FPGAs as Compute Engines

- **Proven successful**
  - 1000x speedup in bioinformatics, computational chemistry, etc.
  - Accelerator board fits standard PC backplane
  - Off-the-shelf availability, modest HW cost

- **What makes them work so well**
  - >400 memory busses, 3Tbit/sec* total bandwidth
  - Massive parallelism, fast on-chip communication

- **So why doesn’t everybody use them?**

* Xilinx XC2VP100
What is an FPGA?

A bag of uncommitted computer parts
No defined function – can be whatever you want

Why is programming a problem?

CPU runs the application, FPGA is the application
All of what’s hard in programming, only more so

How are we changing the model?

Separating circuit design from application logic
Addressing specific application areas
### CPU vs. FPGA Hardware

<table>
<thead>
<tr>
<th>Category</th>
<th>CPU has …</th>
<th>FPGA has …</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic &amp; logic</strong></td>
<td>1-10 pipelines</td>
<td>&gt; 400 HW multipliers</td>
</tr>
<tr>
<td></td>
<td>Fixed data width</td>
<td>~100K function cells</td>
</tr>
<tr>
<td><strong>Registers &amp; memory</strong></td>
<td>Fixed reg. array</td>
<td>~200K reg. bits</td>
</tr>
<tr>
<td></td>
<td>0-4 caches</td>
<td>&gt; 400 cache RAMs</td>
</tr>
<tr>
<td><strong>Connectivity &amp; communication</strong></td>
<td>Fixed datapath</td>
<td>Arbitrary data path</td>
</tr>
<tr>
<td></td>
<td>1 ext. data bus</td>
<td>&gt;1000 data I/O pins</td>
</tr>
<tr>
<td></td>
<td>1-32 dedicated I/O</td>
<td>20 links, 3-10Gbit</td>
</tr>
<tr>
<td><strong>Process technology</strong></td>
<td>Incremental growth</td>
<td>Exponential growth</td>
</tr>
<tr>
<td></td>
<td>Process limited</td>
<td>Process driver</td>
</tr>
</tbody>
</table>
Programming Skills vs. FPGAs

CPU model

- **Single-threading**
  - *No synchronization*
  - *for/if/switch control*
- **Incremental execution**
  - *One instruction at a time*
  - *Results are immediate*
- **Common parallelization**
  - *Large units of work*
  - *Costly communication*

FPGA model

- **Massive parallelism**
  - *Visible timing relations*
  - *State machine/hardwired*
- **Pipelined execution**
  - *All operations active*
  - *Visible dependencies*
- **Parallelism model**
  - *Fine grain – one ALU op*
  - *Cheap on-chip comm.*
Attempts to Date

- Hardware description languages
  *Unfamiliar control & resource models*
- Graphical design entry - tedious
  *Example: $X = 3*Y + 5*Z$*
- Standard programming languages
  *Good SW structure ≠ good HW structure*
- Semantic gap works both ways
  *Good HW designers aren’t application experts*
Requirements for a Solution

- Acknowledge SW and HW skills separately
  - *HW expertise: system interface, memory structure, synchronization, computation arrays*
  - *SW expertise: problem origination, data manipulation, algorithm variations, exploration*
  - Allow ‘normal’ representations to both
  - Eliminate dependencies between HW and SW

- Balance generality vs. domain specifics
  - Create multiple levels of generality
Behavior as a Parameter

- Reusable structure
  
  *Standard HW reuse:*  
  prefab leaf components + custom connectivity
  
  *Required HW reuse:*  
  prefab connectivity + custom leaves

- Go beyond VHDL parameterization
  
  *Not just data values as parameters*  
  Behavior as parameter
  
  *Like C library’s*  
  \texttt{qsort(data[], compare())}
Reusing Control, not Function

- **Example: Iterative Optimization**
- Logic designer provides:
  - Parameterized logic model
- End user provides:
  - $X_0$ – Initial candidate
  - $F_j(X)$ – Score next candidate solution $j$
  - $\text{Best}(S_0, S_1, \ldots)$ – Select solution[s] based on score[s]
- Fill-ins define the search algorithm
  - Hill climbing, Gibbs sampling, simulated annealing, ...
Familiar SW Development Style

- Standard design pattern*
  
  *Gamma et al., ‘Design Patterns’

- Event driven – ‘inverted’ flow of control
  
  Sequence and synchronization outside of app.
  System calls app-specific logic when needed
  Widely used for GUI, web applications

- Good match to object oriented design style
  
  System refers to abstract application interface
  Application provides concrete logic

*Gamma et al., ‘Design Patterns’
Preliminary Results

- Computational chemistry: 3D correlation
  - Systolic array for direct correlation
  - Speedup: 400x – 1000x relative to PC (using FFT)

- Microarray analysis
  - Regression analysis of disease vs. healthy state
  - Speedup: ~ 1000× relative to PC

- Approximate string matching
  - Dynamic programming – Smith-Waterman
  - 2.23 – $9.68 \times 10^9$ character comparisons/sec
Work in Progress

- XML representation for models
  - Define abstract application interface
  - Define HW in terms of abstract interface
  - Define abstract FPGA resources & model constraints

- Concrete representation of application logic
  - Create concrete application logic
  - Create concrete FPGA resource description
  - Bind concretions to abstract model

- Create synthesizable output
  - Repeatable elements scaled to actual FPGA resource limits