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 DiSabella

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 $\rightarrow$ FPGA is much harder than Application $\rightarrow$ MPP

*B.C.B. with FPGAs*
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.
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\textsuperscript{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

*BCB with FPGAs*  
*RSSI 2006*
**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

\[
\begin{align*}
  \text{Express } (i,j,k) \text{ in } (x,y,z) \text{ basis} \\
  i=(x_i, y_i, z_i) \quad j=(x_j, y_j, z_j) \quad k=(x_k, y_k, z_k)
\end{align*}
\]
- 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 $\sim0$ 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\textsuperscript{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
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

**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
FPGA Hardware, cont.

C style: Sequential RAM access

FPGA Solution: App-specific interleaving
Scaling applications to FPGAs

**Scenario 5:** Scale application to maximal size given target hardware

- Hardware (invariably) varies

- Scaling depends on:
  - FPGA capacity
  - Application details
  - Computing array
Scaling Applications, cont.

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

\[ N_{opt} = \arg\max U(N)(2)^N | V(N) \land \{ \forall j : r_j^F \geq S_j(N, B) \} \]

**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\textsuperscript{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.\textsuperscript{8}

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

<table>
<thead>
<tr>
<th>Platform</th>
<th>Precision (bits)</th>
<th>Pipe-lines</th>
<th>HW multi’s used (% of usage)</th>
<th>Block RAMs used (% of usage)</th>
<th>Delay (ns)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP70 AMS</td>
<td>35</td>
<td>4</td>
<td>176(53%)</td>
<td>214(65%)</td>
<td>11.1</td>
<td>50.8×</td>
</tr>
<tr>
<td>VP70 AMS</td>
<td>40</td>
<td>4</td>
<td>264(80%)</td>
<td>251(77%)</td>
<td>12.2</td>
<td>46.4×</td>
</tr>
<tr>
<td>VP70 AMS</td>
<td>45</td>
<td>4</td>
<td>288(88%)</td>
<td>285(87%)</td>
<td>13.2</td>
<td>42.7×</td>
</tr>
<tr>
<td>VP70 AMS</td>
<td>51</td>
<td>4</td>
<td>288(88%)</td>
<td>317(97%)</td>
<td>18</td>
<td>31.3×</td>
</tr>
<tr>
<td>VP70 AMS</td>
<td>35</td>
<td>8</td>
<td>256(78%)</td>
<td>326(99%)</td>
<td>22.2</td>
<td>51.0×</td>
</tr>
<tr>
<td>VP100 sim</td>
<td>51</td>
<td>4</td>
<td>288(65%)</td>
<td>317(77%)</td>
<td>13.6</td>
<td>41.5×</td>
</tr>
<tr>
<td>VP100 sim</td>
<td>35</td>
<td>8</td>
<td>256(58%)</td>
<td>334(73%)</td>
<td>12.8</td>
<td>88.5×</td>
</tr>
</tbody>
</table>
Use Appropriate Arithmetic Mode

**Scenario 7:** integer or floating point not always optimal

Example 1: 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
Use Appropriate Arithmetic Mode, cont.

Find most significant 1 to:
- get format
- extract a
- extract (x-a)

Format

\[(x-a)\]

\[((C3*(x-a)+C2)*(x-a)+C1)*(x-a)+C0\]

Coefficient Memory

\[r^{14}, r^8, r^3\]

<table>
<thead>
<tr>
<th>Format</th>
<th>Add</th>
<th>Mul</th>
<th>Pipeline</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogiCore DP</td>
<td>692</td>
<td>540</td>
<td>n.a.</td>
<td>19566</td>
</tr>
<tr>
<td>LogiCore FP</td>
<td>320</td>
<td>120</td>
<td>n.a.</td>
<td>6908</td>
</tr>
<tr>
<td>Semi fp 35-bit</td>
<td>70</td>
<td>390</td>
<td>n.a.</td>
<td></td>
</tr>
<tr>
<td>Integer 35-bit</td>
<td>18</td>
<td>400</td>
<td>n.a.</td>
<td></td>
</tr>
<tr>
<td>Combined semi, int</td>
<td>n.a.</td>
<td>n.a.</td>
<td></td>
<td>5624</td>
</tr>
</tbody>
</table>
Applications (often) come in families, not point solutions\textsuperscript{10,11}

**Scenario 8:** Application has large number of complex variations
- Passes function as parameter

Example: Approximate string matching using dynamic programming

**FPGA Solution:** True object-oriented support

\textsuperscript{10. ASAP 2004  
11. JMM 2006}
good SW data structure $\neq$ good HW structure\textsuperscript{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
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\textsuperscript{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

\textsuperscript{13} FCCM 2006
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\textsuperscript{14}

**Scenario 12:** dense, non-standard memory access pattern

Example: size-3 subsets of vectors

C Style: 
\[
\text{for } i = 0 \text{ to } N \\
\quad \text{for } j = 0 \text{ to } i \\
\quad \quad \text{for } k = 0 \text{ to } j \\
\quad \quad \quad \text{// use } x[i],x[j],x[k]
\]

**FPGA Solution:**

![Diagram of FPGA solution]

\textsuperscript{14} BARC 2004
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


Questions?