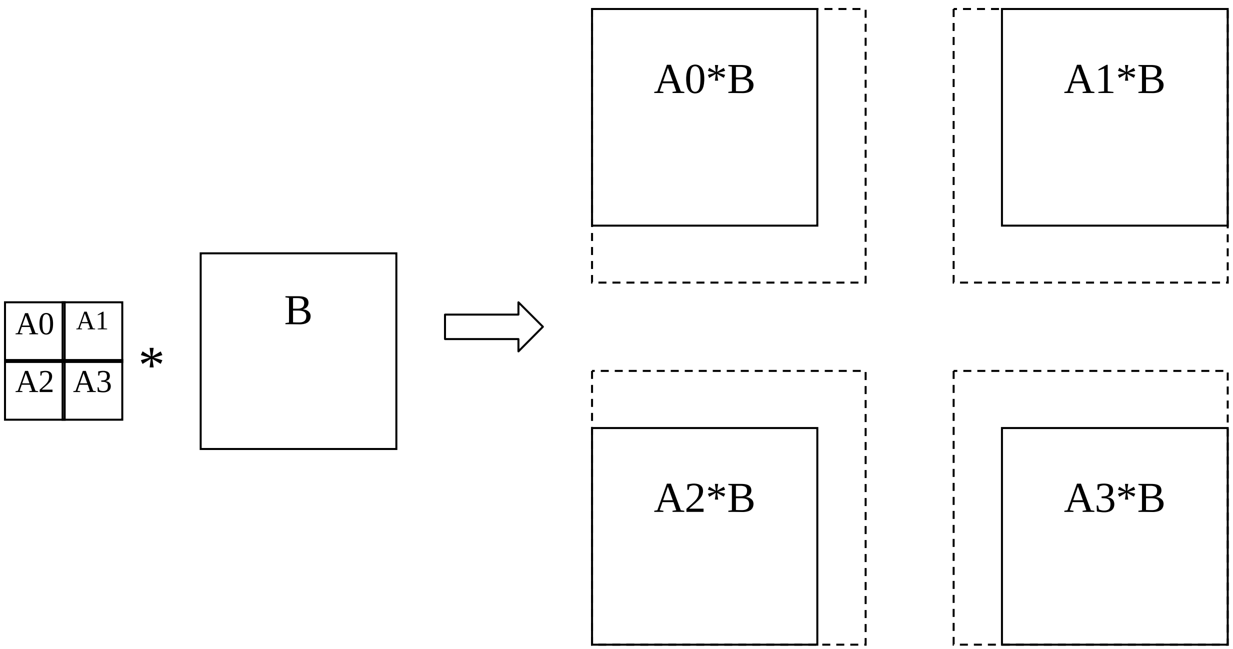
Why no more grids?

Next coarser grid would be 12 x 12 x 12 and smaller than convolution kernel

Handling Large Convolutions

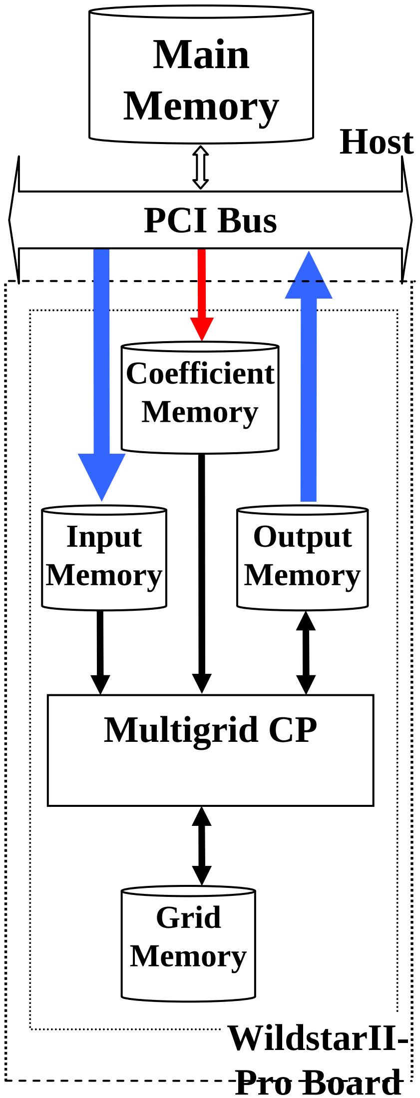
Problem: only 64 convolution units fit on chip (4 x 4 x 4)

