Case Studies in FPGA Acceleration of Computational Biology and their Implications to Development Tools*

Martin Herbordt Tom VanCourt Yongfeng Gu
Bharat Sukhwani Josh Model Al Conti
Doug DiSabello

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, the Naval Research Lab, and by Lincoln Labs

The Problem

Potential performance of FPGAs for HPC is enormous:

- ***Parallelism*** (up to 10,000x for low precision computations)
- ***Payload*** per computation, rather than control (about 10x)

Challenges *(that we can't do anything about):*

- ***Low operating frequency*** (1/10 x)
- ***Amdahl's law***

Therefore →

Performance of HPC using FPGAs is therefore unusually sensitive to the quality of the implementation.

Or more bluntly →

The potential performance is enormous, but it's much easier to get nothing at all.

How hard is it? *(from Snyder86*)*

Fundamental Law of Parallel Computation

A parallel solution utilizing ***P*** processors can improve the best sequential solution by at most a factor of ***P.***

Corollary of Modest Potential

Physical problems tend to have 3rd or 4th order complexity.

Parallel Computation therefore offers only modest benefit ($P^{1/3}$ or $P^{1/4}$).

Thesis

Overhead must be scrupulously avoided in the implementation of parallel systems, both in languages and in architectures. Because the benefit is so modest, the whole force of parallelism must be transferred to the problem, not converted to "heat" in implementational overhead.

Application → FPGA *is much harder than* **Application → MPP**

*Ann. Rev. of Computer Science (1986)

How hard is it, cont.

… and also portability, productivity …

Overview

Body of talk:

12 ways to avoid generating heat as derived from our experiences with BCB

Motivation 1:

the 2x to 100x losses in performance avoided are important

Motivation 2:

if we want to automate application development, then these must be made part of that process (if we are to do more than struggle to break even)

Some ways to avoid generating “heat”

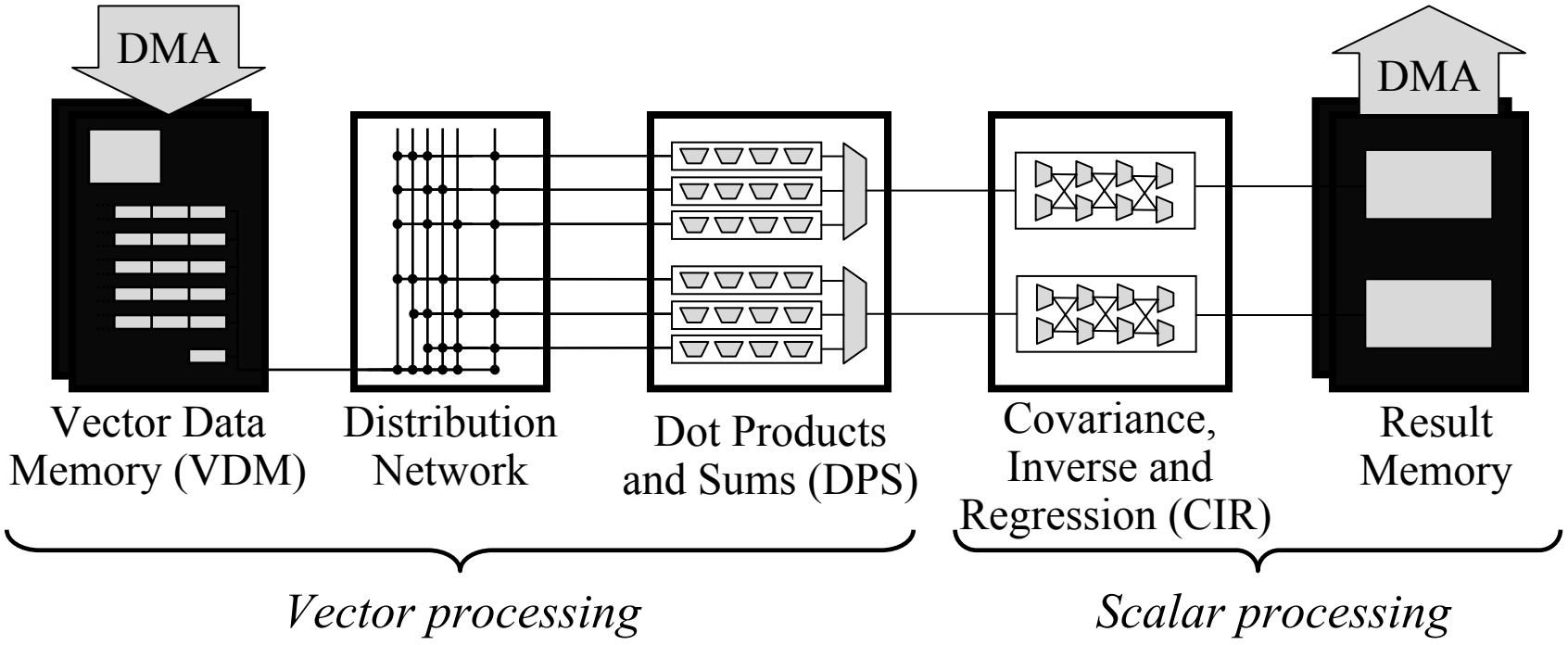
1. Use the correct programming model.
2. Use an appropriate (FPGA) algorithm.
3. Speed match sequences of functionally different computations.
4. Scale computation to use maximal chip resources.
5. Hide latency of independent functions.
6. Use appropriate constructs.
7. Use FPGA resource types appropriately.
8. Arithmetic 1: Use appropriate precision.
9. Arithmetic 2: Use appropriate operations
10. Arithmetic 3: Use appropriate mode
11. Support application family, not point solution
12. Proper memory access

Note: these are not exhaustive, they overlap, but not one has anything to do (necessarily) with Verilog/VHDL !

