CS 250
VLSI System Design

Lecture 6 – Accelerator Projects

2013-9-17
Professor Jonathan Bachrach
slides by John Lazzaro

TA: Colin Schmidt

www-inst.eecs.berkeley.edu/~cs250/
Today’s lecture plan ...

* Power and energy. The techniques you’ll be able to use in your project.

* Pareto Optimality ... and how it impacts your project definition.

* Accelerator interface ... and its limits.

* Worked examples. Several project ideas, sketched out in detail.

* Starting points. Brief descriptions of projects ideas you may want to pursue.

---

Break
Power techniques available for project

- Parallelism and pipelining ✔
- Power-down idle transistors ✗
- Slow down non-critical paths ✔
- Clock gating ✔
- Data-dependent processing ✔
- Thermal management ✗
Cell libraries characterized at multiple $V_{dd}$ values.

So, you can pick a different $V_{dd}$ value for each of your implementations.

Several $V_{dd}$ values in one implementation not supported.

The main trick:

Top block processes “left”, bottom “right”.

$P \sim \#\text{blks} \times F \times V_{dd}^2$

$P \sim 2 \times 1/2 \times 1/4 = 1/4$

$CV^2$ power only
Not by varying $V_{dd}$, but by cell choice

The critical path

Most wires have hundreds of picoseconds to spare.

\[(H,S,L) == \text{High Vt, Standard Vt, Low Vt}\]

\[V_{dd}\ is\ fixed,\ so\ cell\ choice\ only\ helps\ leakage\ power\ \ldots\]

\[(40, 45, 50)\ are\ channel\ lengths\ (in\ \text{nm})\]
Evolutionary Trade-Offs, Pareto Optimality, and the Geometry of Phenotype Space

**Pareto Optimality**

**A** Darwin’s ground finches

- Archetypes:
  1. Long beak, medium body
  2. Large thick beak, large body
  3. Small thick beak, small body

- Task (Diet):
  1. Insects, nectar
  2. Large, hard seeds
  3. Small, soft seeds

**B** Leaf-cutter ants (A. sexdens)

- Archetypes:
  1. Small head width (HW), small poison sac / pronotal width (PS/PW)
  2. Medium HW, large PS/PW
  3. Large HW, small PS/PW

- Task:
  1. Gardening / nursing
  2. Foraging
  3. Soldiering

**C** Bats (Microchiroptera)

- Archetypes:
  1. Low aspect ratio, small body
  2. High aspect ratio, medium body
  3. Low aspect ratio, large body

- Task:
  1. Prey: small insects, near vegetation
  2. Prey: high flying large insects
  3. Prey: animals, near vegetation

---

**Evolutionary Trade-Offs, Pareto Optimality, and the Geometry of Phenotype Space**


The data reported in this paper are from a study of ground finches, leaf-cutter ants, and bats, illustrating Pareto optimality and the geometry of phenotype space.
Allometric relations often behave as power laws, because all m. Omnivore molars are weighted averages of these ends of the observed line segment: a herbi-

archetype, the more important that task (lower fitness, and will be selected against. The projection on the line 

is maximized by a single phenotype. The pheno-

nature fill only a small fraction of morphospace.

We next calculate the Pareto front in morpho-


The case of a trade-off between two tasks 

predicted is a function of the log of the traits (mathematically, \( \text{distance} \), we mean a metric based on an inner product norm, such as Euclidean dis-

variation in several classic studies of animal mor-

Each vertex. The species near the archetypes sug-

For three tasks, the Pareto front is the full triangle 

whose vertices are the three archetypes. In this 

Pareto front: the line segment connecting the two archetypes (unlike (A), axes are traits, not perform-

species described in the caption (Fig. 1C). Spe-

however, half of the species fall within a triangle 

(Grant and colleagues on Darwin’s finches, Figs. 3A, inset). A triangle is found in the study of Grant 

triangle. The species near the vertices of the tri-

— 

that are dominated on all tasks by other feasible phenotypes 

the area proportions of the molar teeth of 29 ro-

sinism relative contributions of each task to the orga-

performance in each task. Which of the pheno-

throw away”

Clock Frequency

Energy

\( \text{Task 2 performance} \)

\( \text{Task 1 performance} \)

\( \text{Pareto Front} \)

\( \text{Vdd} \)

% of Low-Vt Cells

Hi Speed

archetype 2

Lo Power

archetype 1

\( v \)

\( v' \)

\( d_1 \)

\( d_1' \)

\( d_2 \)

\( d_2' \)

\( v \)

\( v' \)

Trajectories in trait space are linear in biology ... but in chip design? Open question.
In multiple dimensions

“Tasks”: Power, Speed, Area, Yield

“Traits”: $V_{dd}$ choice, cell $V_t$ choice, choice of SRAM vs. synthesized state, etc.
Rocket Facts
Blue “1” bits yield register queuing shown on diagram ...
the 7-bit funct7 field is accelerator’s 128 opcodes.