So, the convolution must be cut into pieces and assembled…



Implementation

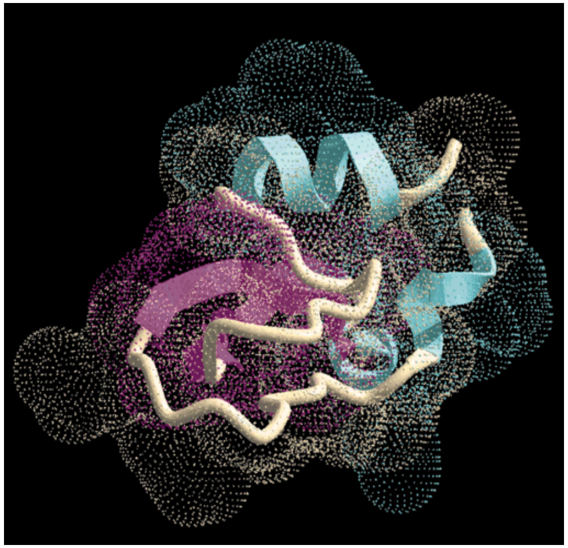
- HW Platform:
  - Annapolis Microsystems Wildstar II Pro PCI Board
  - Xilinx Virtex-II Pro VP70 -5 FPGA
  - FPGA clocks at 75MHz
- Design Flow:
  - VHDL using Xilinx, Synplicity, and ModelSim design tools
- System Specification
  - Capable of 256K particles of 32 atom types
  - Cell-lists for short range force computation
  - 35-bit precision semi floating point



Results – Validation

Both SW only and accelerated codes were evaluated …

- SW only: double precision floating point (53 bit precision)
- Accelerated: 35-bit precision semi floating point



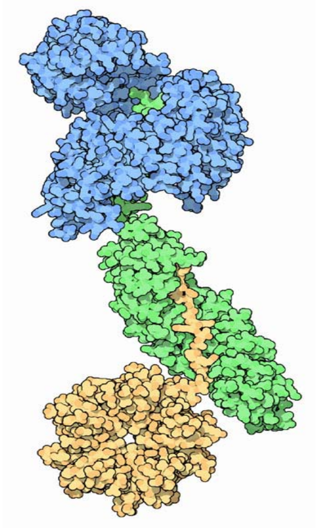
- Model:
  - 14,000 particles
    - bovine pancreatic trypsin inhibitor in water
  - 10,000 time steps

  (similar results with larger model)

- Energy Fluctuation:
  - Both versions both have relative rms energy fluctuations ~$5*10^{-4}$

$$\frac{\sqrt{\left|\left\langle E^2 \right\rangle - \left\langle E \right\rangle^2\right|}}{\left|\left\langle E \right\rangle\right|}$$