Speed-Matching[1,2]

Scenario 1: Data passes through sequence of functions, where timing of functions varies drastically

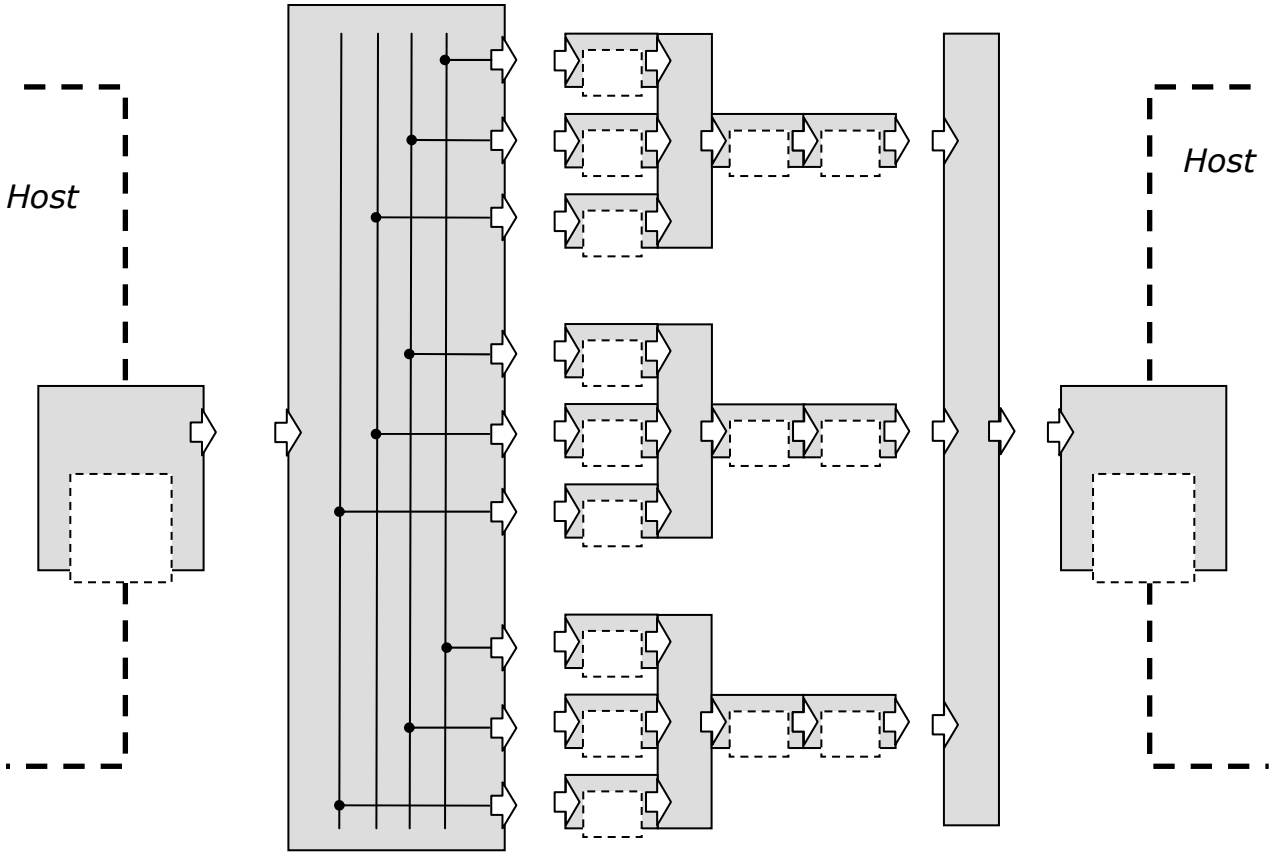
Example: *(From microarray analysis)* CIR 10x faster than DPS



1. FPL 2003
2. JMM 2004

Speed-Matching, cont.

FPGA Solution: Replicate slower units 10x for each fast unit



Hide Latency of Independent Functions[3,4,5]

Scenario 2: Independent functions *(e.g. that generate parameters)*

Example:

- In object finding (docking), rotate 3D image (molecule) for each correlation
- Method: retrieve voxels in "rotated" order

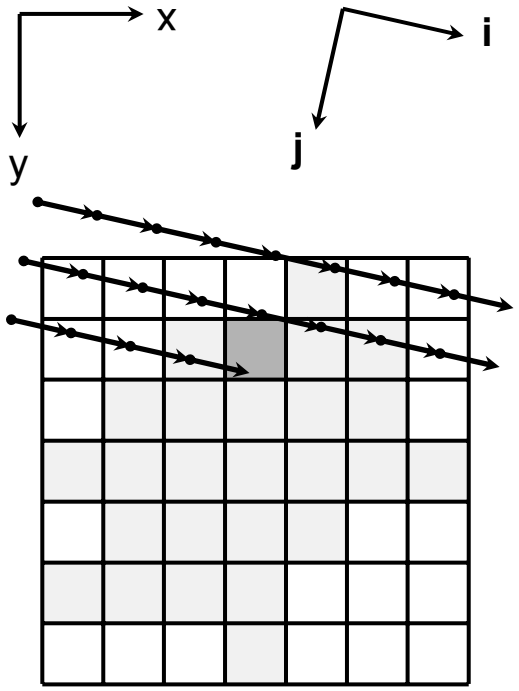
- Express (i,j,k) in (x,y,z) basis

  $i=(x_i, y_i, z_i)$ $j=(x_j, y_j, z_j)$ $k=(x_k, y_k, z_k)$
- Traverse (i,j,k) index space
- Find (x,y,z) from (i,j,k)

