High Performance Molecular Dynamics Simulations with FPGA Coprocessors*

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

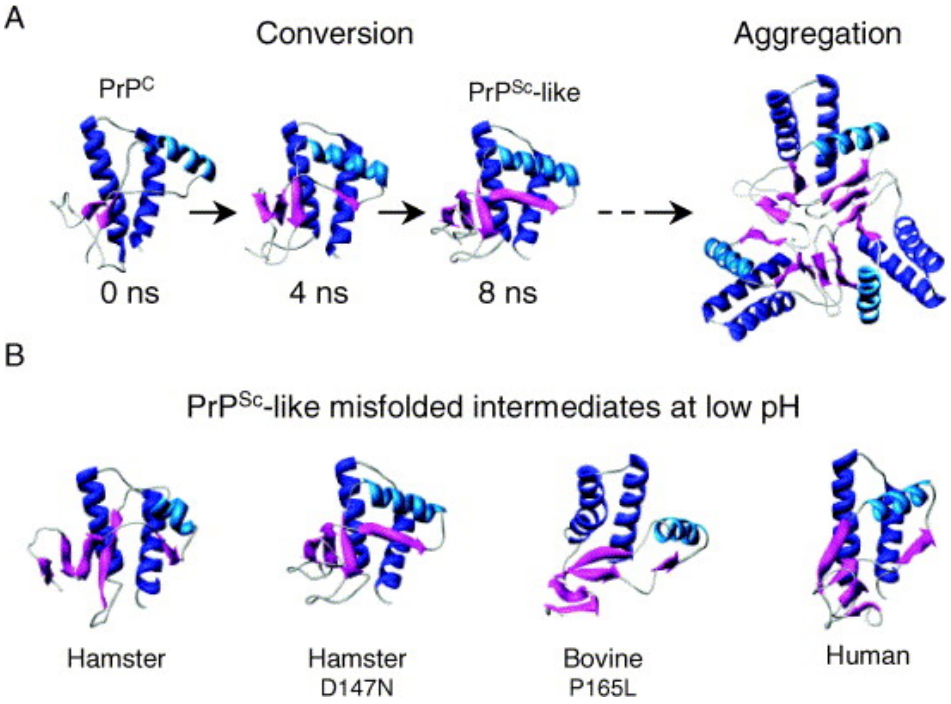
BOSTON UNIVERSITY

Agenda

- **Motivation**
- Background
- Algorithm-level designs
- Implementation and results
- Future directions

Why Molecular Dynamics Simulation is so important …

- Core of Computational Chemistry
- Central to Computational Biology, with applications to
  - → Drug design
  - → Understanding disease processes …



From DeMarco & Dagett: PNAS 2/24/04

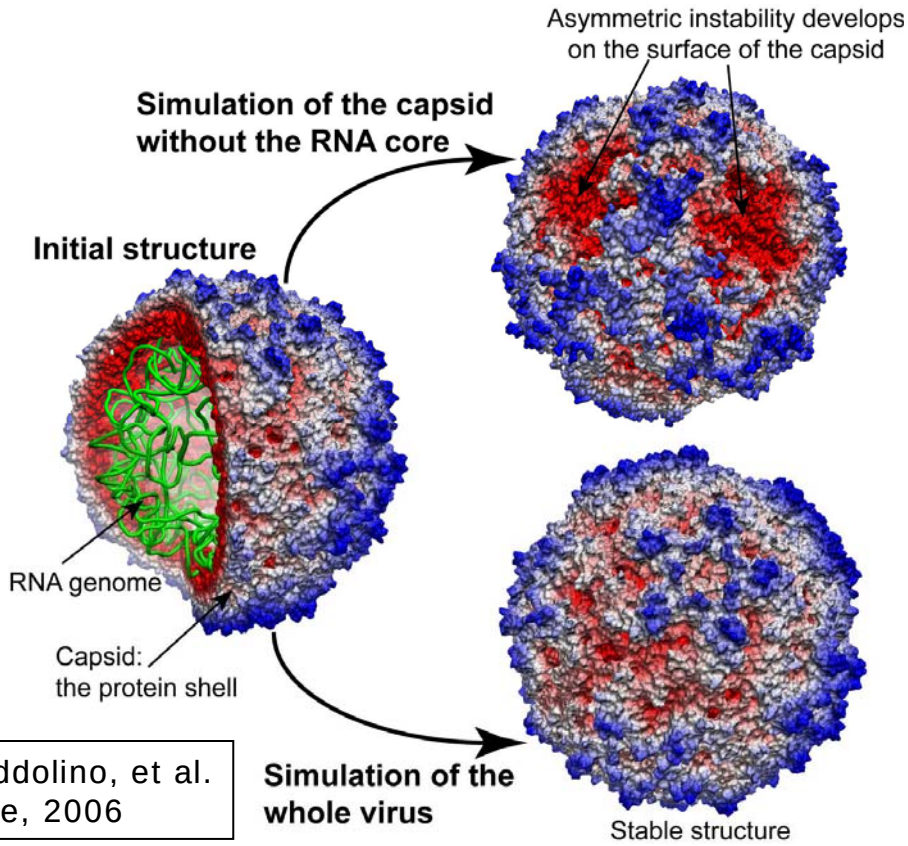
Shows conversion of PrP protein from healthy to harmful isoform. Aggregation of misfolded *intermediates* appears to be the pathogenic species in amyloid (e.g. "mad cow" & Alzheimer's) diseases.

Note: this could *only* have been discovered with simulation!

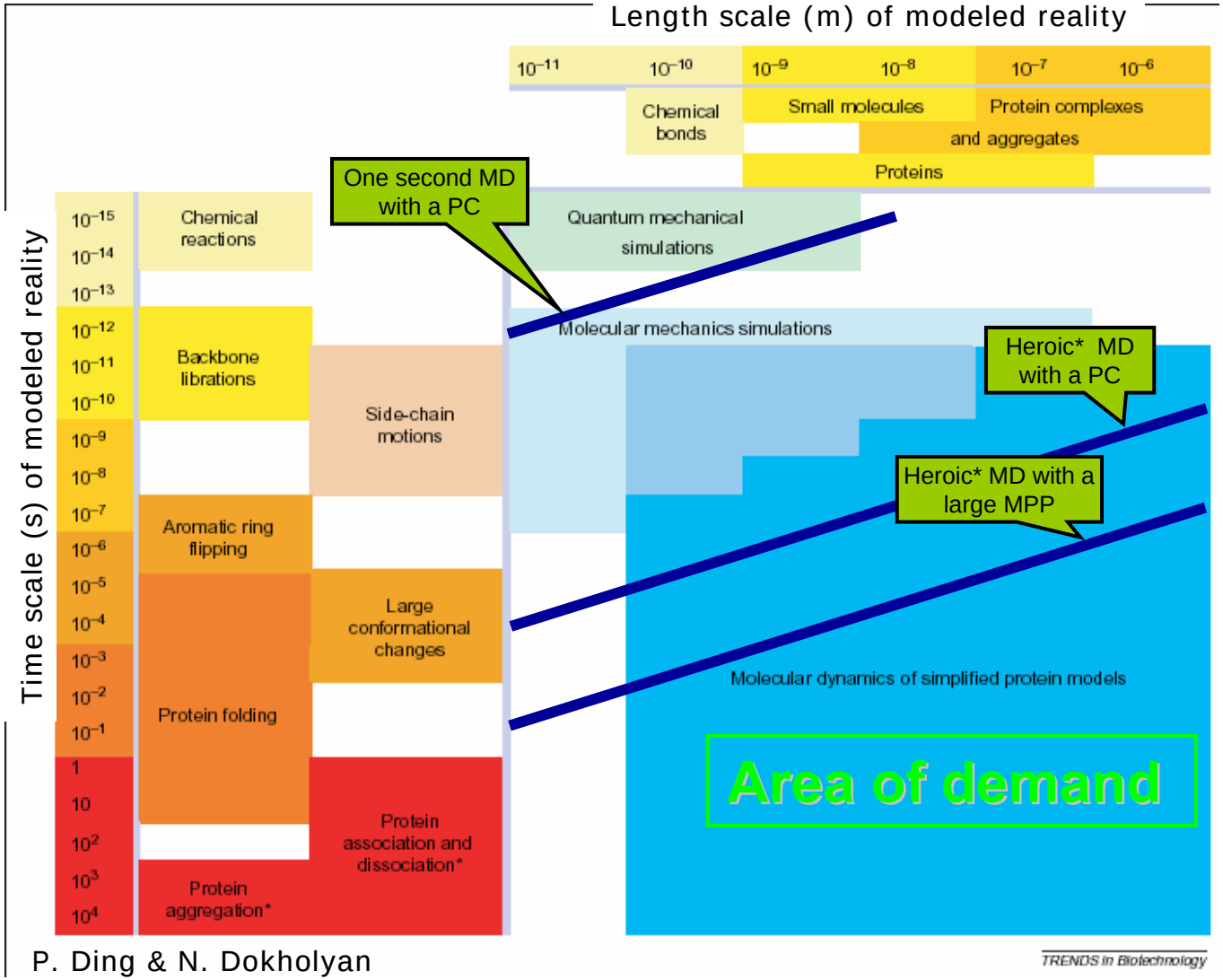
Why *Acceleration* of MD Simulations is so important …

MD simulations are often "heroic" →

100 days on 500 nodes …



P.L. Freddolino, et al. Structure, 2006



P. Ding & N. Dokholyan
Trends in Biotechnology,2005

*__Heroic__ ≡ > one month elapsed time

The State of High Performance Computing Node Architecture

- **High performance computing (HPC) requires ever more computing power**

  But, while
  - Transistor density continues to increase with Moore's Law,
  - Microprocessor operating frequency hasn't increased since 2003
    - *due to the problems with power density*
  - New microprocessors have multiple CPU cores
    - *Much harder to use efficiently than traditional microprocessors*
    - *Questions whether scalable performance will ever be achieved*

- **Alternative architectures are being explored:**
  - Specialized multicore (Cell), GPUs, SIMD (Clearspeed), **FPGAs**
  - Idea: Match appropriate accelerator to appropriate domain

FPGAs Are Promising Because …

Large amount of logic (sea of configurable gates)

Large number of computer parts (sea of microcores)

- Hundreds of DSP computation units (MACs)
- Hundreds of independently accessible memories (Block RAMs, BRAMs)

Configurable into various architectural styles

- Very deep pipelines (hundreds of stages)
- Pipeline replication (hundreds of small pipelines)
- Complex data reference patterns (flexible supercomputing-style interleaving)
- Associative computing (broadcast, matching, query processing, reduction)

Tremendous computing capability

- Massive parallelism (100 ~ 1000+ PEs)
- Flexible memory interface and enormous bandwidth (> 1 TB/sec)
- Payload on every cycle (control in hardware, not software)

Commodity parts

- Leverage markets for communication switches and DSP
- High-end always uses latest VLSI processes (65nm)
- Well balanced among performance, generalization, and development effort.

Many vendors, support from dominant companies

- Offerings from SGI, Cray, SRC, and many smaller companies
- Intel supporting accelerators including FPGAs at all levels including front side

FPGAs Are Not Completely General Purpose

- **Low operating frequency (~ 1/10x)**
  - Freedom is not free.
- **Sharp learning curve**
  - Few application experts are good at FPGA design.

Therefore – ***Performance of HPC using FPGAs is unusually sensitive to the quality of the implementation.***

Overhead must be scrupulously avoided in implementation, both in tools and in architectures.

The Problem We Address is …

… how to bring to researchers in computational science substantially more cost-effective MD capability.

We address this problem by enabling the use for MD of *a new generation of computers based on reconfigurable logic devices: Field Programmable Gate Arrays or FPGAs*.

Haven't we seen enough MD talks?

Groups working on accelerating MD with FPGAs: sample recent publications

Alam	Computer 2007
Azizi	FCCM 2004
Gu	FPL 2006
Kindratenko	FCCM 2006
Komeiji	J. Comp. Chem 1997
Scrofano	FCCM 2006

Axes in Design Space

- Precision
- Arithmetic mode
- Base MD code
- Target hardware
- Design flow
- Scope of MD acceleration

The Challenge

Although FPGAs have achieved 100x speed-ups, these applications have often had the following characteristics:

- Low precision
- Integer arithmetic
- Small, regular computational kernels
- Simple data access patterns

MD codes have:

- Floating point (some integer in some codes)
- Double precision (some single precision in some codes)
- Several small-medium kernels, and much other code
- Moderately complex data access patterns

Our Approach

FPGA-aware redesign of almost every aspect of Molecular Dynamics (MD):

- Overall design
- Algorithm selection/restructuring
- New arithmetic mode
- Experimentally determined interpolation
- Experimentally determined precision
- Partitioning, integration into MD production code
- Micro-architectures for several major components

Summary of Contributions

- **Substantial speedup over production MD codes, e.g. NAMD**
  - By far the best reported performance of FPGA acceleration.
- **FPGA-aware mappings of MD algorithms**
  - Short-range forces
  - Long-range force using multigrid
    - so far the only published FPGA solution for long-range force computation
  - Cell-list
  - Non-bonded force exclusion
- **Complete MD system**
  - Coprocessor micro-architectures
  - Production code integration
- **Novel numerical computation approach**
  - Semi floating point arithmetic mode

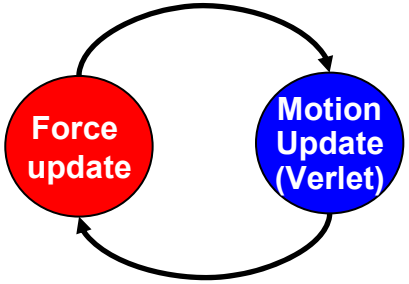
Agenda

- Motivation
- **Background**
- Algorithm-level designs
- Implementation and results
- Future directions

What is Molecular Dynamics?