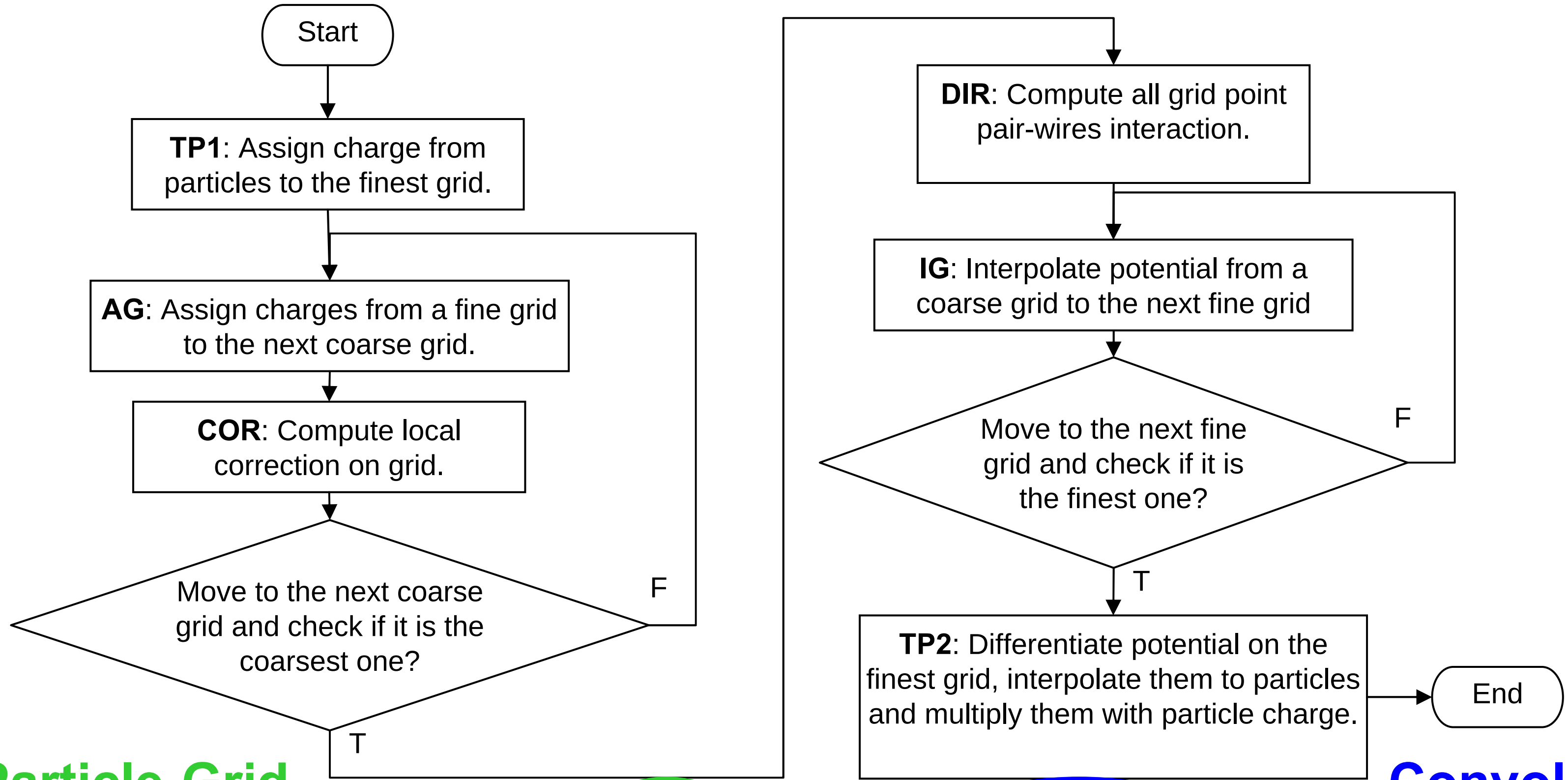
Results – MD Performance



- 77,000 particle model running 1,000 steps
  - Importin Beta bound to the IBB domain of Importin Alpha
    - *The PDB "Molecule of the Month" for January, 2007 !*
  - 93Å x 93Å x 93Å box
- Multigrid speed-up
  - 3.8x over software version of **multigrid** in original ProtoMol
  - 2.9x over software version of **PME** in NAMD
- Total speed-up
  - 7.3x over original ProtoMol
  - 6.5x (5.6x) over NAMD