$$\begin{pmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ z_i & z_j & z_k \end{pmatrix} \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Round and range check
- Pipelined, parallel computation

  gives ***~0 ns*** overhead for rotation



3. FPL 2004a
4. CAMP 2005
5. JASP 2006

Hide Latency, cont.

A solution: precompute indices and load as needed.

Problem: 1MB per pose, thousands of poses, lots of data!

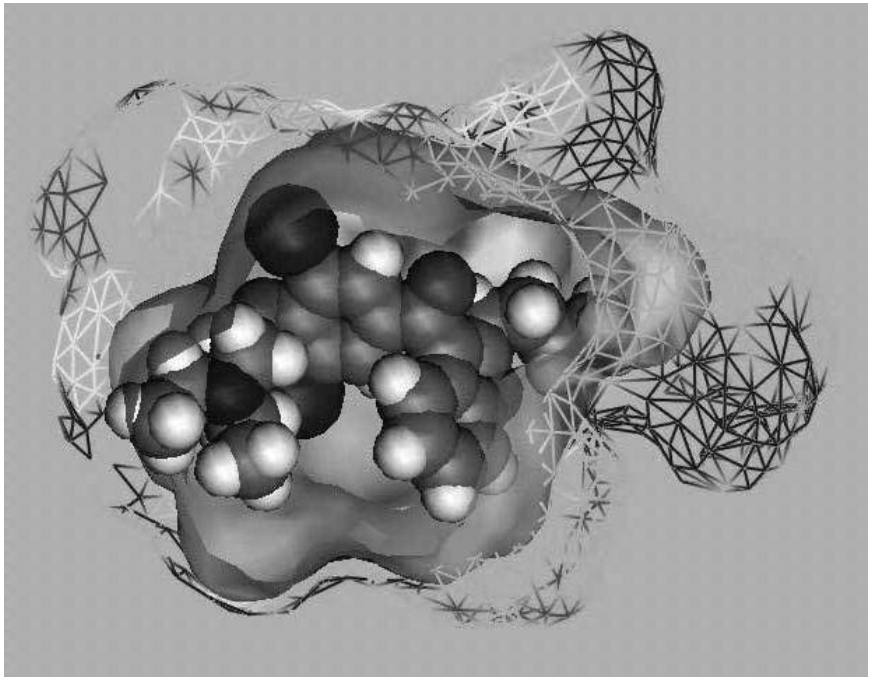
FPGA Solution: Separate hardware computes "rotated" indices and delivers them just-in-time to voxel fetch unit.

- 20 parameter function, but only takes a few percent of VP70
- can be pipelined to generate indices at operating frequency

Select FPGA-Optimal Algorithm[3,5]

Scenario 3: multiple known algorithms for a task; different ones are optimal for RAM and FPGA

Example: Modeling interactions of rigid molecules with correlation



From:http://www.biograf.ch/images/publications/chemmedchem/2006_1
Lill, et al. ChemMedChem **1** (2006)

3. FPL 2004a
5. JASP 2006

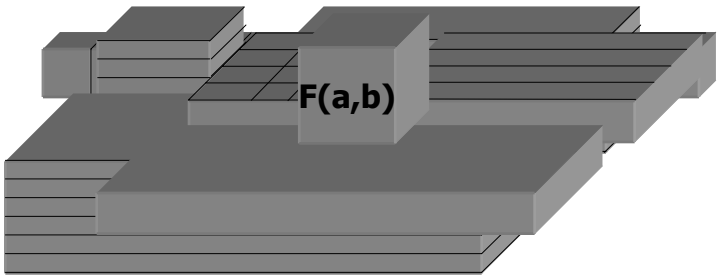
Algorithm Selection, cont

Serial processor preferred method: Fourier transform $\mathcal{F}$

– $A \otimes B = \mathcal{F}^{-1}(\mathcal{F}(A) \times \mathcal{F}(B))$

FPGA Solution: Direct application of correlation

– RAM FIFO



Take Advantage of FPGA Hardware[6]

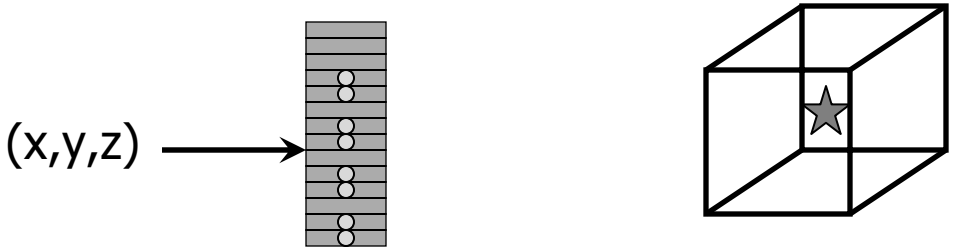
Scenario 4: FPGA has unusual but extraordinarily powerful features, such as hundreds of independently accessible quad-ported memories (VP100)