MD – An iterative application of Newtonian mechanics to ensembles of atoms and molecules

Runs in phases:



Generally O(n), done on host

Initially O(n²), done on coprocessor

Many forces typically computed, $F^{total} = F^{bond} + F^{angle} + F^{torsion} + F^{H} + F^{non-bonded}$

but complexity lies in the non-bonded, spatially extended forces:
van der Waals (LJ) and Coulombic (C)

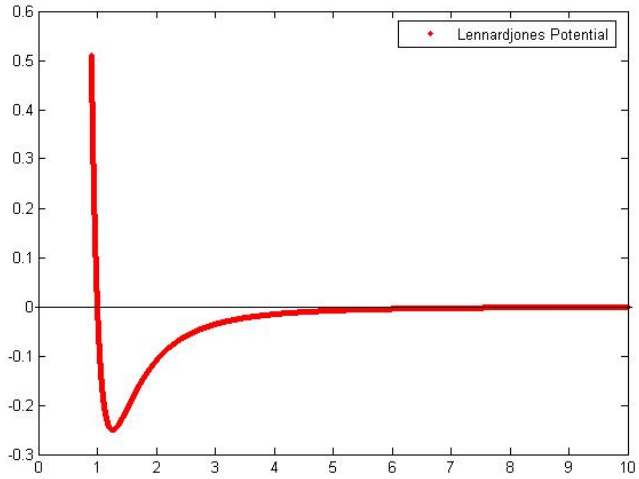
$$F_i^{LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \vec{r}_{ji}$$

$$F_i^{C} = q_i \sum_{j \neq i} \left(\frac{q_i}{|r_{ji}|^3} \right) \vec{r}_{ji}$$

Reducing the $O(N^2)$ Complexity …

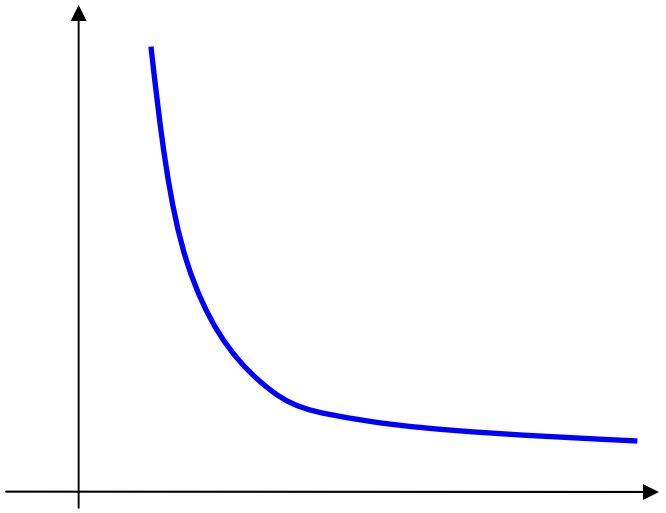
LJ force gets small quickly …

$$F_i^{LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \vec{r}_{ji}$$



… while the Coulombic force does not ...

$$F_i^{C} = q_i \sum_{j \neq i} \left(\frac{q_i}{|r_{ji}|^3} \right) \vec{r}_{ji}$$



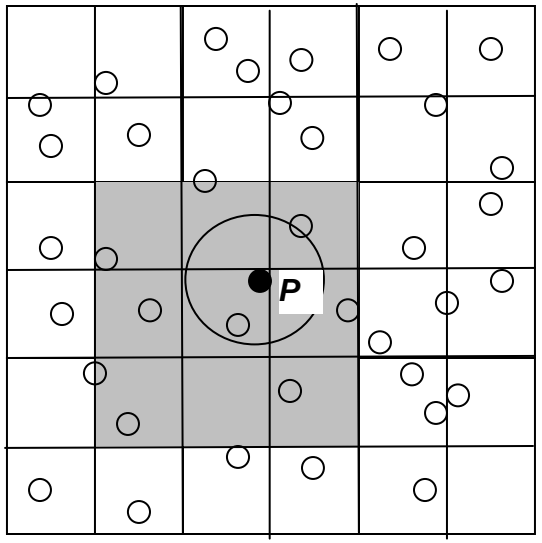
Make LJ *O(N)* with Cell Lists

$$F_i^{LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \vec{r}_{ji}$$

Observation:

- Typical volume to be simulated = 100Å^3
- Typical LJ cut-off radius = 10Å

Therefore, for all-to-all O(N^2) computation, *most work is wasted*



Solution:

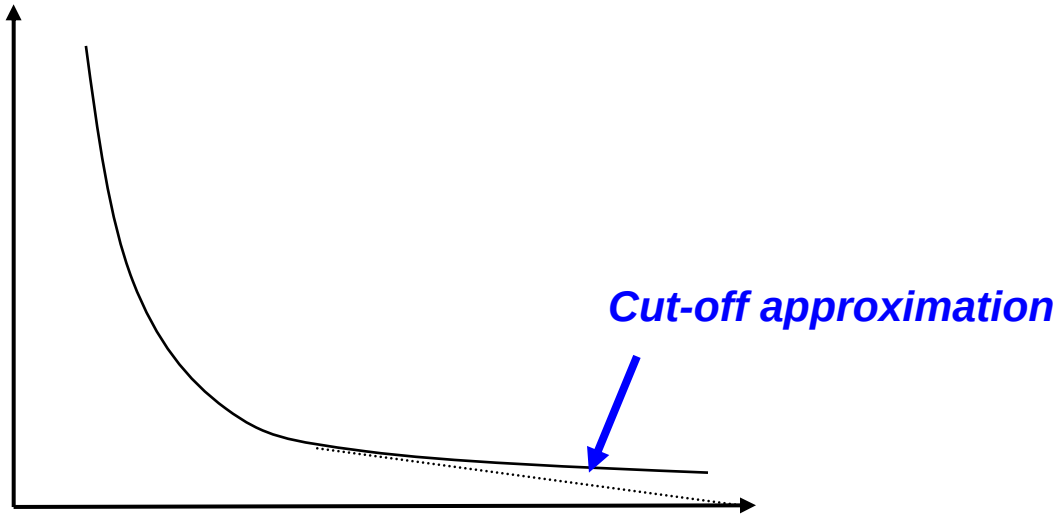
Partition space into "cells," each roughly the size of the cut-off

Compute forces on ***P*** *only w.r.t. particles in adjacent cells.*

- Issue → shape of cell – spherical would be more efficient, but cubic is easier to control
- Issue → size of cell – smaller cells mean less useless force computations, but more difficult control. Limit is where the cell is the atom itself.

Make Coulombic *O(N)* with Cut-Off

The Coulombic force can be approximated with a “cut-off” function, and computed with cell lists along with the LJ …,



… but this is often insufficiently accurate

Standard HPC Implementation, cont.

Make long-range force *o(N^2)* with transforms

Various clever methods to reduce complexity:

- Ewald Sums
  - *O($N^{3/2}$)*
- (Smooth) Particle Mesh Ewald
  - *O(N log N)*
- Fast Fourier Poisson
  - *O(N log N)*

Standard HPC Implementation, cont.

Many software packages:

NAMD, AMBER, CHARMM, GROMACS, LAMMPS, ProtoMol, …, and more coming.

Agenda

- Motivation
- Background
- **Algorithm-level designs**
  - **Short-range force**
  - Long-range force
- Implementation and results
- Future directions

Short-Range Force Computation

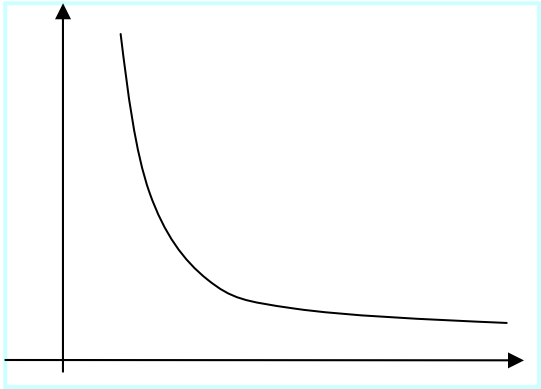
Problem:

– Compute force equations such as $$F_i^{LJ} = \sum_{j \neq i} \frac{\varepsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \vec{r}_{ji}$$

Difficulty:

– It requires expensive division operations for r^{-x}.

Method: Use table look-up with interpolation, but on individual terms (r^{-4}, r^{-7})



Also used for short-range component of Coulombic

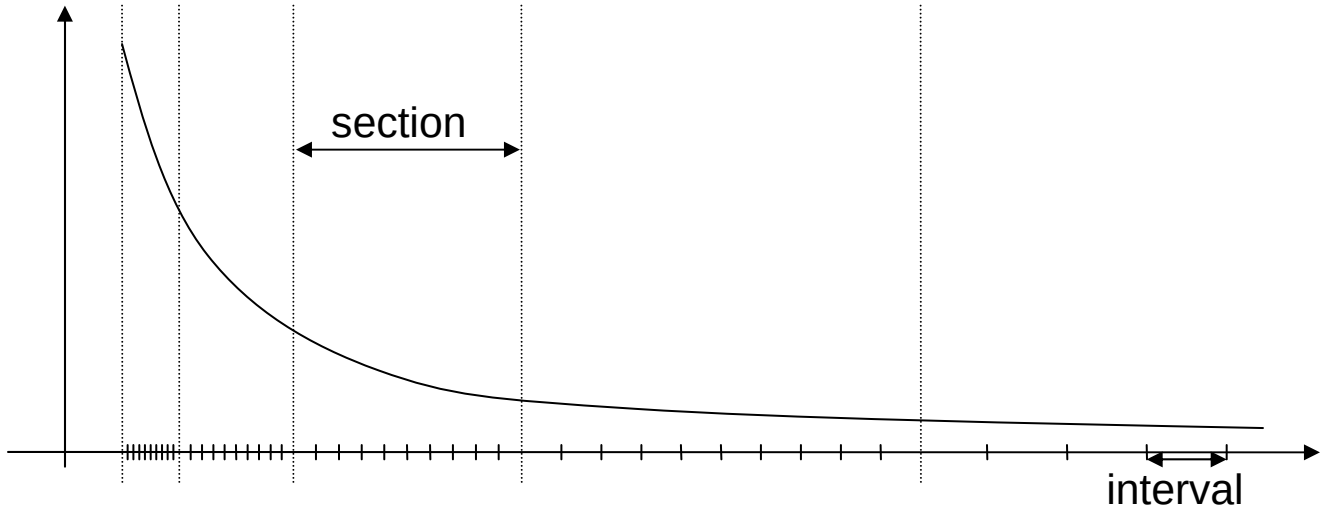
→ three tables are needed, plus further computation

An Optimization

Vary section size → Length of each is double that of previous

Advantages:

- Concentrates intervals in region of highest curvature
- Allows simple extension to large cut-off radii



Polynomial Interpolation

- **Equation:**

$$f(x) = C_0 + C_1(x-a) + C_2(x-a)^2 + C_3(x-a)^3 + \ldots + C_M(x-a)^M + o(x^M)$$

- **Two Issues:**
  - Order (M) and interval resolution (N)
    - Depending on the accuracy/performance trade-off
  - Coefficients $\{C_i\}$ – what *type* of interpolation?
    - Depending on the required properties of the approximating curves

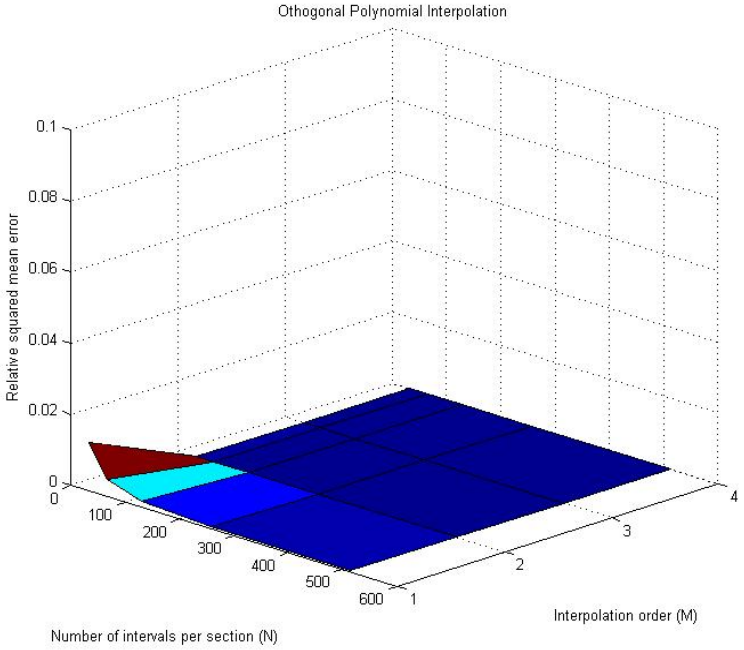
Interpolation Order & Interval Resolution

- **Problem:**
  - Minimize approximation error with given HW resources.
- **Solution:**
  - Trade-off between interpolation order (M) and interval size (N) for r^{-x} term.
  - Note: M and N affect different resource types (multipliers versus BRAMs).

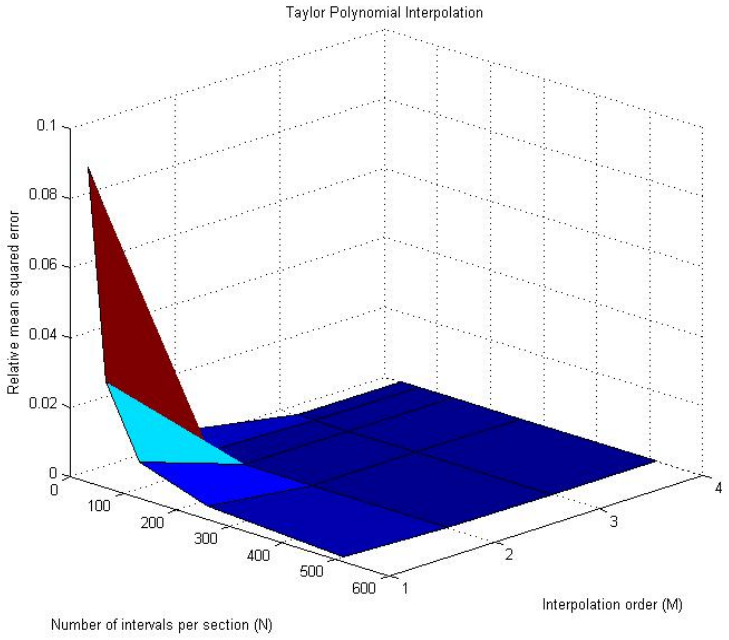
N	M	Average Relative Error	Maximum Relative Error
32	4	2.55e-7	3.67e-6
64	4	7.35e-9	1.08e-7
64	3	3.74e-7	4.19e-6
128	3	2.26e-8	2.55e-7
128	2	2.17e-6	1.73e-5
512	2	3.32e-8	2.66e-7
512	1	1.17e-5	6.04e-5
2048	1	7.31e-7	3.76e-6

Coefficients: Polynomial Interpolation Comparison

- Some candidates: **Taylor, Hermite, Orthogonal**
- Relative root mean squared error with logarithmic size intervals
- Target function: $f(r)=r^{-7}$ on $(2^{-4},2^{7})$.
- N: number of intervals per section; M: interpolation order