<table>
<thead>
<tr>
<th>funct7</th>
<th>rs2</th>
<th>rs1</th>
<th>xd</th>
<th>xsl</th>
<th>xs2</th>
<th>rd</th>
<th>dest</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

All 32 bits show up here ...
How to manage private 32-entry “A-File” register bank without using up opcode bits ...

MOV: A-file to Regfile

MOV: Regfile to A-File
D-Cache Facts

CMD: Load, Store

MTYPE: 8, 16, 32, 64 bits.

MADDR: Align to MTYPE

TAG: 9-bits. Lets loads be OoO, up to 4 missed loads.

Performance: 4 cycle latency on a cache hit, 40-60 on a miss. No prefetching built in ...
Click Prediction Acceleration
Advertisers pay Google $1.47, on average, if Google Search displays their ad in response to the search term "mt fuji vacation."
Since Google is only paid if the user clicks, they predict, in real time, which of the bidding ads is most likely to yield a click.

Hawaii ad penalized.
Basic idea: Billions of “features” are developed to predict, given an ad and a search, how likely it is that the searcher will click on the ad.

\[ a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \]

**a vector**: feature values for the search.

**b vector**: feature values for the ad.

Example 50 features: Does geo info indicate that the searcher is in the state of (1) Alabama? (2) Alaska .... (50) Wyoming. Binary, sparse features.
To rank each ad: Take the dot product of $a$ and $b$ for each ad, give the highest-valued ads placement.

$$a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

“$n$” here is in the billions, but non-zero “$a$” and “$b$” values are in the thousands. This real-time system needs to exploit the sparsity to perform well.

A good candidate problem for an accelerator.
Assumptions

```
a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n
```

"a" and "b" are binary. The click prediction system uses fixed-point "b" and integer "a". Binary vectors let us focus on deeper issues.

Sorted 32-bit index list, zero-terminated.

- **a**: 1, 13, 1827, 2000, 1938475, 0 (24 bytes)
- **b**: 12, 13, 2000, 1602938, 0 (20 bytes)

Dot product is **2**: (13 and 2000 match)
One Instruction

SBDT dest_reg, a_reg, b_reg

a_reg: Holds 64-bit memory address pointing to the first byte of “a” list.

b_reg: Holds 64-bit memory address pointing to the first byte of “b” list.

dest_reg: 32-bit unsigned int # of list elements.

saturating 16-bit unsigned ints.

\[ a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \]
Simple implementation

- Increment `a count` and `b count`.
- Increment `dot count`.
- Check if `most recent a` equals `most recent b`.
- If equal, request the next element from memory.
- Otherwise, keep the current memory controller.

Request to memory for the next element of the "lagging" list.
Improving performance

Do more compares/clock.

Tactics:

Memory controller should strive to reach one 64-bit load per clock (max possible).

Replace most-recent registers by list structures. Comparator? Parallelize.
Beyond one 64-bit load per clock

Delta data compression

Turn \( a: 200, 300, 301, 400, 402, 0 \)

into \( a: 200, 100, 1, 99, 2, 0 \)

Store with MIDI compression scheme:

Coding 0–127:

Encoded form: 0ddddddd
Decoded form: 00000000 00000000 00000000 0ddddddd

Coding 128–16383:

Encoded form: 1ccccccc 0ddddddd
Decoded form: 00000000 00000000 00cccccc cddddddd

For “\( a \)” list above, \( 3.4 \times \) memory bandwidth gain.
Improving performance

Do fewer compares/instruction.

Observations:
To compute the correct dot product, we only need to load list elements that match.

Because lists are sorted, a speculative memory controller can “skip” many loads.

Works best if RISC-V passes in list lengths.
Project variants

Let “b” vector be sparse fixed-point.

Unsorted 32-bit index list, zero-terminated. Changes many aspects of the problem.

Change sparse vector representation to be binary bit streams stored in memory:

a: 00000000100000000001000000

Accelerator defines a compression method that differs from “indices list”.