Example: Use highly parallel memory access in trilinear interpolation

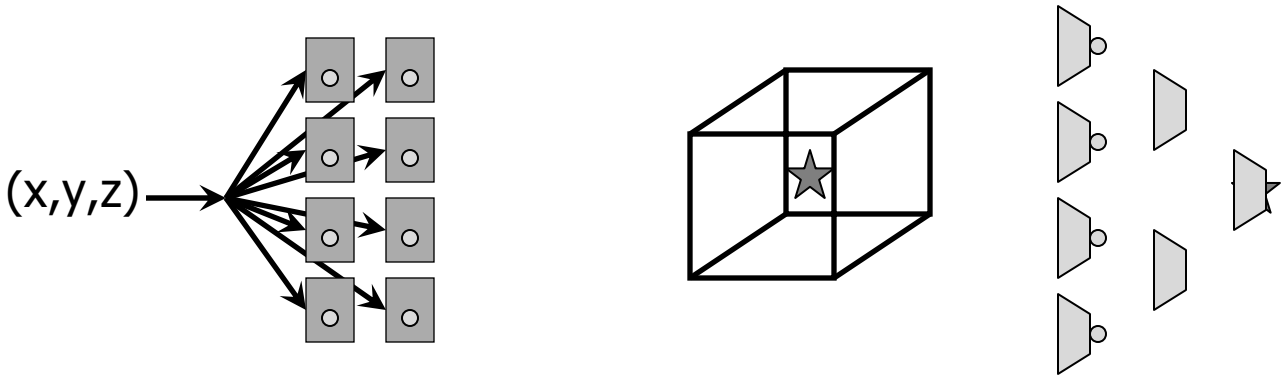
6. FPL 2006a

FPGA Hardware, cont.

C style: Sequential RAM access



FPGA Solution: App-specific interleaving

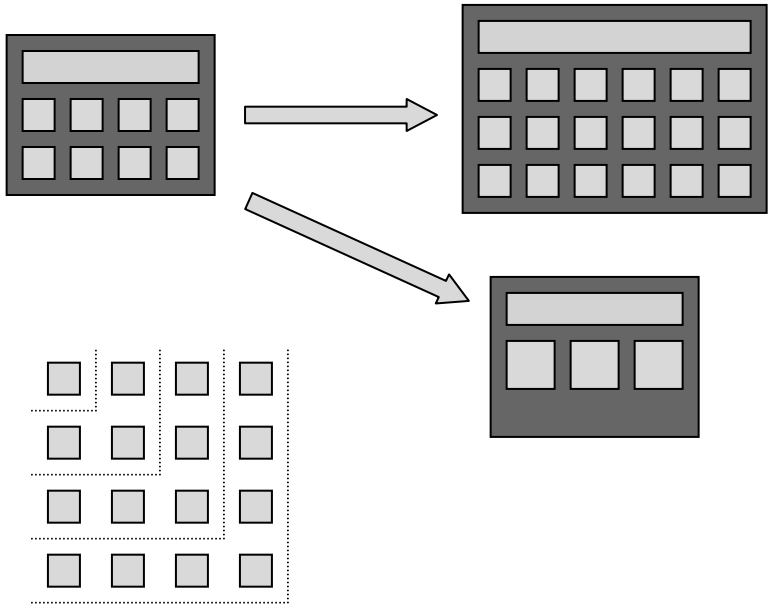


Scaling applications to FPGAs[7]

Scenario 5: Scale application to maximal size given target hardware

- Hardware (invariably) varies

- Scaling depends on:
  - FPGA capacity
  - Application details
  - Computing array



7. FPL 2006b

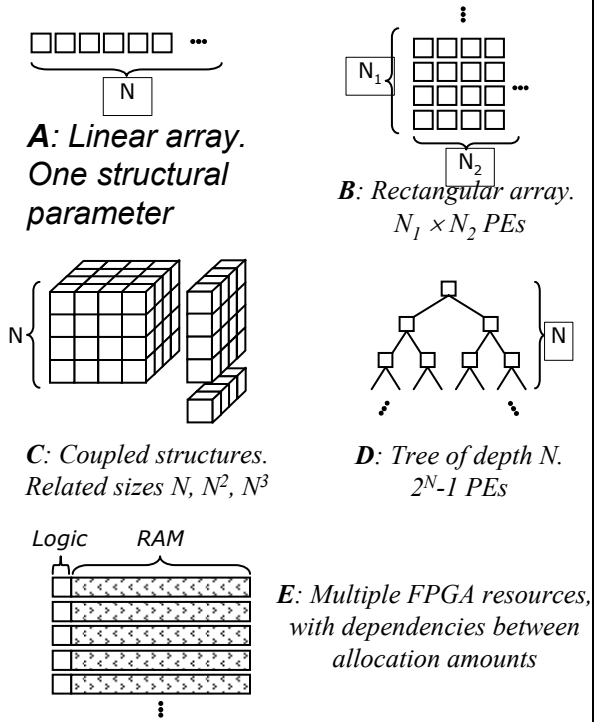
Scaling Applications, cont.