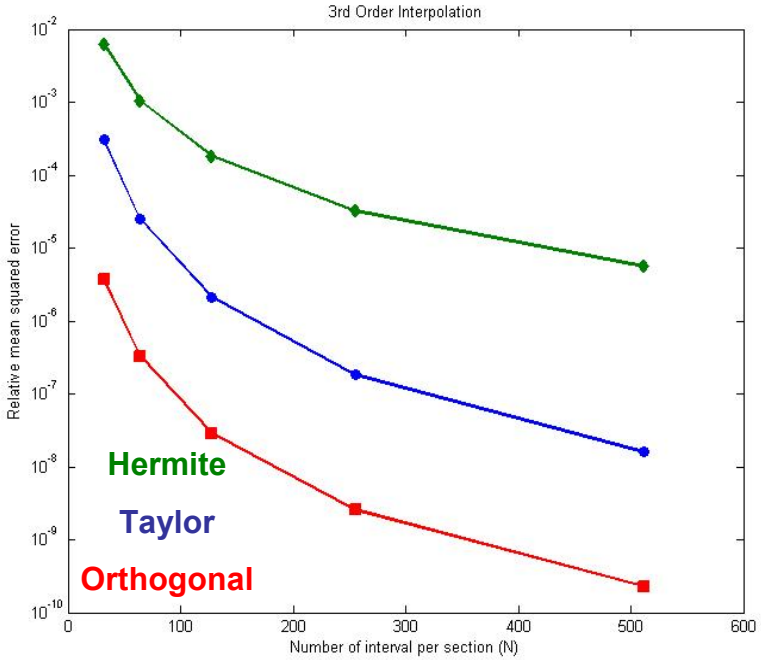
Orthogonal Polynomial Interpolation



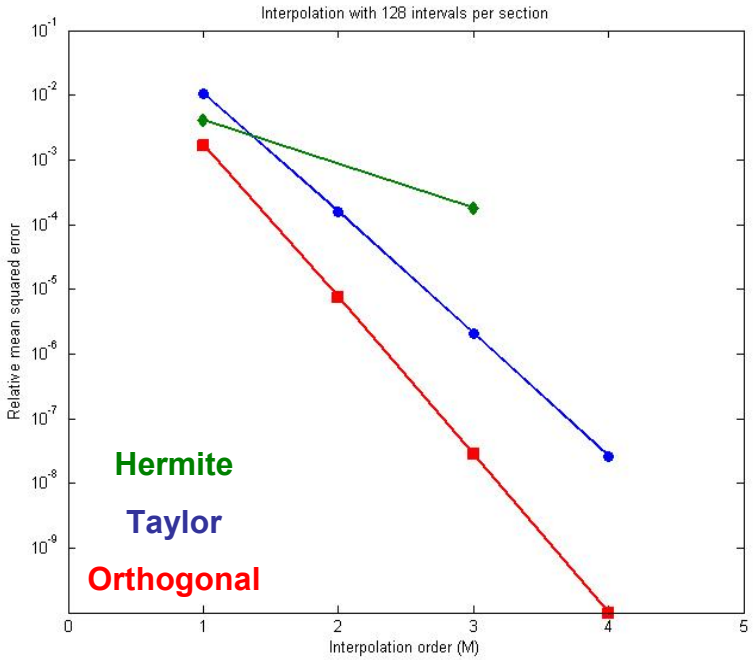
Taylor Polynomial Interpolation

Polynomial Interpolation Comparison, cont.

- 3 order interpolation with variant interval numbers



- 128 intervals per section with variant interpolation order



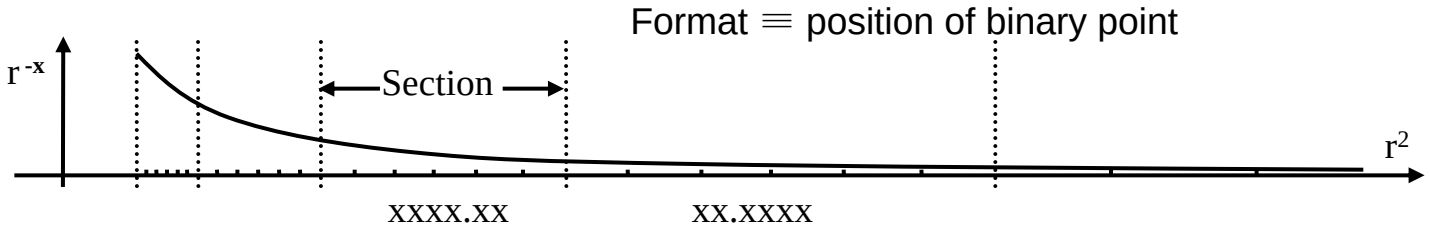
- **Orthogonal polynomial interpolation has the least squared error under same combination of interpolation order and interval numbers**
- **Zero approximation bias, i.e.** $\int_a^b (F(x) - P_{ab}(x))dx = 0$

Arithmetic Mode – Semi Floating Point

Interpolation needs accurate arithmetic mode.

- **Floating point number**
  - Problem: too expensive on FPGAs
- **Fixed point number**
  - Problem: the magnitude of interpolation coefficients varies tremendously
- **Our solution: Semi-FP**
  - Fixed point numbers of different, but limited formats among sections; same format within one section; same data width for all sections.

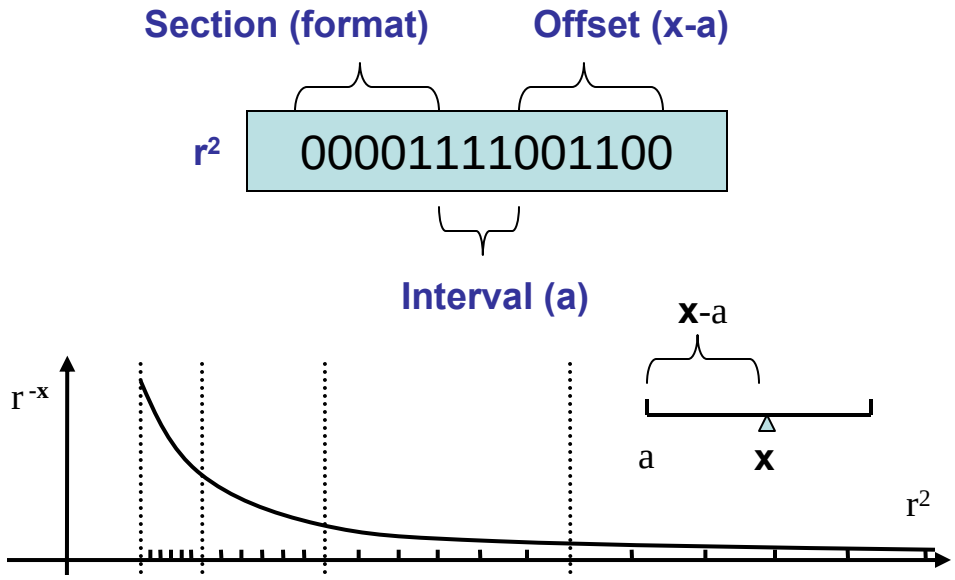
# of slices / Delay	Multiplier	Adder
DP	540 / 8	692 / 12
Semi-FP	316 / 7	70 / 1

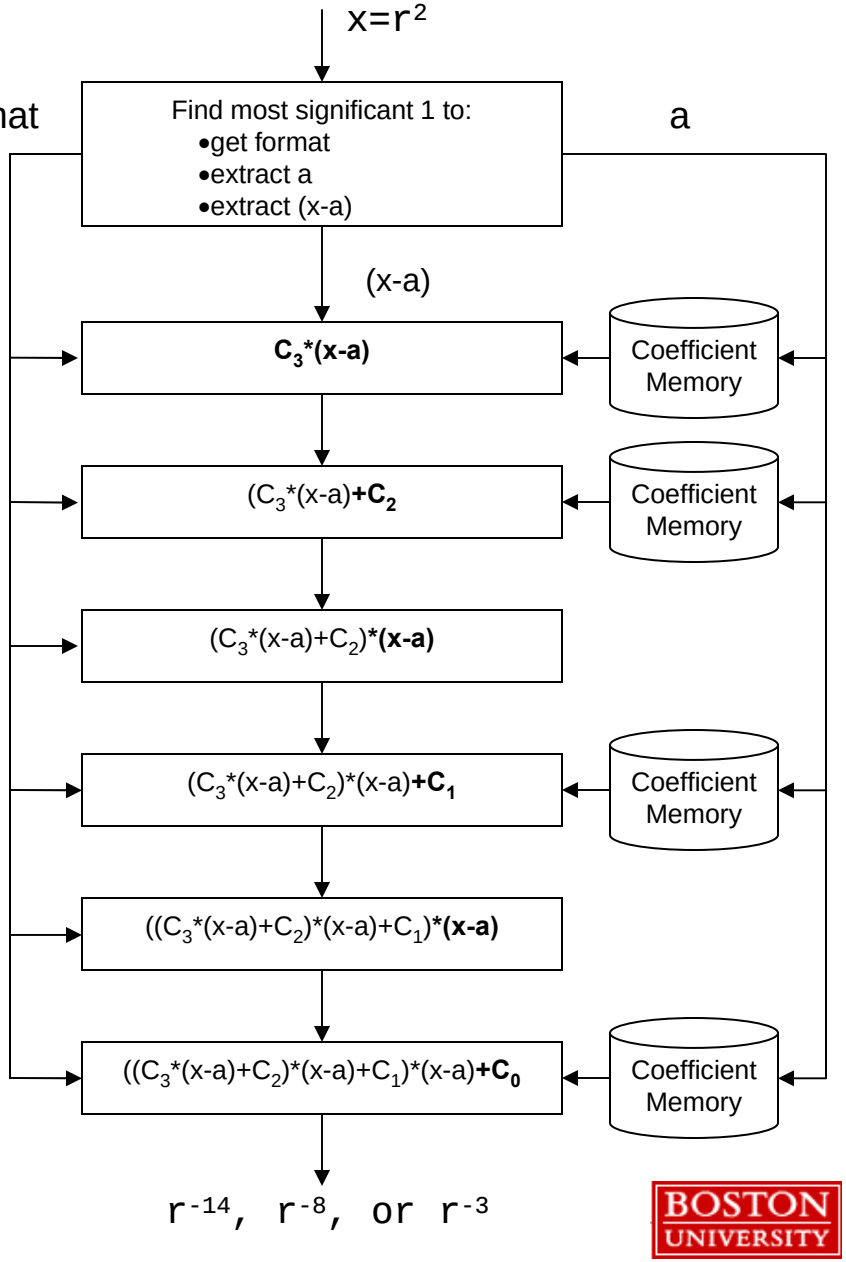


Interpolation Pipeline with Semi-FP

- r^{-x} interpolation Pipeline
  - 'a' is the starting point of an interval

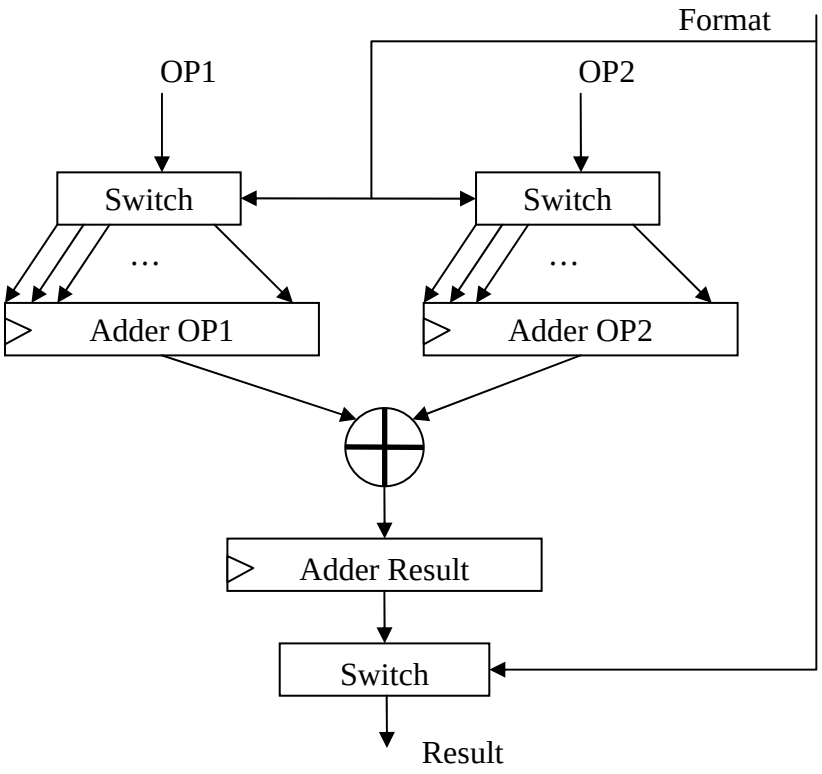
$$f_{section}(x) = \sum_{i=0}^{M} C_i (x-a)^i$$

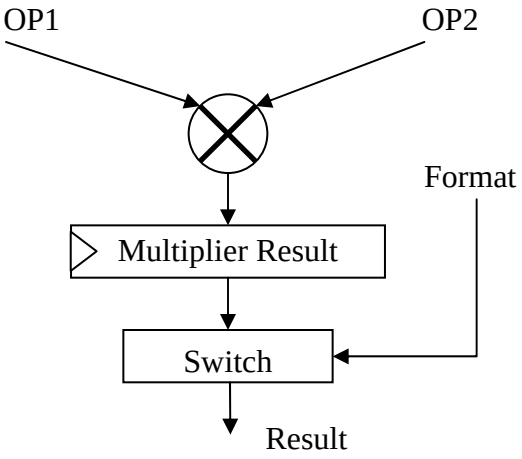




Hardware for Semi-FP Operations

- Semi-FP Adder and Multiplier





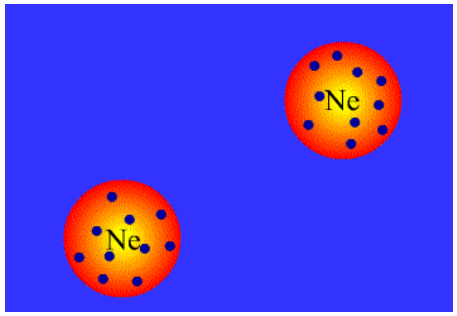
Benefits of Semi-FP

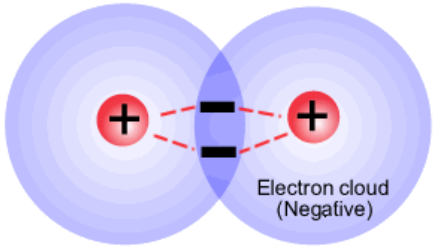
- Slices per force pipeline

Format	Adder	Multiplier	Complete Force Pipeline
LogicCore double precision FP – 53 bits	692	540	19566
LogicCore single precision FP – 24 bits	329	139	6998
Semi-FP – 35 bits	70	316	-
Integer – 35 bits	18	307	-
Combined Semi-FP, integer – 35 bits	-	-	4622

Non-bonded Force Exclusion

- **Problem**:
  - **Non-bonded forces only exists between particles without covalent bonds.**