		Short Range Forces	**Long Range Forces**	Bonded Forces	Motion Integration	Comm. & overhead	Init. & misc.	**TOTAL**
FPGA	FPGA Accelerated **ProtoMol** (2 VP70s) • Multigrid every cycle	533.3	**61.0** (Multigrid)	21.5	20.8	25.6	9.2	**570**
serial	Original **ProtoMol** • Multigrid every cycle	3867.8	**234.1** (Multigrid)	21.6	21.5	0	12.9	**4157**
serial	**NAMD** • SPME every cycle		**177.3** (Multigrid)					**3726**
serial	**NAMD** • SPME every 4th cycle		**---**					**3194**

Results – Multigrid Detail



Particle-Grid

Convolutions

	TP1	TP2	AG	IG	COR	DIR	TOTAL
ProtoMol Multigrid on **PC** 14K model	7.5%	12.3%	5.0%	3.6%	43.9%	27.7%	100%
FPGA Multigrid on **VP70** 14K model	1.1%	3.1%	1.3%	2.0%	60.7%	31.8%	100%

Results – Multigrid Detail

Time in seconds per 1000 iterations

	TP1	TP2	AG Anterpolate	IG Interpolate	COR Correction	DIR Direct Sol.	Overhead	**Total**
Convolution characteristic*			$14^3 \otimes 4^3$	$17^3 \otimes 4^3$	$28^3 \otimes 10^3$	$17^3 \otimes 17^3$		
ProtoMol Multigrid on **PC**	31.6s	45.9s	6.9s	7.5s	62.6s	79.8s	0s	**234s**
FPGA Multigrid on **VP70**	4.1s	12.3s	.52s	.85s	10.7s	13.3s	19.2s	**61s**
FPGA Fraction of peak	---	---	56%	61%	43%	38%	---	**31%**
Speed-up	7.7x	3.7x	13.1x	8.7x	5.9x	6.0x	---	**3.8x**

*only partially describes some of these operations

Convolutions (AG, IG, COR, DIR)

Comments

- Overhead is cost of using this coprocessor
- Matching hardware to problem is critical:
  1. AG is well-matched to HW
     → Kernel held on chip
  2. COR is mismatched:
     → 4^3 hardware for 10^3 kernel
     → Swapping overhead costly
- Start-up/tear-down cost is significant for FPGA
- DIR computes symmetric pairs (2x loss)

Discussion

Why such modest speed-up? *The last time we did 3D convolutions our speed-ups were in the mid-hundreds …*

1. Sensitivity to overhead, problem mismatches, complexity, Amdahl's law
   - 64 Convolution Units gives us a peak throughput of 9.6 GFLOPs
     - **31% is achieved**
   - PC has a peak throughput of 4.8 GFLOPs
     - **16% is achieved**

2. Old FPGAs (VP70)
   - Going to top of line (VP100) helps
   - V4 and V5 are much faster, but don't help with critical component counts … *(they are also cheaper?!)*

3. 35-bit precision is expensive
   (even with semi FP; full FP would be much worse)
   - 4 to 16 bit integer is much easier
   - Hard FP cores would help

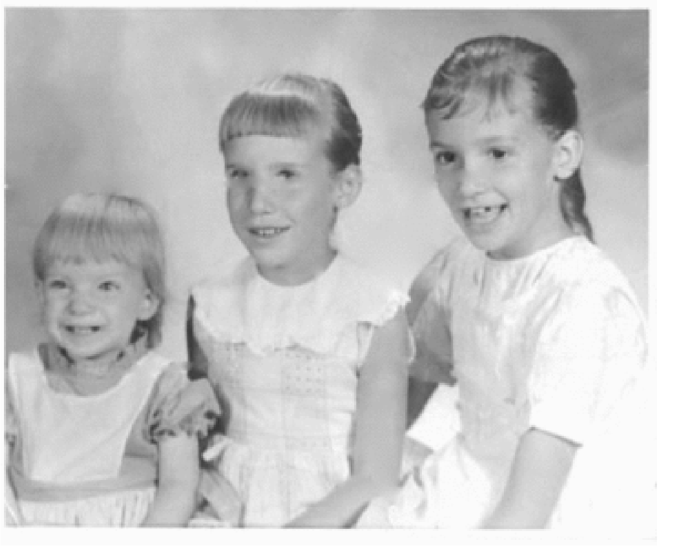
4. Improvable design
   We've done little optimization …