Scaling is often open-ended and complex, rather than fixed function:

$$N_{opt} = argmax\ U(N)(2)N \mid V(N) \wedge \{ \forall j : r_j^F \geq S_j(N, B) \}$$

FPGA Solution: build into design tools →

- Support for complex parameterization
- Fast (synthesis) estimation of component attributes → size, timing



Growth laws for computing arrays specified in terms of structural parameters

Use Appropriate Precision[all]

Scenario 6: Application data type size is non-standard

Examples:

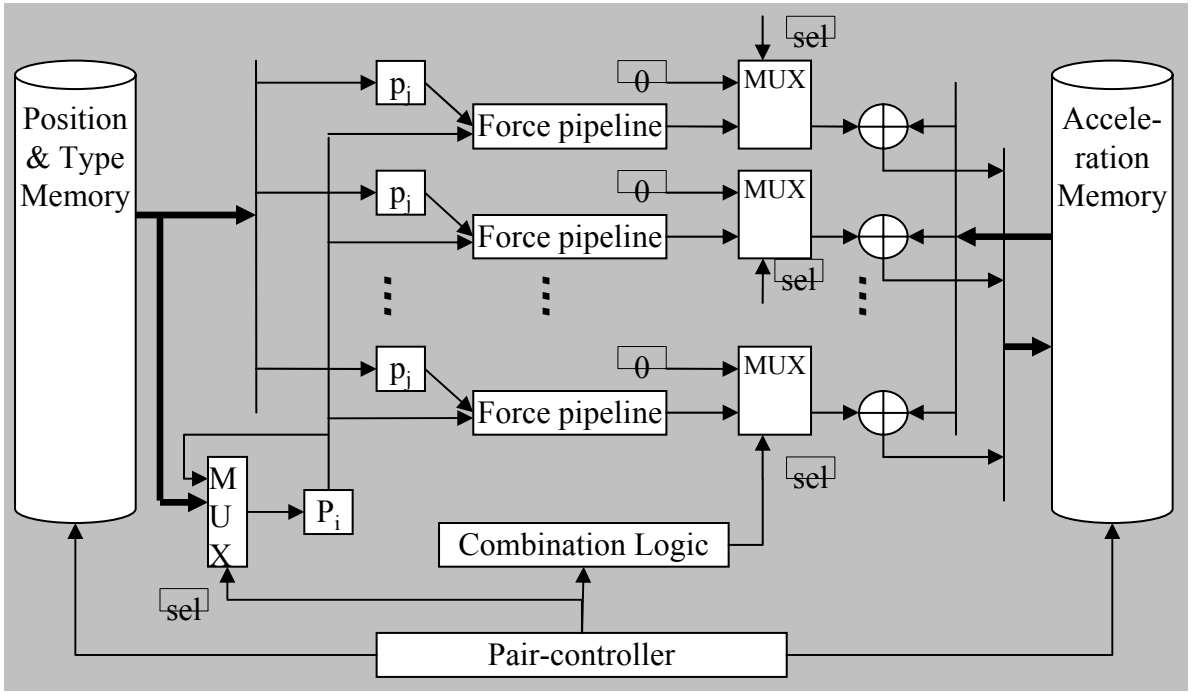
Amino acid	5-6 bits
Nucleic acid	2-4 bits
Microarray spot intensity (log ratio)	2-4 bits
Spatially mapped steric component	1-2 bits
MD measures	25-45 bits

FPGA Solution: trade off (unneeded) precision for parallelism

- Datapath uses what's needed
- Extra FPGA components used for datapath replication

Use Appropriate Precision, cont.[8]

Example:
In MD force computation, varying precision changes possible number of parallel pipelines



Platform	Precision (bits)	Pipe-lines	HW mult's used (% of usage)	Block RAMs used (% of usage)	Delay (ns)	Speed-up
VP70 AMS	35	4	176(53%)	214(65%)	11.1	50.8×
VP70 AMS	40	4	264(80%)	251(77%)	12.2	46.4×
VP70 AMS	45	4	288(88%)	285(87%)	13.2	42.7×
VP70 AMS	51	4	288(88%)	317(97%)	18	31.3×
VP70 AMS	35	8	256(78%)	326(99%)	22.2	51.0×
VP100 sim	51	4	288(65%)	317(77%)	13.6	41.5×
VP100 sim	35	8	256(58%)	334(75%)	12.8	88.5×

8. IEE CDT 2006

Use Appropriate Arithmetic Mode[9]