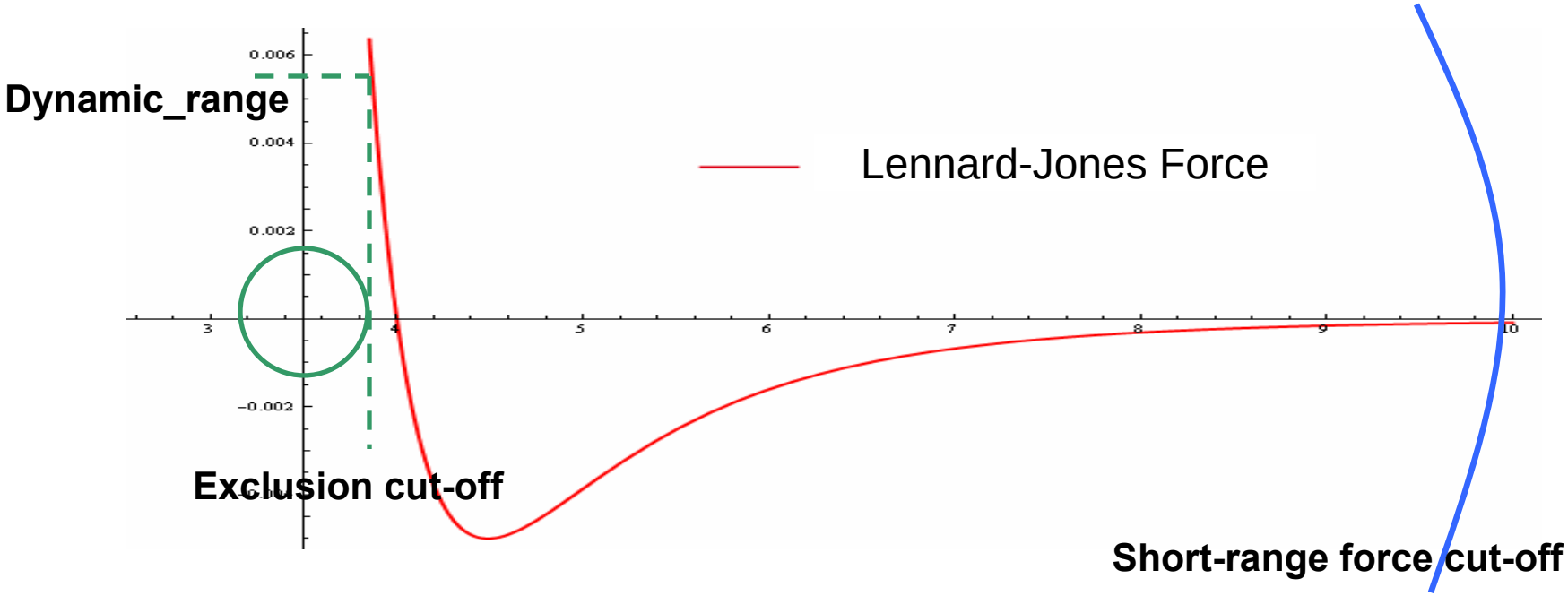
- **Solutions:**
  - **Check bonds on-the-fly** during computing non-bonded forces
    - Expensive to check bond information on-the-fly
  - **Build a pair-list** for non-bonded particle pairs
    - Random access for particles needed
  - **Compute force blindly** first, and **subtract the complementary force** later
    - Loss of precision caused by fake forces (underflow) or pure fixed-point arithmetic required (overflow)

Picture sources: http://ithacasciencezone.com/chemzone/lessons/03bonding/mleebonding/van_der_waals_forces.htm

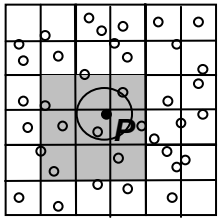
http://ibchem.com/IB/ibnotes/full/bon_htm/4.2.htm

Non-bonded Force Exclusion

- **Optimization:**
  - **Alternative extend from the last solution**
    - Close-range cut-off prevents huge fake forces.
    - Calculate the cut-off with $\vec{F}^{short}(r^2) < dynamic\_range$.
    - Multiple cut-offs according to particle types are required.

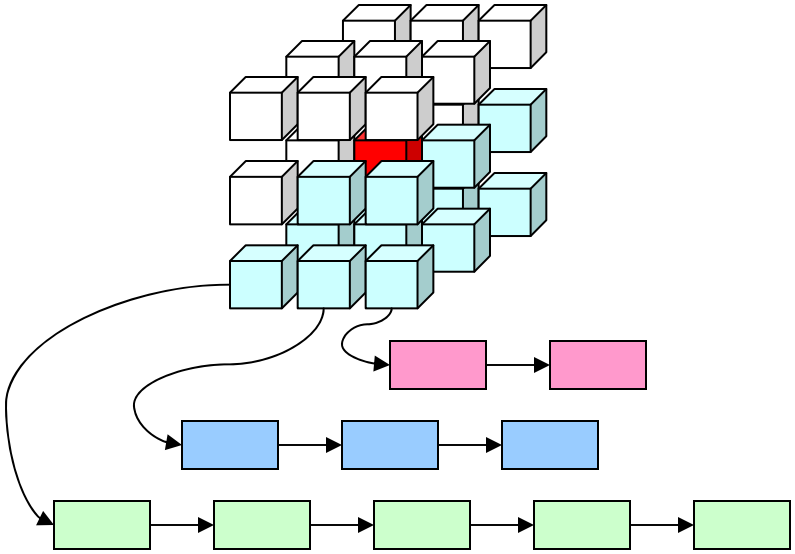


Cell-list, No Longer a List



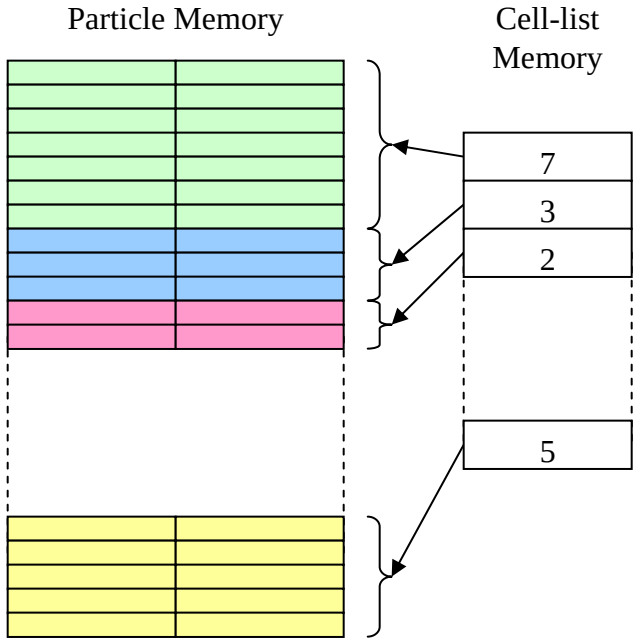
Software:

- **Link lists of indices**
  No particle moving in the particle memory during cell-list construction.



FPGA:

- **Index table and segmentation**
  Particles grouped by cells in the particle memory.



Handling Large Simulations

- **Problem:**
  - On-chip BRAMs are limited for large simulations, so use off-chip memories.
- **Difficulties:**
  - Off-chip interface is different among platforms.
  - Off-chip access may reduce on-chip processing efficiency because of bandwidth and latency.
- **Solution, part 1:**
  - Abstract memory interface is defined to minimize platform dependency.

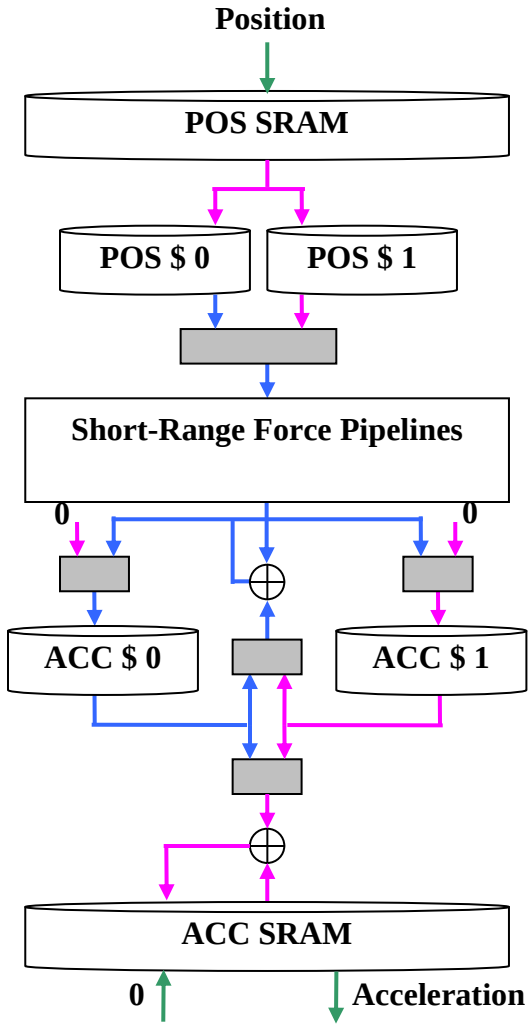
Particle Data Access Characters

- **Observations:**
  - Spatial Locality:

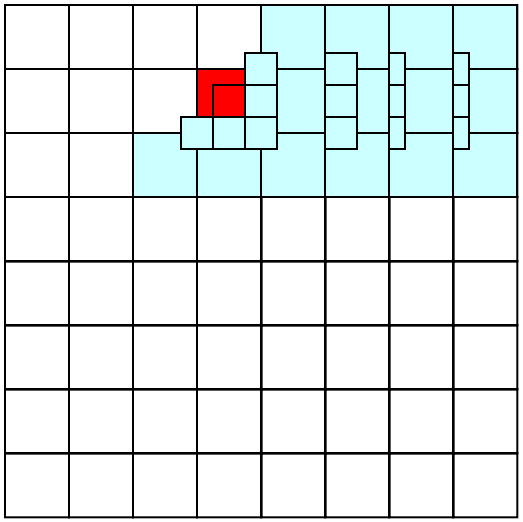
    Anytime, only a small piece of simulation model is processed.
  - Temporal Locality:

    Once a piece of simulation model (N particles) is swapped on-chip, $O(N^2)$ computation are to be conducted.
  - Cache Line Conflict:

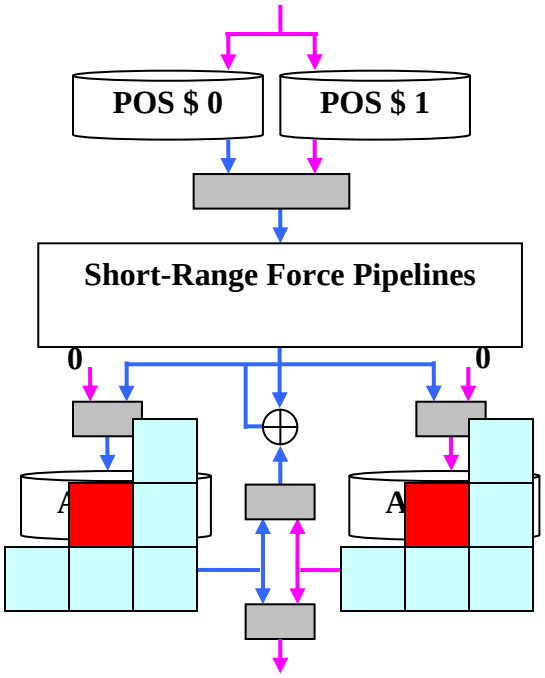
    Particle memory access pattern is deterministic, no miss or congestion.



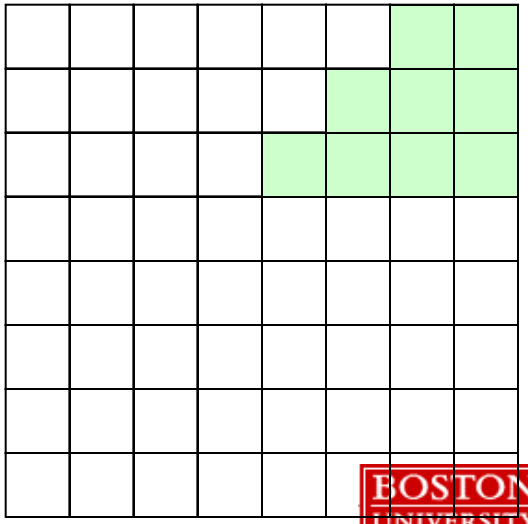
Explicitly Managed Cache



POS SRAM



ACC SRAM



Solution, Part 2:

–Explicitly Managed Cache

Large Simulation with Off-chip Memory

- **Benefits:**
  - Support more particles.
  - Save BRAMs for more pipelines.

Chip: VP70	# of particles	# of pipelines
w/o off-chip mem	8K	2
w/ off-chip mem	256K	4

  - Abstract memory interface is defined with relaxing bandwidth requirement
    - **Dual-port SRAM interface**
    - **~100 bits per cycle**

Summary – Short-Range Force Computation

- Force pipeline
  - Orthogonal polynomial interpolation for r^{-x}
  - Semi-FP arithmetic mode
  - Exclusion with short distance cut-off checking
- Cell-list algorithm
- Explicitly managed cache and off-chip memory interface

Agenda

- Motivation
- Background
- **Algorithm-level designs**
  - Short-range force
  - **Long-range force**
- Implementation and results
- Future directions

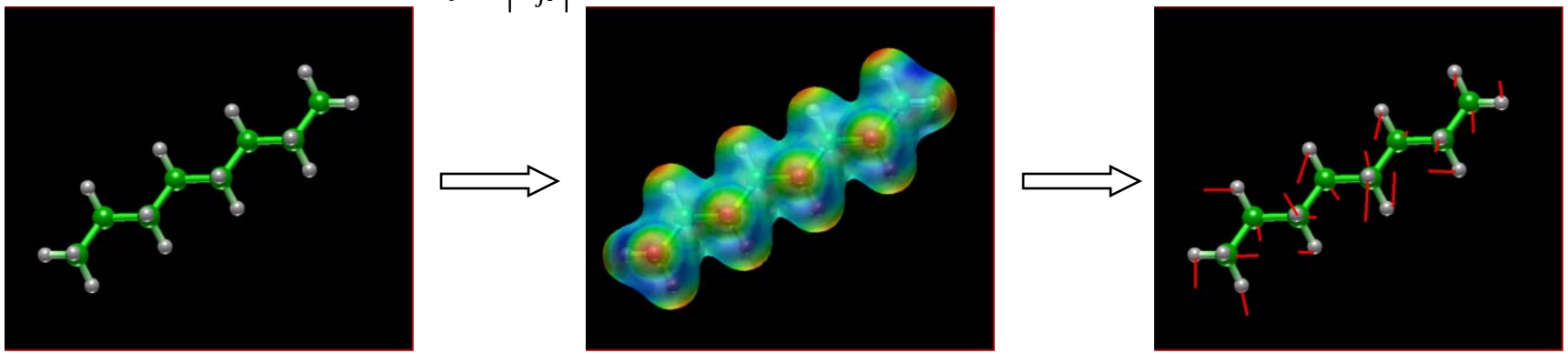
Long-Range Force Computation

- **Problem:**

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

- **Difficulties:**
  - Summing the charges is again an all-to-all operation.