Outline

- Introduction
- Multigrid: Motivation and Overview
- Multigrid for computing the Coulomb force
- **Multigrid for computing linear diffusion**
- Discussion

Image Restoration w/ Linear Diffusion

Linear diffusion is a standard technique in image restoration:*



* M. Bertalmio, et al., SIGGRAPH 2000

Image Restoration w/ Linear Diffusion

- The V-Cycle is constructed by a series of recursive calls on every grid level, from the finest to the coarsest.
- On every level (l), there are 9 steps:
  1. If this is coarsest grid,
     solve $L_l * u_l = q_l$ and return u_l
  2. $u_l = Relax0(u_l, q_l, l)$
  3. $r_l = q_l - L_l * u_l$
  4. $q_l+1 = A^{l+1}_l * r_l$
  5. $u_l+1 = 0$
  6. $u_l+1 = V\text{-}Cycle(u_{l+1}, q_{l+1}, l+1)$
  7. $u_l = u_l + I_l^{l+1} * u_{l+1}$
  8. $u_l = Relax1(u_l, q_l, l)$
  9. Return u_l

Initial Guess

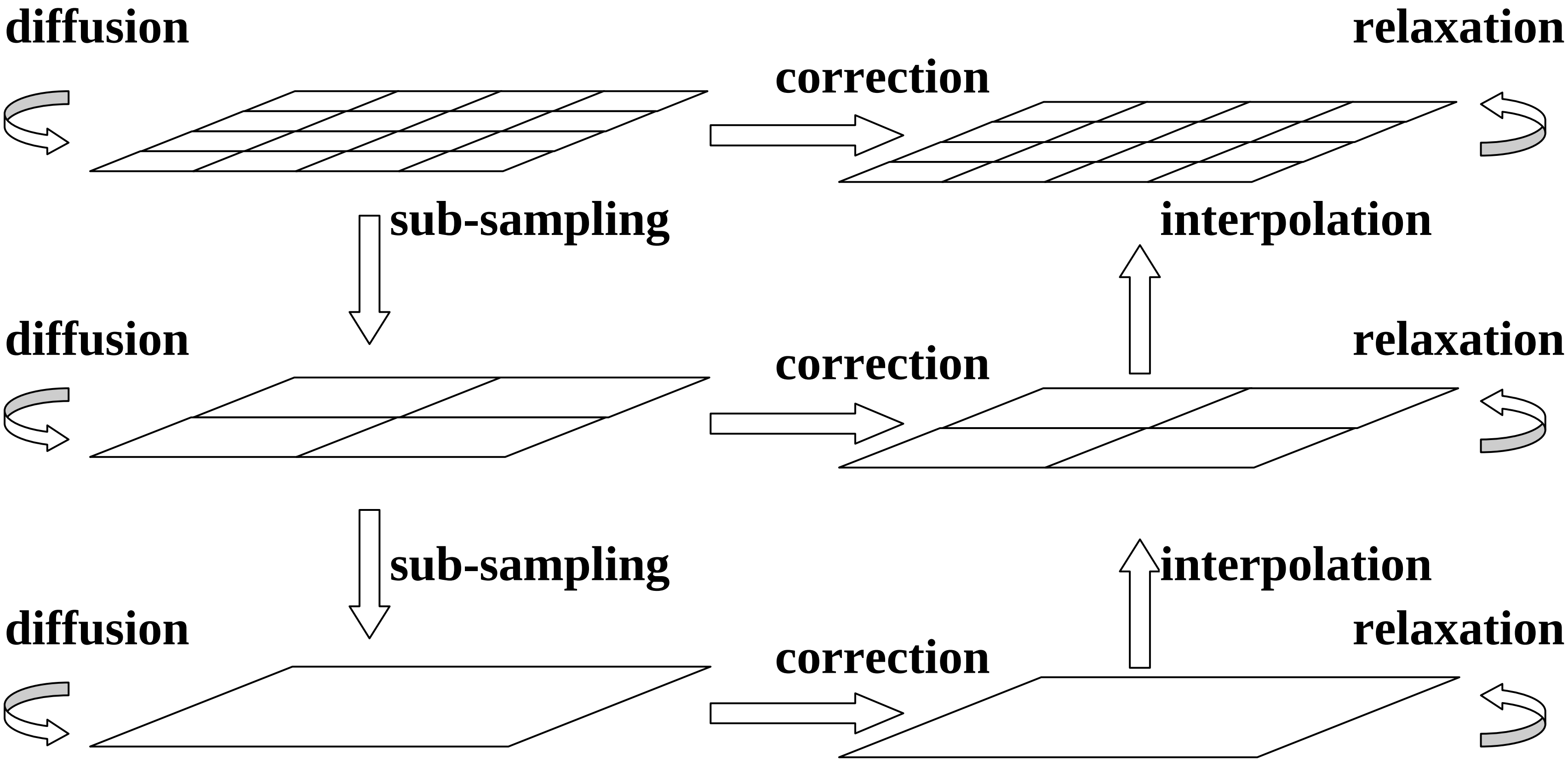
After Relaxation

After Correction

Linear Image Diffusion

- General Form: $A \cdot X = B$
- V-cycle:
  - Initial guess $\hat{X}_l$
  - Relaxation $\hat{X}_l = S(\hat{X}_l)$
  - Residue $R_l = B_l - \hat{X}_l$
  - Subsampling $A \cdot E_{l+1} = R_{l\downarrow} = (B_l - \hat{X}_l)_\downarrow$
  - ……
  - Direct Solution $E_{coarsest} = 0$
  - ……
  - Correction and Interpolation $\hat{X}_l = \hat{X}_l + E_{l+1\uparrow}$
  - Relaxation $\hat{X}_l = S(\hat{X}_l)$

Linear Image Diffusion



BOSTON UNIVERSITY

FPGA/Multigrid Problem Space

Design space dimensions:

- System scale:
  - Size of the finest grid & # of dimensions
- Computation kernel scale:
  - Size of relaxation and diffusion matrices
- Kernel operations:
  - MAC? A complex function?
- Data type:
  - Integer? Floating-point? Complex structure? Other?
- Overhead:
  - Discretizations? Other?