Scenario 7: integer or floating point not always optimal

Example1: log-based arithmetic

Example 2: MD force computation →

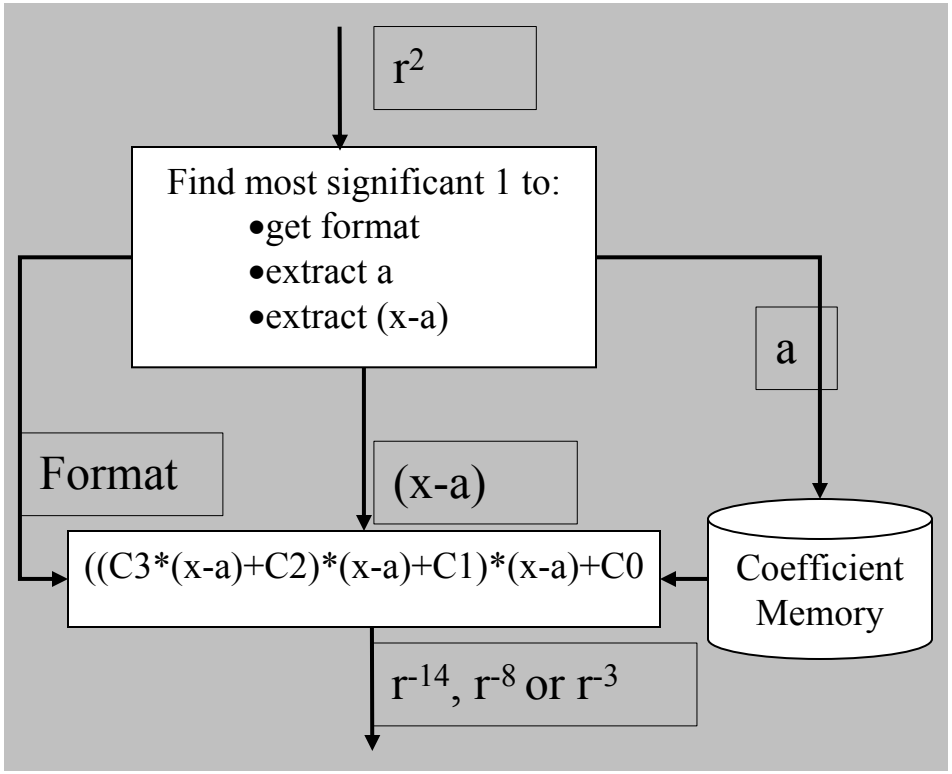
- precision critical, but not canonical (e.g. 24,32,53,64)
- dynamic scaling critical, but over a limited predictable range

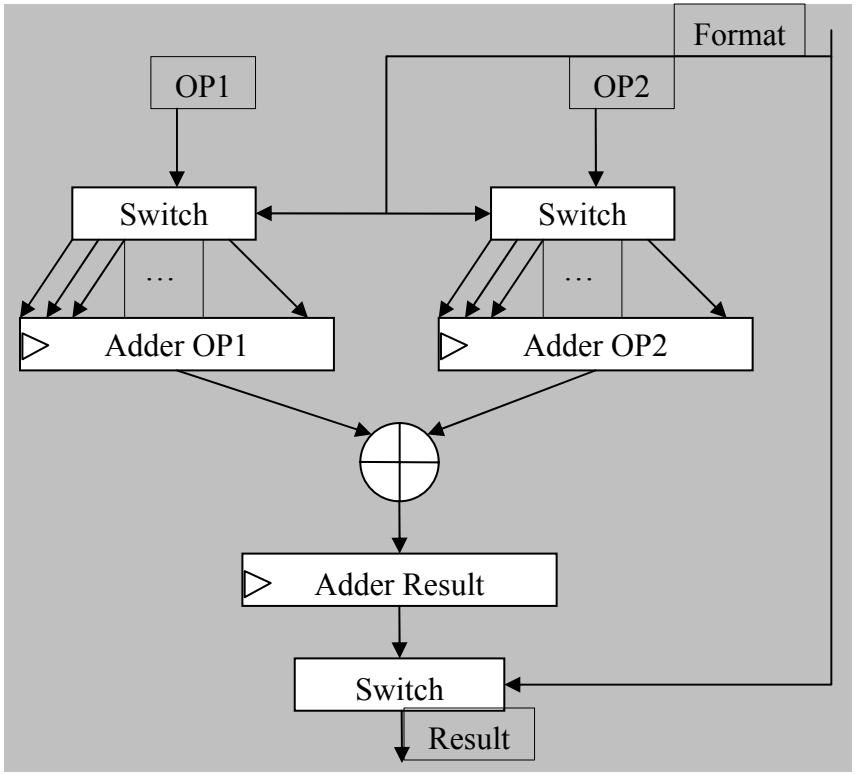
FPGA Solution: "Semi" Floating Point

- Exponent known from index into look-up table
- Table entries have predetermined ranges

9. FPL 2006c

Use Appropriate Arithmetic Mode, cont.





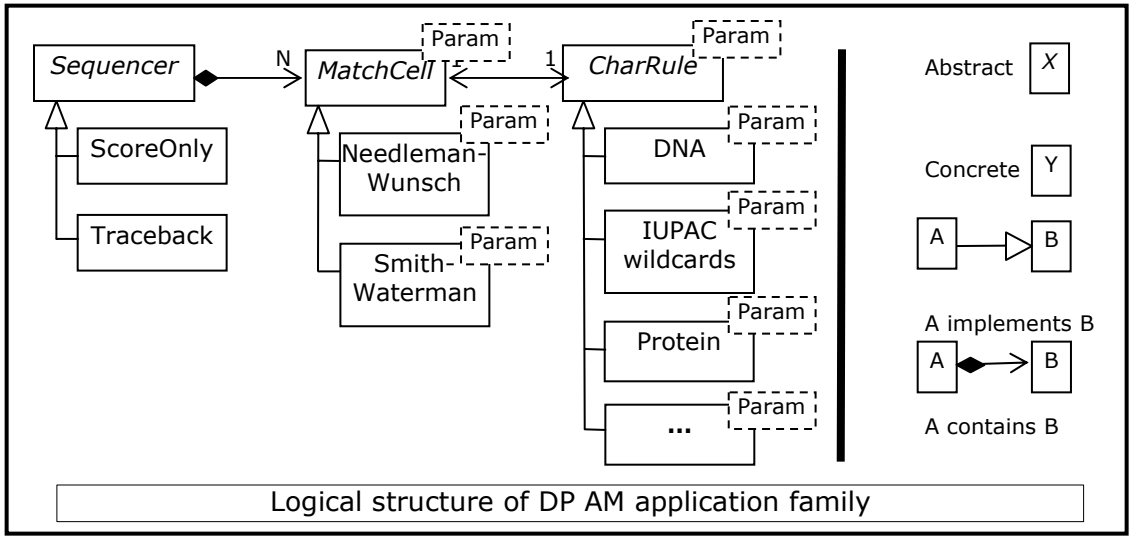
Format	Add	Mul	Pipeline Total
LogiCore DP	692	540	19566
LogiCore FP	329	139	6998
Semi fp 35-bit	70	390	n.a.
Integer 35-bit	18	400	n.a.
Combined semi, int	n.a.	n.a.	5624