Goal: Make Coulomb Force $o(N^2)$
Common HPC Solution – Use a Transform

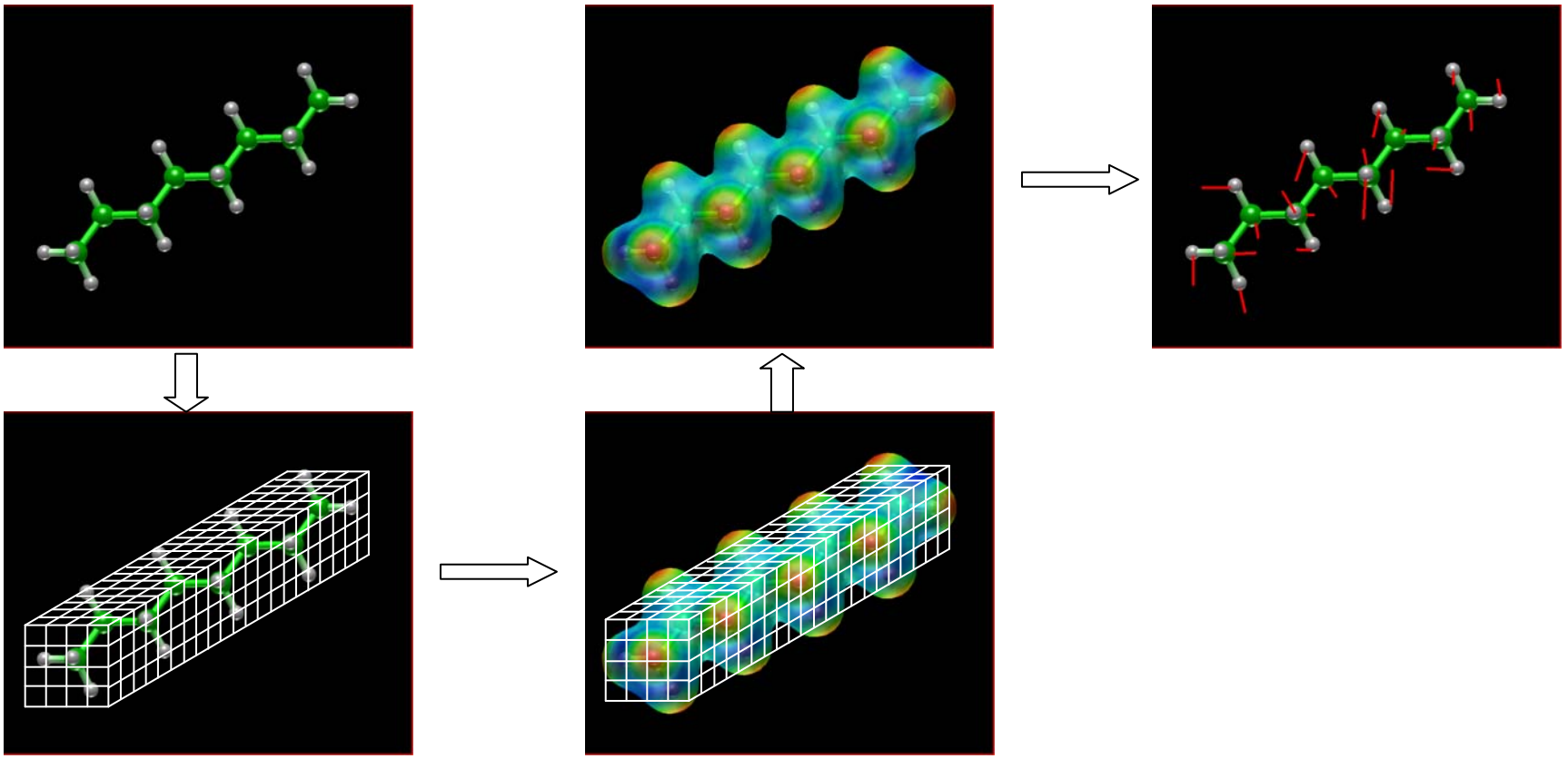
- **Various clever methods to reduce complexity:**
  - Ewald Sum → $O(N^{3/2})$
  - (Smooth) Particle Mesh Ewald → $O(N \log N)$
  - Fast Fourier Poisson → $O(N \log N)$

- **Previous work w/ FPGAs**
  - S. Lee (MS Thesis 2005)
  - Others either ignore or do in SW

- **Difficulty** → ***requires 3D FFTs***

Let's try something that FPGAs are really good at ...

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, …

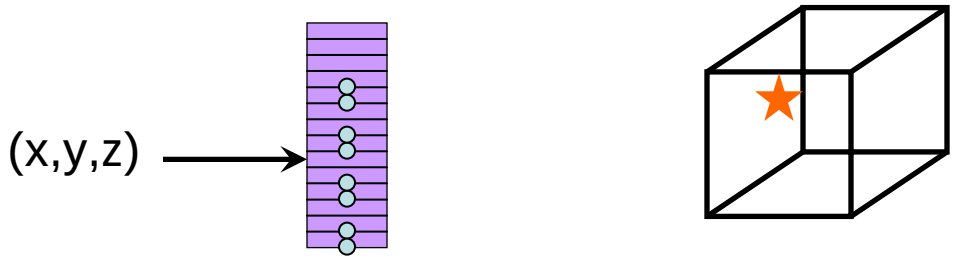
2. Convert *charge density grid* to *potential energy grid*
   - Solve Poisson's equation … $\nabla^2\Phi = \rho$

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

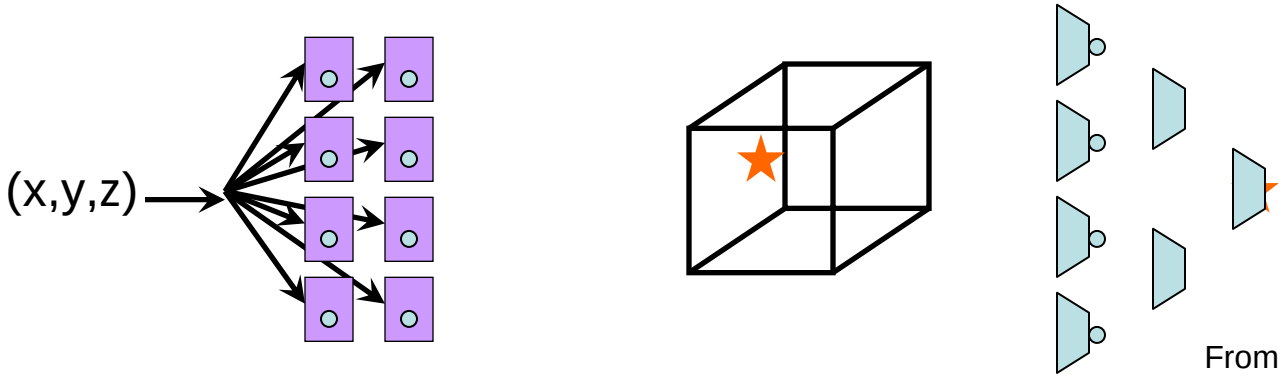
Particle-Grid (1) & Grid-Particle (3) Map Really Well to FPGAs …

Example: Trilinear Interpolation

- SW style: Sequential RAM access



- HW style: App-specific interleaving

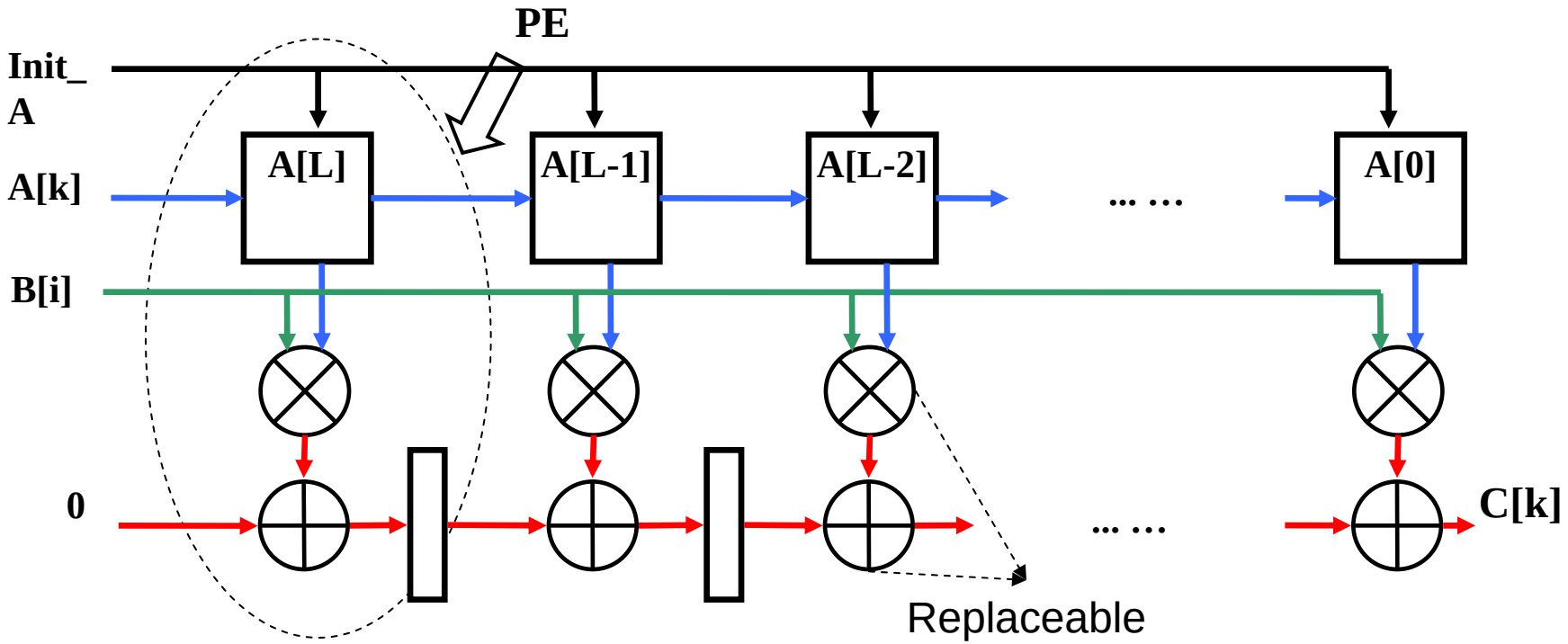


From VanCourt, *et al.* FPL06

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)

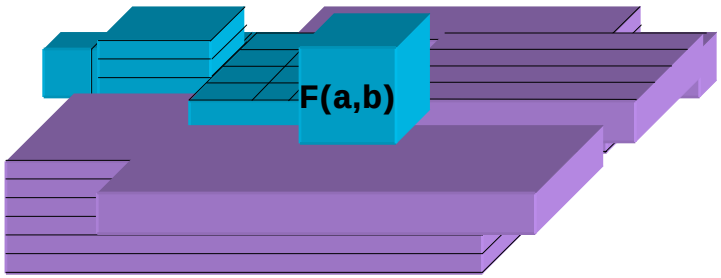


3D Grid-Grid (2)
also Maps Really Well to FPGAs …

Example: 3D Correlation

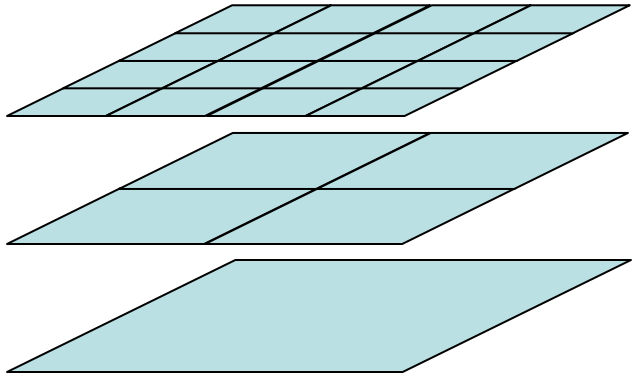
- Serial processor: Fourier transform $\mathcal{F}$
  - $A \otimes B = \mathcal{F}^{-1}(\mathcal{F}(A) \times \mathcal{F}(B))$

- FPGA: Direct summation
  - RAM FIFO

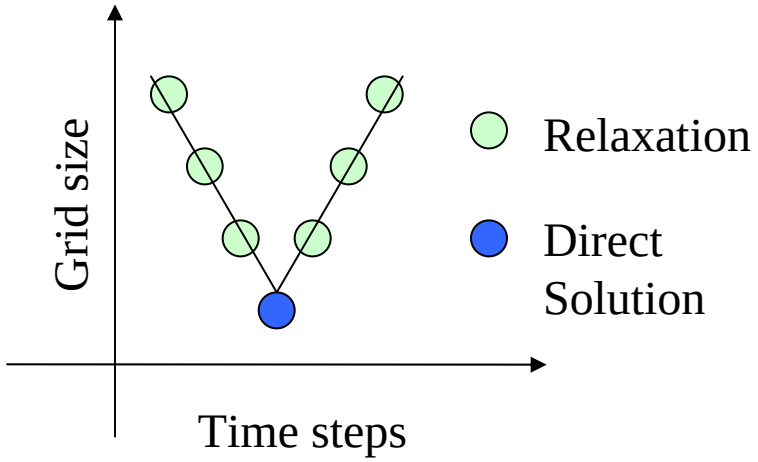


Multigrid Method

- 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



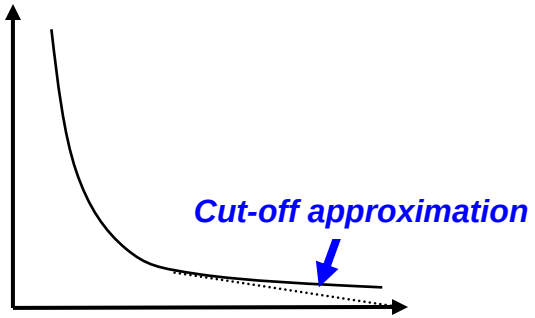
- V-Cycle
  - The down and up traversal of the grid hierarchy is called a V-cycle



Multigrid for Coulomb Force

Difficulties with Coulomb force:

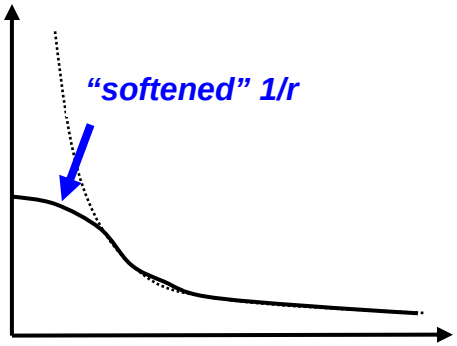
- converges too slowly to use cell lists
- cut-off is not highly accurate

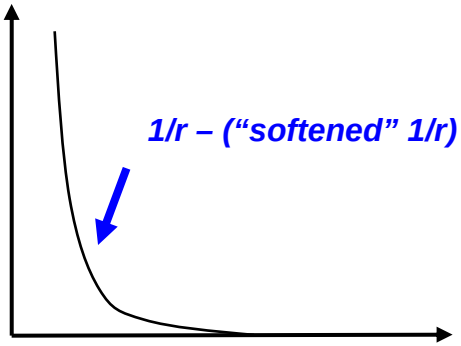


Idea:

- 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 $g_a(r)$:

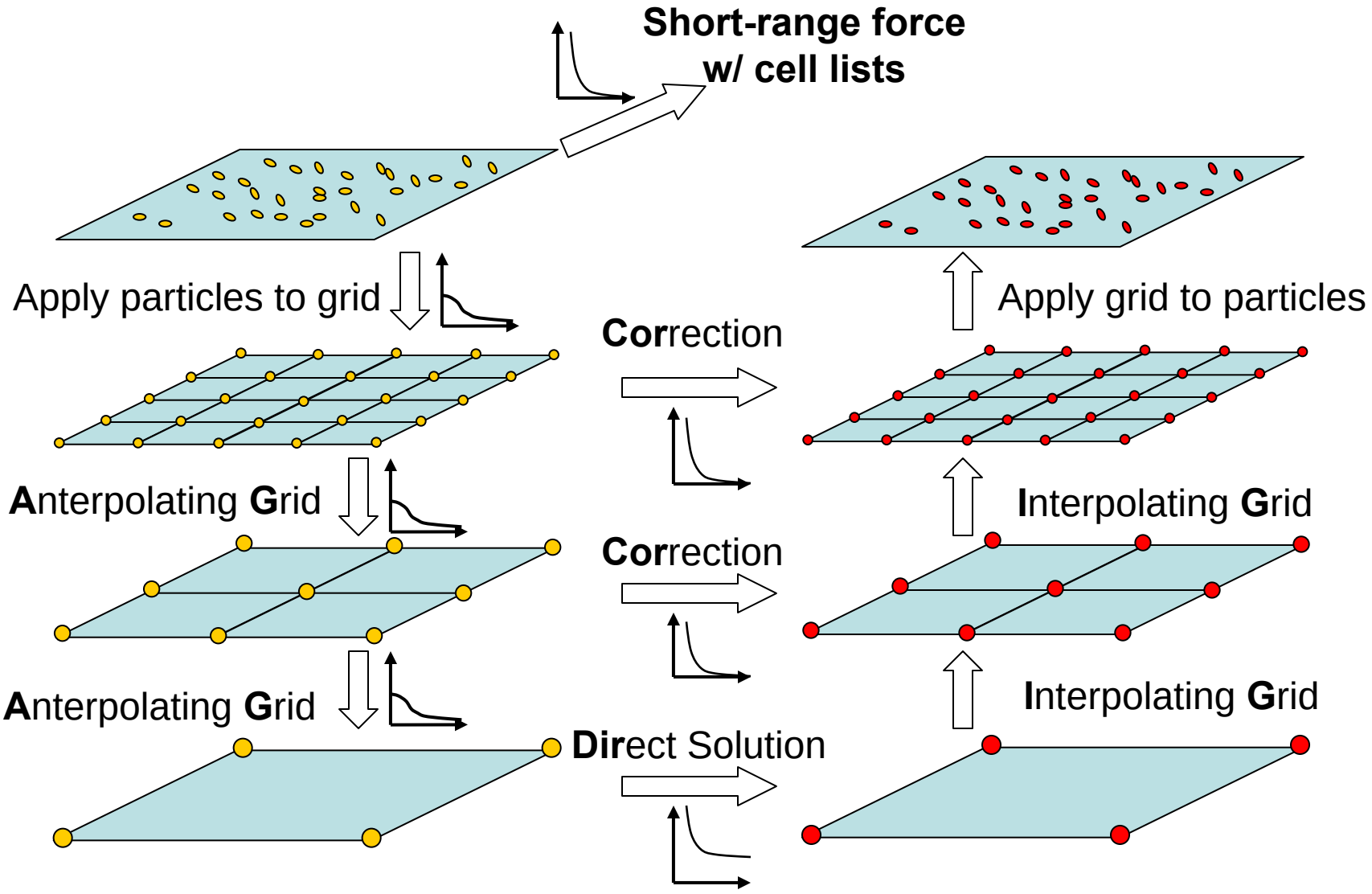
$$V_i^{CL} = \sum_{j \neq i} \frac{q_j}{|r_{ji}|} \quad \Longrightarrow \quad \frac{1}{r} = \left(\frac{1}{r} - g_a(r)\right) + g_a(r)$$

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

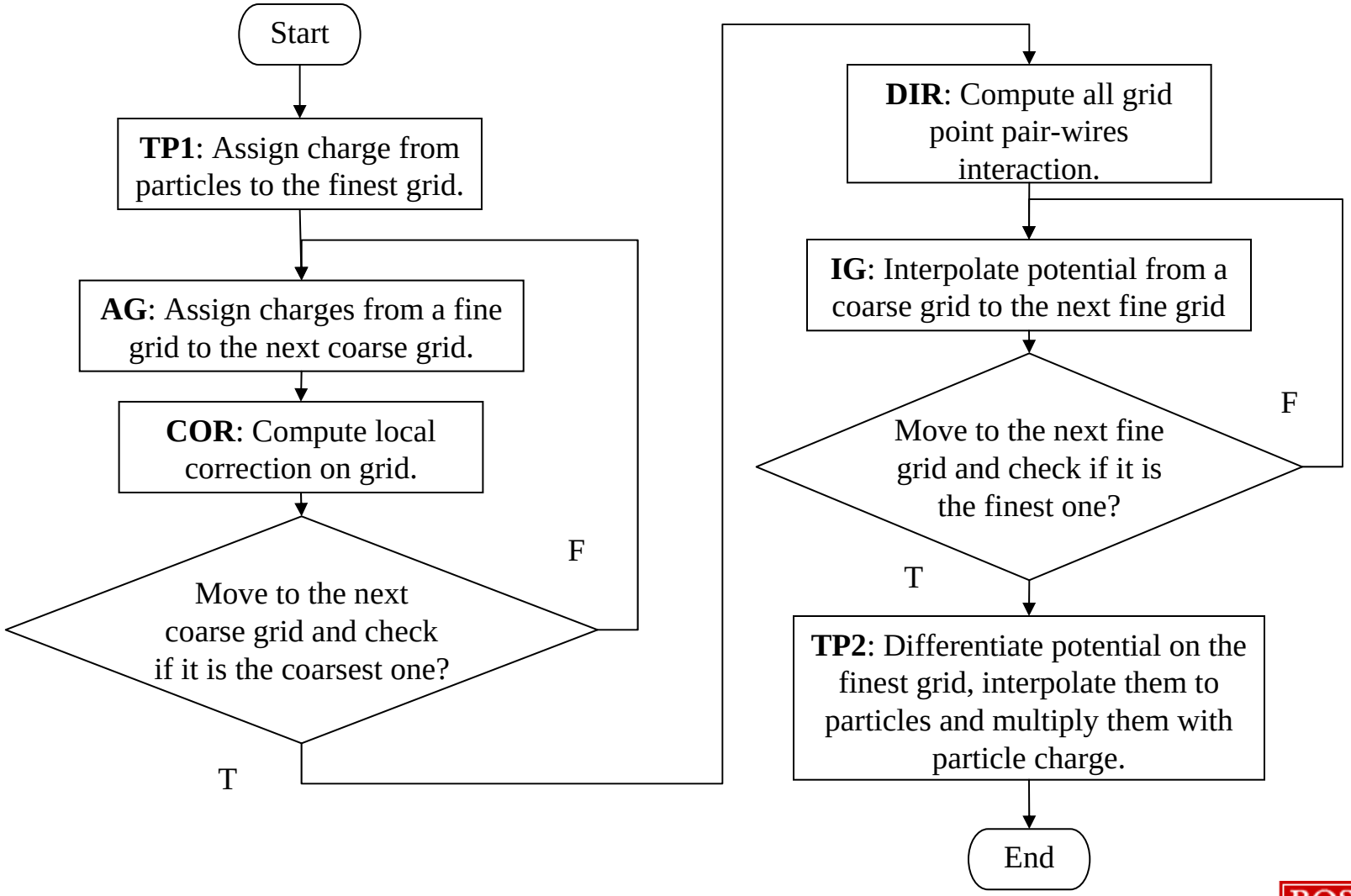
$$g_a(r) = (g_a(r) - g_{2a}(r)) + g_{2a}(r)$$

- $g_a(r)$ - $g_{2a}(r)$, 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



Multigrid for Coulomb Force



Multigrid for Coulomb Force

- **Two types of operations**
  - Particle-Grid Conversions
    - TP1 and TP2
    - One particle $\Longleftrightarrow$ P^3 neighboring grid points
  - Grid-Grid Convolution
    - AG, IG, COR, and DIR
    - Grid * Pre-computed Operators $\Longrightarrow$ Grid

Note – no relaxation steps necessary in Coulombic multigrid

Grid-Particle/Particle-Grid Computation

Step 1: We scale our coordinates to match the finest grid.

In one dimension …

particle position = **gi | oi** ➔ **gi** = grid point

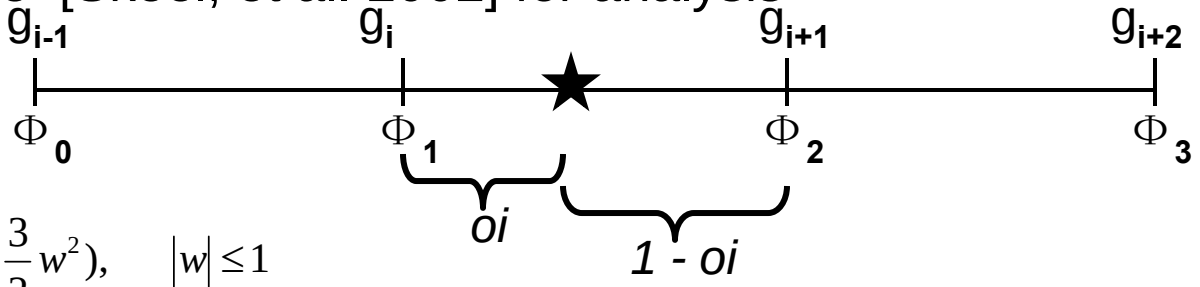
oi = distance from grid point

Step 2: We use **oi** to compute the contributions of **q** to the neighboring **gi**

Grid-Particle/Particle-Grid Computation

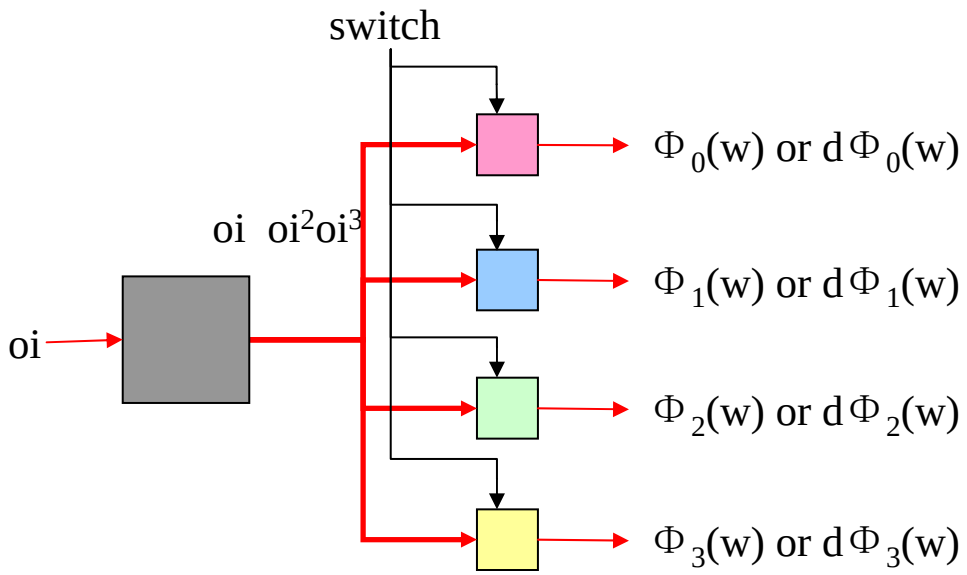
Approximate with appropriately chosen basis functions

– 3rd order, see [Skeel, et al. 2002] for analysis



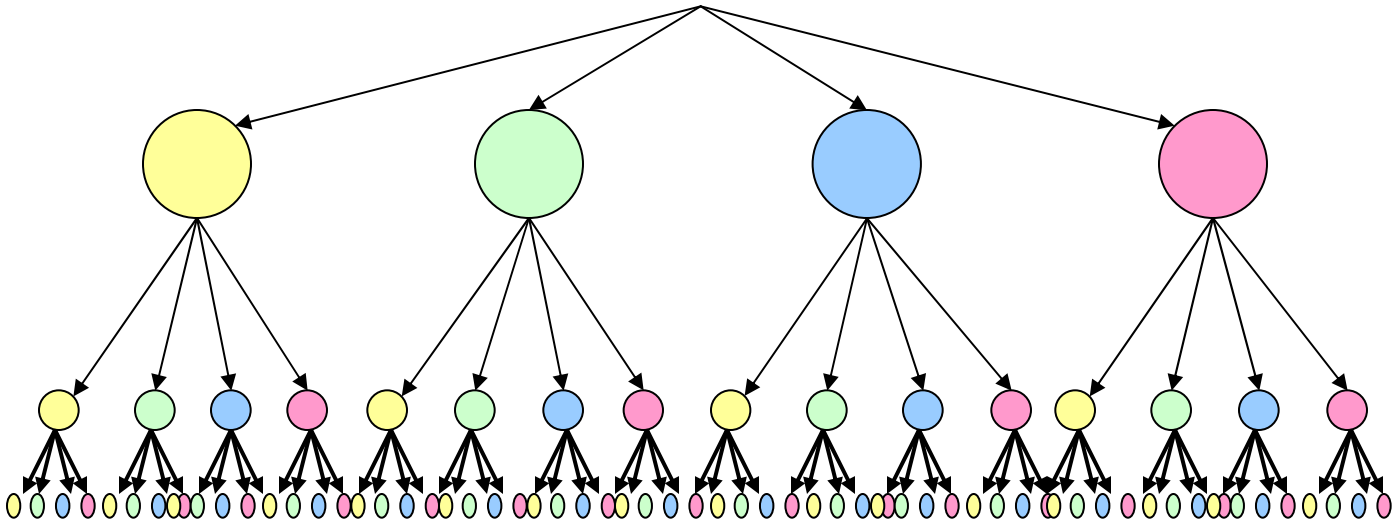
$$\phi(w) = \begin{cases} (1-|w|)(1+|w|-\frac{3}{2}w^2), & |w| \le 1 \\ -\frac{1}{2}(|w|-1)(2-|w|^2), & 1 \le |w| \le 2 \\ 0, & |w| \ge 2 \end{cases}$$

$$\begin{cases} \phi_0(oi) = -\frac{1}{2}oi^3 + oi^2 - \frac{1}{2}oi \\ \phi_1(oi) = \frac{3}{2}oi^3 - \frac{5}{2}oi^2 + 1 \\ \phi_2(oi) = -\frac{3}{2}oi^3 + 2oi^2 + \frac{1}{2}oi \\ \phi_3(oi) = \frac{1}{2}oi^3 - \frac{1}{2}oi^2 \end{cases}$$