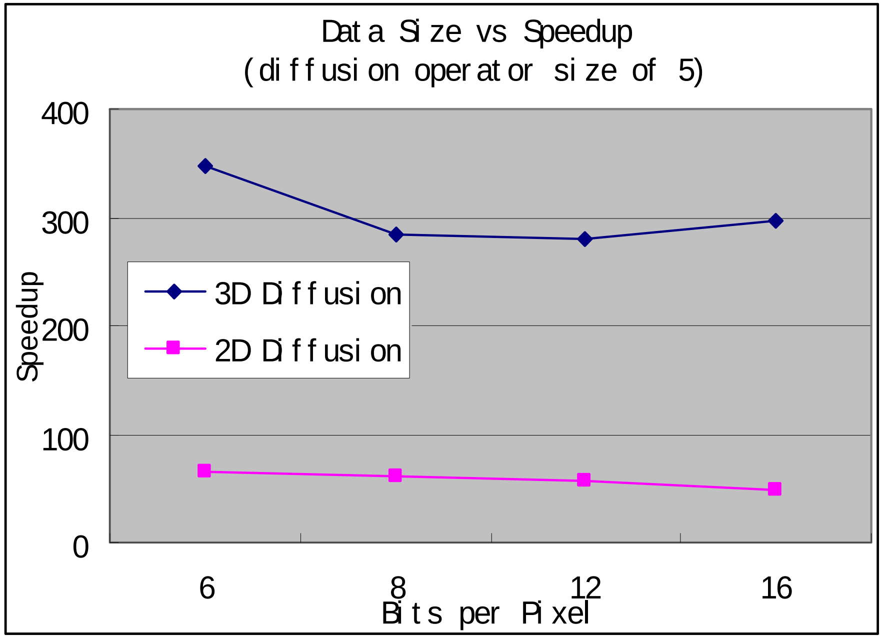
Linear Image Diffusion

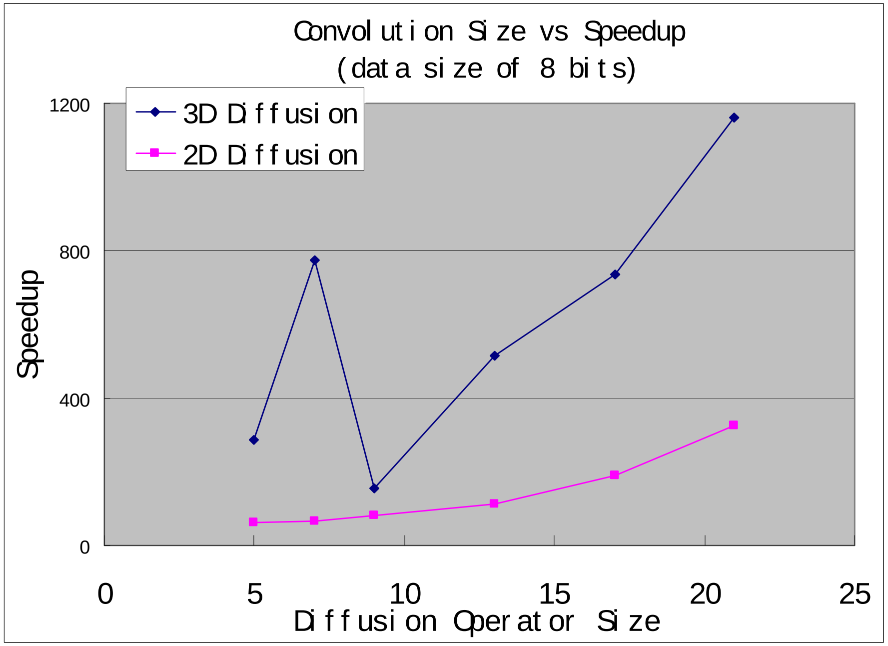
<u>Fixed settings</u>

- Image size: 256 pixels per dimension
- Gauss-Seidel relaxation: 3 pixels per dimension
- Relaxation: two iterations per level
- Grid levels: 3
- Reference codes: single core 2.4GHz PC

<u>Variable settings</u>

- Dimensions: 2D or 3D
- Bits per pixels: 6 to 16
- Diffusion operator size: 5 to 21





Diffusion Discussion

- **Speedup vs. Data size**
  - As long as the pixel width is less than the HW multiplier capability, speedup remains almost constant for 2D.
  - In 3D, larger pixels cause high chip utilization which results in reduced operating frequency.

- **Speedup vs. Operator size**
  - As long as the convolution kernel fits on chip, the larger the kernel, the more parallelism and speedup achieved.
  - When the kernel does not fit (in some 3D cases), the 3D convolution is split into multiple 2D ones. Speedup drops at first, but then increases with the operator size and parallelism.

Outline

- Introduction
- Multigrid: Motivation and Overview
- Multigrid for computing the Coulomb force
- Multigrid for computing linear diffusion
- **Discussion**

Conclusions

- Solving Poisson's equation …

  Effectiveness = *borderline*

- Simple image restoration …

  Effectiveness = *high*

- Generalization …
  - Multiplier size, # of multipliers, critical for determining integer data type
  - Real floating point requires more support

Questions?