Applications (often) come in families, not point solutions[10,11]

Scenario 8: Application has large number of complex variations

- Passes function as parameter

Example: Approximate string matching using dynamic programming



Logical structure of DP AM application family

FPGA Solution: True object-oriented support

10. ASAP 2004
11. JMM 2006

good SW data structure ≠ good HW structure[12]

Scenario 9: FPGA implementation of common software data structures and constructs

Examples:

- FIFO, Priority Queue, Tree, Stack
- Search
- Reduction
- Parallel Prefix
- Suffix Trees
- Corner Turning *(thanks Duncan!)*

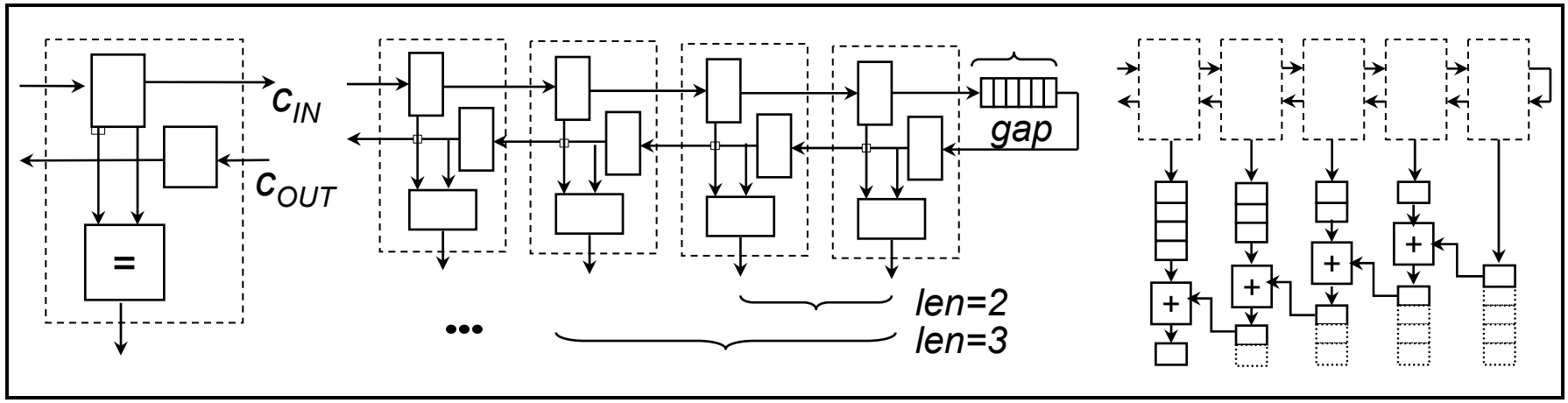
FPGA Solution: *Various well-known hardware structures*

12. FPL 2004b

Standard HW structures, cont.

Example: Finding palindromes of various lengths, and with arbitrary gap size, at streaming rate

- Use well-known palindrome structure



good SW mode ≠ good FPGA mode[13]

Scenario 10: common modes of computation

Basic examples:

- Good software modes: random access, pointer following (as long as we stay in cache)
- Good FPGA modes: streaming, systolic arrays, associative computing, fine-grained automata

Modes of Computation, cont.

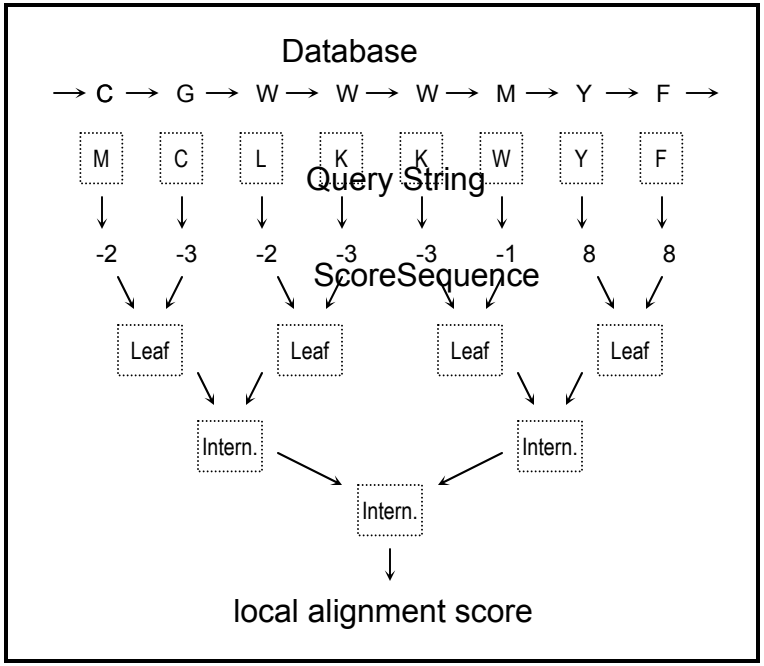
Example: BLAST

Serial solution: random access into database to extend seeds

FPGA solution: stream database through 2D systolic structure

Operation:

- Query string held in place, database streams over it
- On each cycle (alignment), one ScoreSequence generated
- ScoreSequences evaluated systolically by the tree structure



Relative cost of arithmetic

Scenario 11: Software division & multiplication have different relative costs versus FPGA division & multiplication

Example: FPGA division is painfully expensive, while multiplication is handled with hard-wired components

FPGA solution:

- Rewrite expressions to avoid division
- Use hard multipliers

HPC = HP data access[14]

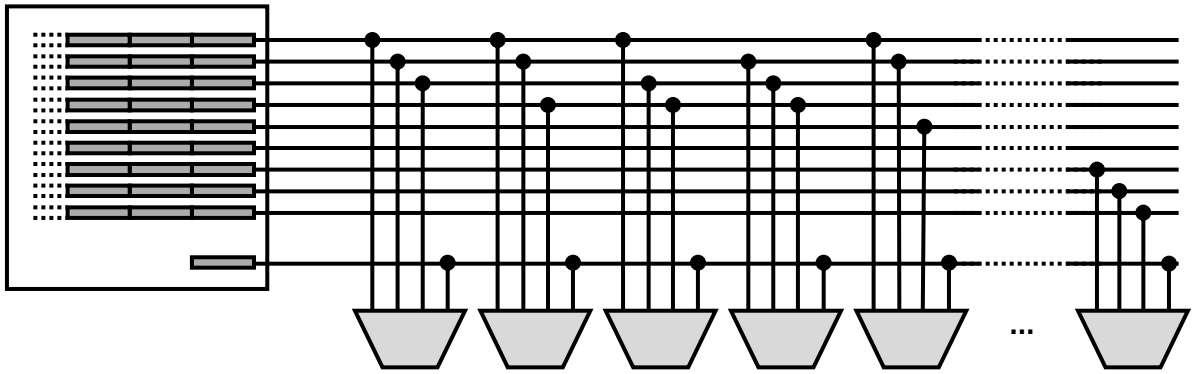
Scenario 12: dense, non-standard memory access pattern

Example: size-3 subsets of vectors

C Style:

```
for i = 0 to N
  for j = 0 to i
    for k = 0 to j
      // use x[i],x[j],x[k]
```

FPGA Solution:



Summary – *Scenarios handled with …*

EDA – language, synthesis, P&R:

- Applications with realistic (oo) parameterization
- Scaling to use resources
- Sizing for speed-matching
- Non-standard data types for FPGA-specific computation modes
- Generators for commonly used function types (memory reference)
- Function parallelism

Libraries

- Non-standard arithmetic
- "data" and computation structures

Programmer/Designer Training: FPGA-Awareness

- Algorithm selection, creation
- Arithmetic: rewriting expressions, choosing appropriate precision
- ***Use*** of libraries

Programmer/Designer: Logic-Awareness *… that's another talk!*

Work Referenced

- L. Snyder (1986): "Type Architectures, Shared Memory, and the Corollary of Modest Potential", Annual Review of Computer Science.
- Y. Gu, T. VanCourt and M.C. Herbordt (2006): "Accelerating Molecular Dynamics Simulations with Configurable Circuits," IEE Proc. Computers and Digital Technology, 153 (3). *(extended version of FPL 2005)*
- T. VanCourt, M.C. Herbordt (2006): "Families of FPGA Accelerators for Approximate String Matching," Microprocessors and Microsystems. *(extended version of ASAP 2004)*
- T. VanCourt, Y. Gu, V. Mundada, M.C. Herbordt (2006): "Rigid Molecule Docking: FPGA Reconfiguration for Alternative Force Laws," Journal on Applied Signal Processing. *(extended version of FPL 2004)*
- T. VanCourt, M.C. Herbordt, R.J. Barton (2004): "Microarray Data Analysis Using an FPGA-Based Coprocessor," Microprocessors and Microsystems 28 (4). *(extended version of FPL 2003)*
- Y. Gu, T. VanCourt, M.C. Herbordt (2006): "Improved Interpolation and System Integration for FPGA-Based Molecular Dynamics Simulations," FPL 2006.
- T. VanCourt, M.C. Herbordt (2006): "Application-Specific Memory Interleaving for FPGA-Based Grid Computations: A General Design Technique," FPL 2006.
- T. VanCourt, M.C. Herbordt (2006): "Sizing of Processing Arrays for FPGA-Based Computation," FPL 2006.
- M.C. Herbordt, J. Model, Y. Gu, B. Sukhwani, T. VanCourt (2006): "Single Pass, BLAST-Like, Approximate String Matching on FPGAs," FCCM 2006.
- T. VanCourt, M.C. Herbordt (2005): "Three Dimensional Template Correlation: Object Recognition in 3D Voxel Data," CAMP 2005.
- A. Conti, T. VanCourt, M.C. Herbordt (2004): "Processing Repetitive Structures with Mismatches at Streaming Rate," FPL 2004.
- T. VanCourt, M.C. Herbordt (2004): "Processor-Memory Networks Based on Steiner Systems," BARC 2004.

Questions?