Particle-Grid Converter

- Particle-Grid Converter
  - For P^{th} order basis functions, one particle charge is assigned to P^3 neighboring grid points through a tree structured datapath.
  - One level per dimension – P points per level



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 for correction
  - Coarse grid solved directly, i.e. grid charges are integrated to obtain grid potentials (all-to-all)

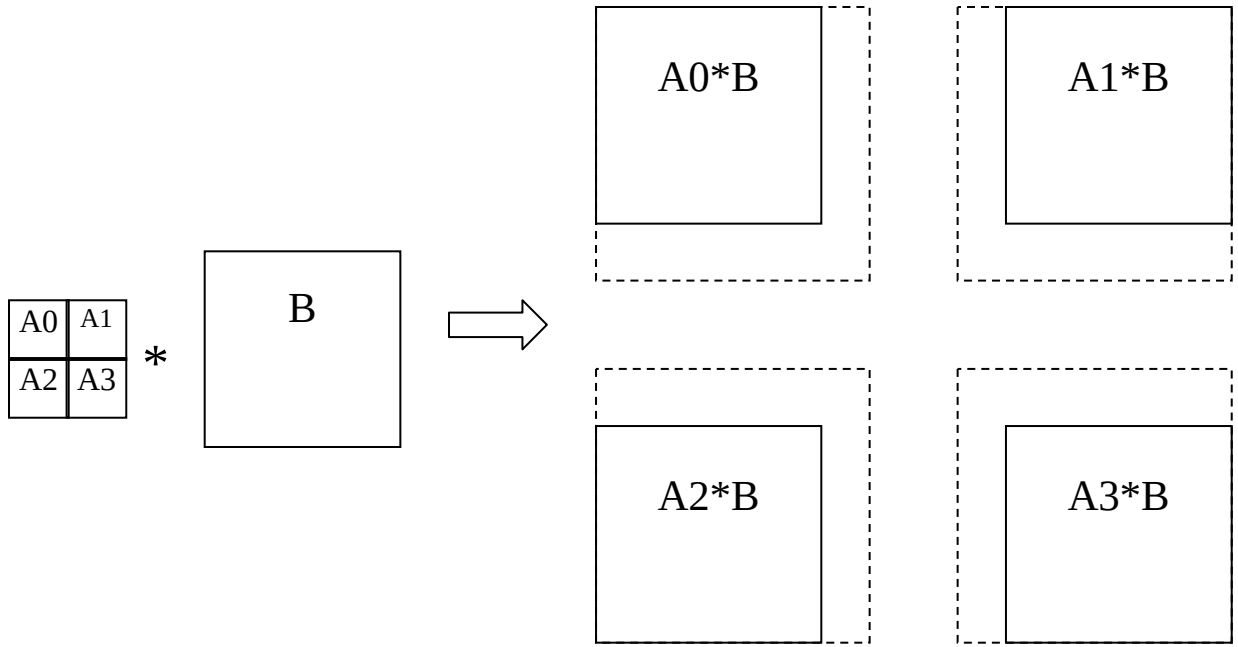
Why no more grids?

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…



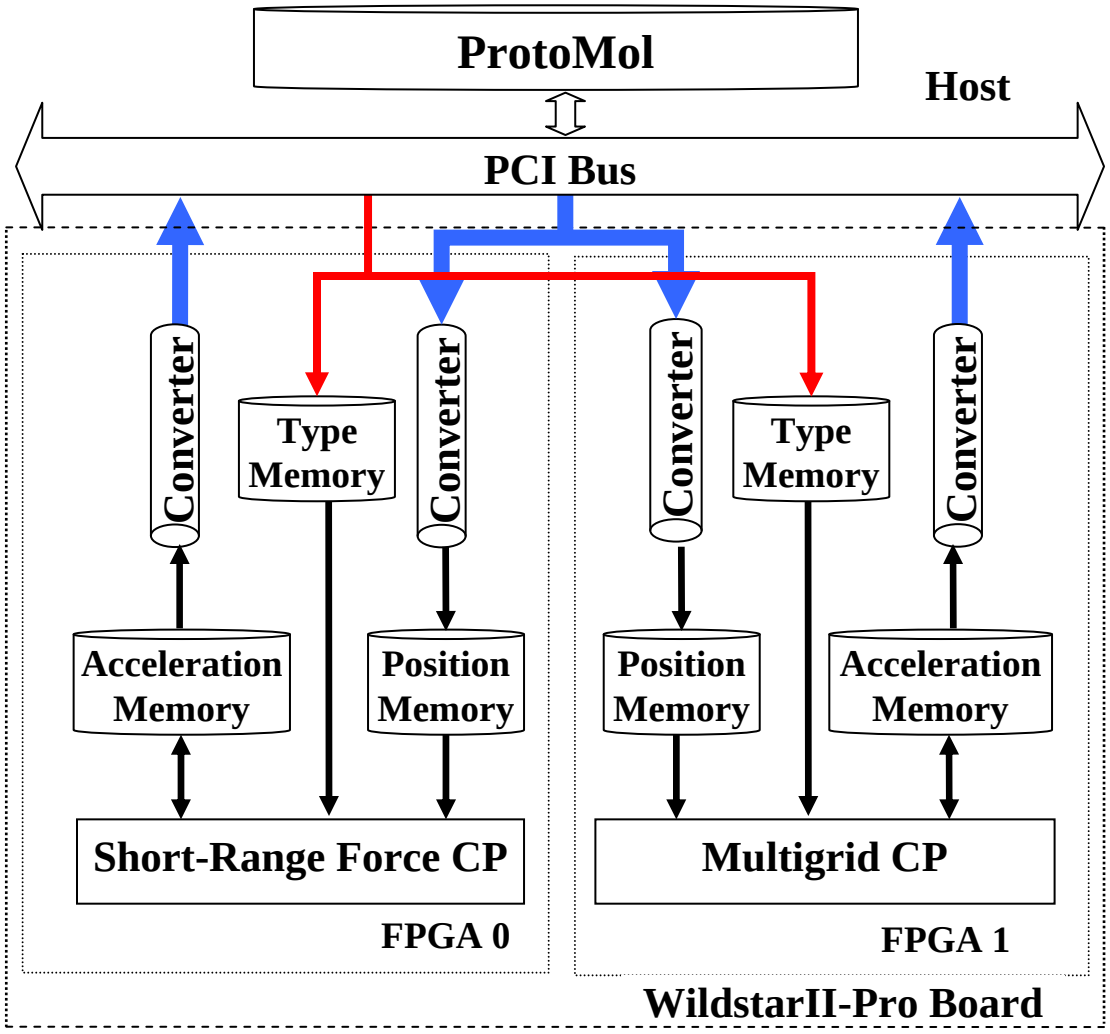
Summary – Long-Range Force Computation

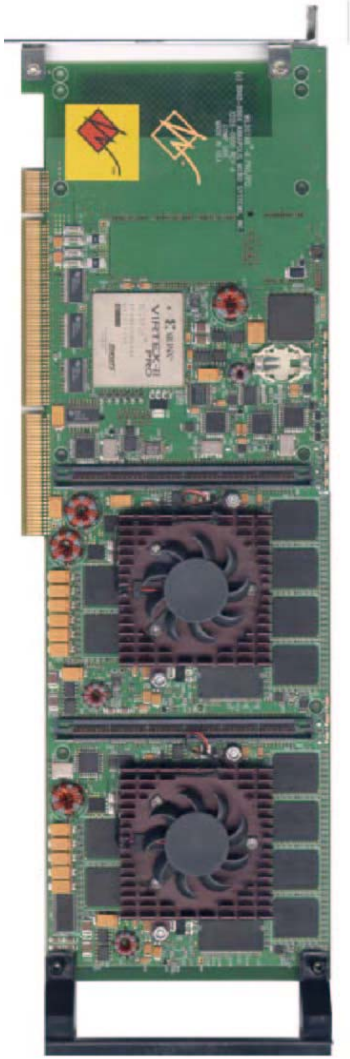
- Multigrid method
- Particle-Grid conversion
  - Interleaved memory
  - Tree-structure pipeline
- Grid-Grid convolution
  - Systolic array convolver
  - Large convolution extension

Agenda

- Motivation
- Background
- Algorithm-level designs
- **Implementation and results**
- Future directions

Implementation

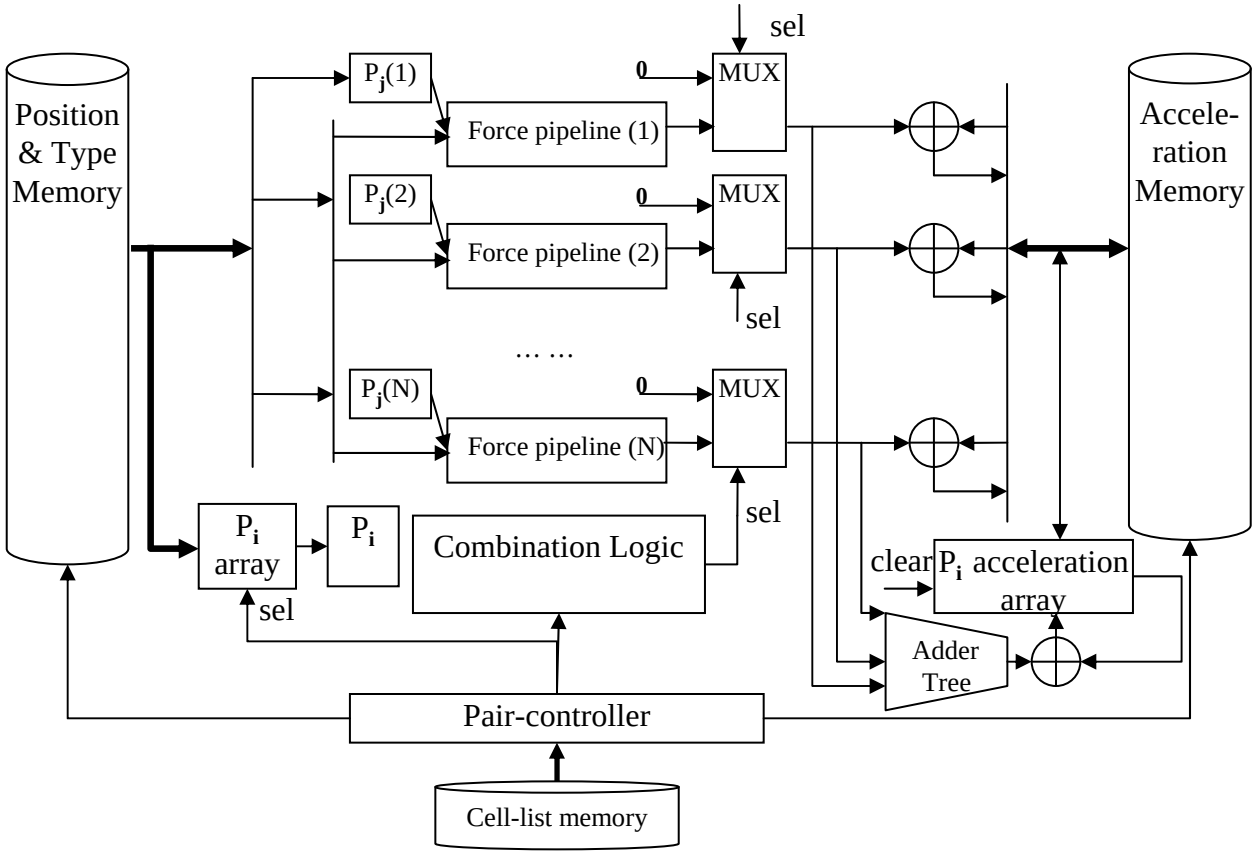




Implementation

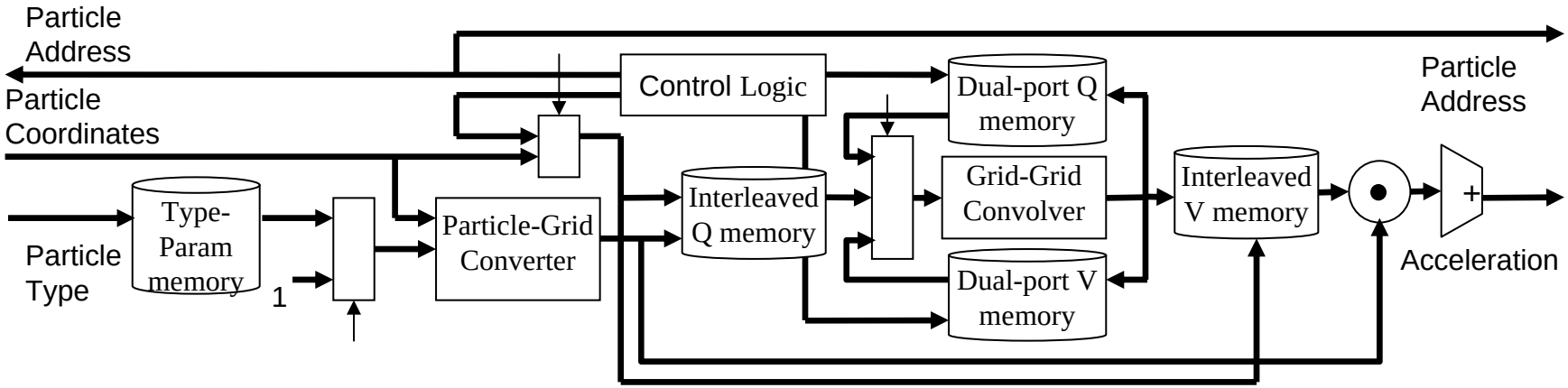
- **FPGA Platform**
  - Annapolis Microsystems Wildstar II Pro PCI Board
  - Xilinx Virtex-II Pro VP70 -5 FPGA
  - FPGA clocks at 75MHz
- **Design Flow**
  - VHDL using Xilinx, Synplicity, and ModelSim design tools
  - Microsoft Visual C++ .Net
  - Annapolis PCI driver
- **Host Platform**
  - 2.8 GHz Xeon Dell PC running Windows XP
  - ProtoMol 2.03
- **System Specification**
  - Capable of 256K particles of 32 atom types
  - Cell-lists for short-range force computation
  - Multigrid for long-range force computation
  - 2 levels of grids and the finest grid up to 32^3
  - 35-bit precision semi floating point

Short-Range Force CP Architecture



- *Please see thesis for details*

Multigrid CP Architecture



- *Please see thesis for details*

Validation Methods

What many serial codes do *(but not GROMACS)* :

- *Use DP floating point and forget about it unless something screws up*

- Monitor a physical invariant (e.g. energy) – a common measure:
  Example: Relative rms fluctuation of total energy [Amisaki 1995]

$$Minimize \quad E_{fluct} \equiv \frac{\sqrt{|\langle E^2 \rangle - \langle E \rangle^2|}}{|\langle E \rangle|}$$

Example: ratio of fluctuations between E_{total} and $E_{kinetic}$ $R < .05$

[van der Spoel 2004] $$R \equiv \Delta E_{total} / \Delta E_{kinetic} < .05$$

- Monitor the error between different methods
  - Measure the error of force and energy between the direct computation and the alternative methods [Skeel, et al. 2002]

$$F_{avg} = \frac{N^{-1} \sum_i (m_i^{1/2} \left\| \tilde{\vec{F}}_i - \vec{F}_i \right\|)}{N^{-1} \sum_i (m_i^{1/2} \left\| \vec{F}_i \right\|)} \qquad U_{pot} = \left| \frac{\tilde{U} - U}{U} \right|$$

Validation Methods

Validated against two serial reference codes:

1. ProtoMol 2.03
2. Our own bit accurate *Hardware Tracker* code (multiple versions)

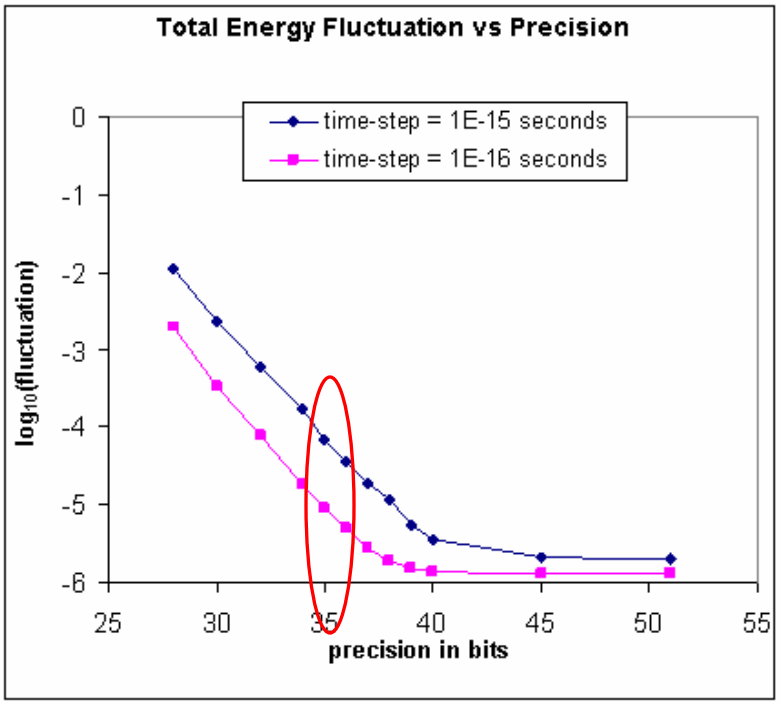
Validation, part1: ProtoMol matches HardwareTracker when HardwareTracker has a floating point datapath

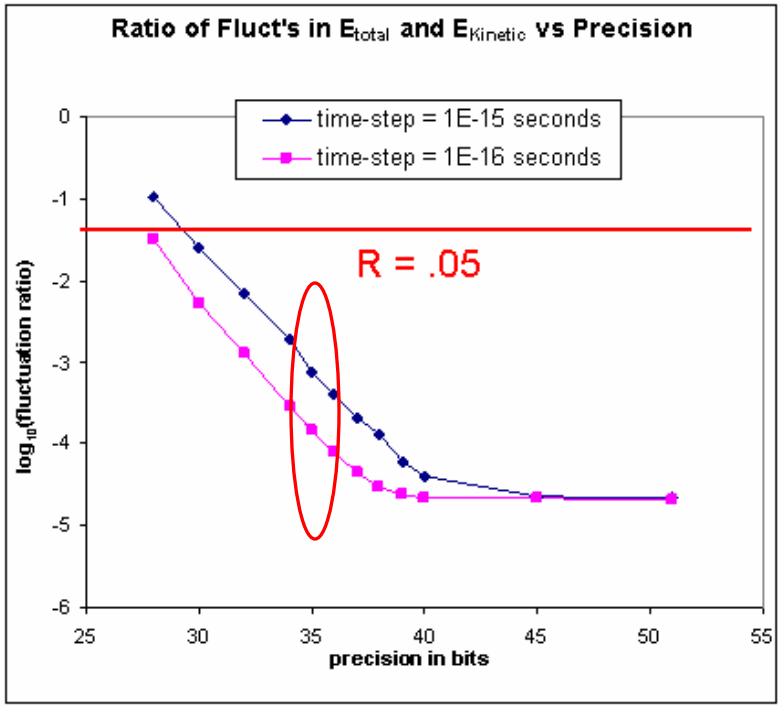
Validation, part2: HardwareTracker matches FPGA implementations for all semi fp datapath sizes

The missing link: *Floating point can only be compared indirectly with lower precision datapaths (see previous)*

Arithmetic – Precision

- Measuring energy fluctuation under different precision
  - 35-bit precision is a sweet spot for both numerical accuracy and FPGA resource mapping.

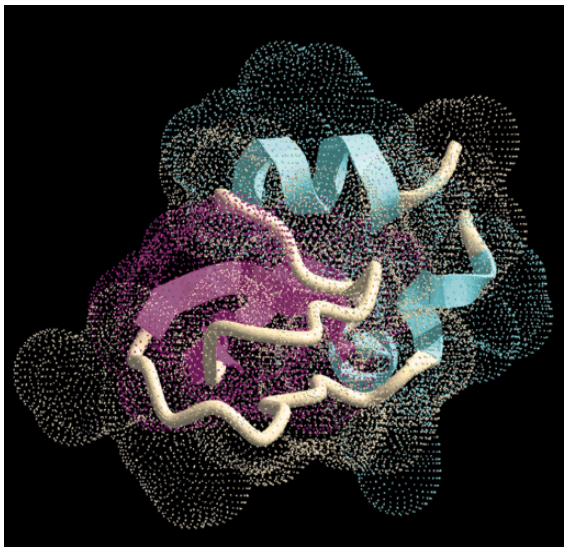




Results – Validation

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

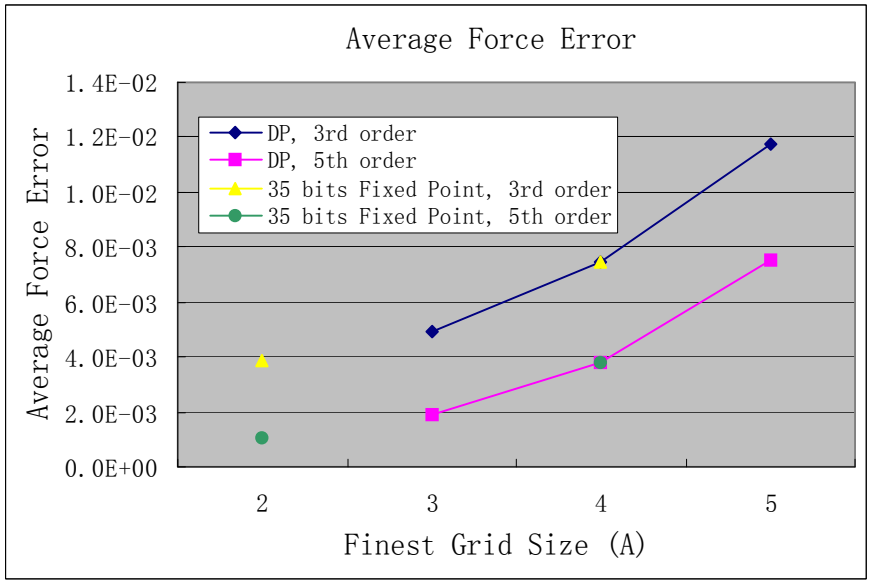
    (similar results with larger model)

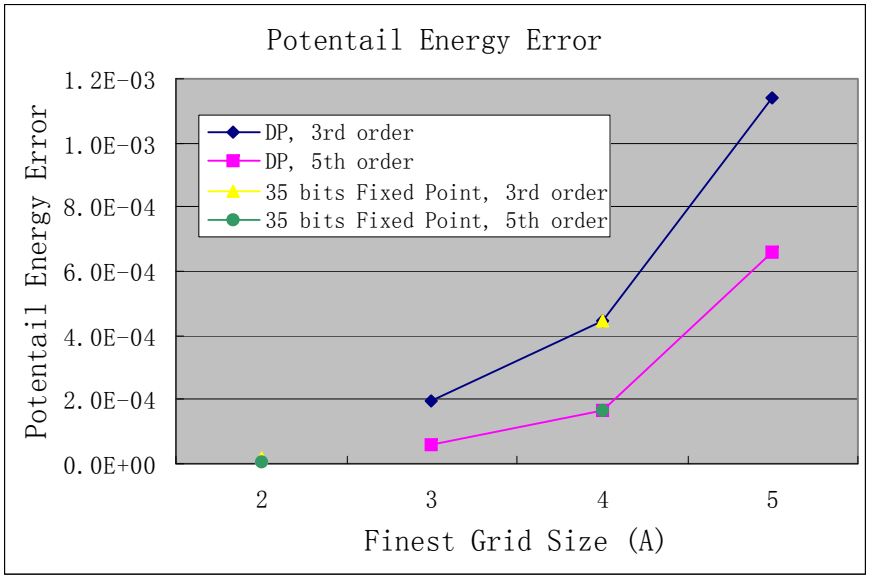
- Energy Fluctuation:
  - Both versions have relative rms energy fluctuations ~$3.5*10^{-4}$

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

Results – Validation

- Average force error and potential energy error:
  - The difference between DP multigrid and 35-bit semi-FP multigrid is ~10^{-4} of the difference between DP direct computation and DP or 35-bit semi-FP multigrid method.





$$F_{avg} = \frac{N^{-1}\sum_i (m_i^{1/2} \left\| \tilde{\vec{F}}_i - \vec{F}_i \right\|)}{N^{-1}\sum_i (m_i^{1/2} \left\| \vec{F}_i \right\|)}$$

$$U_{pot} = \left| \frac{\tilde{U} - U}{U} \right|$$

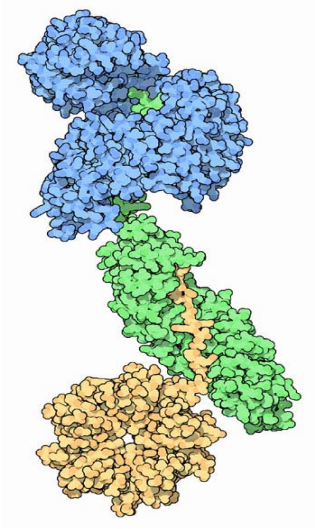
Results – MD Performance

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

Short-range force speed-up
- 11.1x over software version of original ProtoMol

Multigrid speed-up
- 3.8x over software version of **multigrid** in original ProtoMol
- 2.9x over software version of **PME** in NAMD

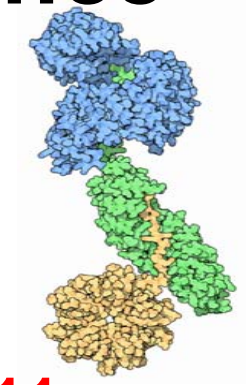
Total speed-up
- 9.8x over original ProtoMol
- 8.6x over NAMD



		Short Range Forces	**Long Range Forces**	Bonded Forces	Motion Integration	Comm. & overhead	Init. & misc.	**TOTAL**
FPGA	FPGA Accelerated **ProtoMol** (2 VP70s) • Multigrid every cycle	348.3	**61.3** (Multigrid)	21.5	20.8	25.6	9.2	**425**
serial	Original **ProtoMol** • Multigrid every cycle	3867.8	**234.1** (Multigrid)	21.6	21.5	0	12.9	**4157**
serial	**NAMD** • PME every cycle		**177.3** (PME)					**3726**

Extended Results – MD Performance

System Tested	Performance
Serial configuration as described	
ProtoMol w/ multigrid every cycle	4.2
NAMD 2.6 w/ PME every cycle	3.7
NAMD 2.6 w/ PME every 4th cycle	3.2
Accelerated configuration as described (2 VP70s)	
ProtoMol w/ multigrid every cycle	.43
ProtoMol w/ multigrid every 4th cycle (dynamic reconfiguration)	.29
ProtoMol w/ multigrid every 4th cycle (dynamic reconfiguration), reduced precision	.21
Accelerated configuration, simulation only	
ProtoMol w/ single V2 VP100 w/ multigrid every 4th cycle (dynamic reconfiguration)	.36
ProtoMol w/ single V2 VP100 w/ multigrid every 4th cycle (dynamic reconfiguration), reduced precision	.33
ProtoMol w/ single V5 LX330T w/ multigrid every 4th cycle (dynamic reconfiguration)	.32
ProtoMol w/ single V5 LX330T w/ multigrid every 4th cycle (dynamic reconfiguration), reduced precision	.19
From NAMD website	
NAMD w/ PME every 4th cycle 90K particle Model	2



11x

6.5x

Can we get better performance?

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

2. 35-bit precision is expensive

   (even with semi FP; full FP would be much worse)
   - Hard FP cores would help

3. Improved design
   - *We've done little optimization*

Extended Discussion – MD on FPGAs

What's a factor of 5x (or 10x) worth?

- *it's not a factor of 50x [Buell RSSI2006]*
- *What do you expect running at 75MHz on a three year old FPGA without hard FP units???*
- *1/5th of 100 days is 20 days !! [Pointer 2006]*
- *NAMD has been seriously optimized*
  - *although perhaps not as much as GROMACS …*
- *Power savings is good, should be advantageous in total cost of ownership …*
- *MD is really important …*

Questions?