Break
Cramer’s Rule

Solve this matrix equation:

With determinants:

To solve for $x_i$:
Substitute $b$ for column $i$ in the numerator ...

$$x_i = \frac{\det(A_i)}{\det(A)} \quad i = 1, \ldots, n$$
Recent work shows how to make Cramer’s rule scale and be stable for large systems:

So, let’s make an accelerator based on it …
Determinant works on a matrix, but returns a scalar. We use accelerator instructions to compute determinants, and let RISC-V compute the x vector by doing the divides.

\[ x_i = \frac{\text{det}(A_i)}{\text{det}(A)} \quad \text{for} \quad i = 1, \ldots, n \]

Determinants of matrices with integer coefficients can be computed exactly, with only integer multiplies and adds. So, we restrict our accelerator accordingly.
Chió's Trick

For \( n = 3 \), computes determinant of a \( 3 \times 3 \) matrix by computing the \( 2 \times 2 \) determinant of four \( 2 \times 2 \) determinant results.

We can reuse the "leaf" 2x2 determinants when we compute the full set of \( \det(A_i) \) and \( \det(A) \).

\[
x_i = \frac{\det(A_i)}{\det(A)} \quad i = 1, \ldots, n
\]
Implicit architected state for the accelerator can be used to store and reuse partial results ...

\[
\begin{vmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{12} & a_{22} & a_{23} \\
    a_{21} & a_{32} & a_{33}
\end{vmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}
\]

\[
\det A = \begin{vmatrix}
    (a_{11} a_{22} - a_{12} a_{21}) & (a_{11} a_{23} - a_{13} a_{21}) \\
    (a_{11} a_{32} - a_{12} a_{31}) & (a_{11} a_{33} - a_{13} a_{31})
\end{vmatrix}
\]

\[
\det A_2 = \begin{vmatrix}
    (a_{11} b_2 - b_1 a_{21}) & (a_{11} a_{23} - a_{13} a_{21}) \\
    (a_{11} b_3 - b_1 a_{31}) & (a_{11} a_{33} - a_{13} a_{31})
\end{vmatrix}
\]

\[
\det A_3 = \begin{vmatrix}
    (a_{11} a_{22} - a_{12} a_{21}) & (a_{11} b_2 - b_1 a_{21}) \\
    (a_{11} a_{32} - a_{12} a_{31}) & (a_{11} b_3 - b_1 a_{31})
\end{vmatrix}
\]
Two instructions

**DETA** dest_reg, a_reg, len_reg

- **a_reg**: 64-bit memory address of $A$ matrix.
- **len_reg**: Holds the $n$ of the $n \times n$ $A$ matrix.
- **dest_reg**: Return register for det($A$).
  
  Retains $n$ and partial results for $A$.

**DETAI** dest_reg, b_reg, col_reg

- **b_reg**: 64-bit memory address of $b$ vector.
- **col_reg**: The “$i$” (column) for det($A_i$)
- **dest_reg**: Return register for det($A_i$).
  
  Adds to partial results (for $A_i$).

First, it clears all implicit state
First Pentium with SIMD instructions. Released in 1996. Commemorative keychain, using a die that failed test.
The “right” complexity for your project ...

PCMPESTRM — Packed Compare Explicit Length Strings, Return Mask

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>66 OF 3A 60 /r imm8</td>
<td>RMI</td>
<td>V/V</td>
<td>SSE4_2</td>
<td>Perform a packed comparison of string data with explicit lengths, generating a mask, and storing the result in XMM0.</td>
</tr>
<tr>
<td>PCMPESTRM xmm1, xmm2/m128, imm8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEX.128.66.0F3A.WIG 60 / r ib</td>
<td>RMI</td>
<td>V/V</td>
<td>AVX</td>
<td>Perform a packed comparison of string data with explicit lengths, generating a mask, and storing the result in XMM0.</td>
</tr>
<tr>
<td>VPCMPPESTRM xmm1, xmm2/m128, imm8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PCMPISTRI — Packed Compare Implicit Length Strings, Return Index

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>66 OF 3A 63 /r imm8</td>
<td>RM</td>
<td>V/V</td>
<td>SSE4_2</td>
<td>Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.</td>
</tr>
<tr>
<td>PCMPISTRI xmm1, xmm2/m128, imm8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEX.128.66.0F3A.WIG 63 / r ib</td>
<td>RM</td>
<td>V/V</td>
<td>AVX</td>
<td>Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.</td>
</tr>
<tr>
<td>VPCMPISTRI xmm1, xmm2/m128, imm8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use m64 instead of m128, to match RISC-V register size.
By using an existing ISA spec, you can focus on Pareto tradeoffs and micro-architecture.
GARP'd

You could also take Intel idea as a starting point for a high level redesign ...

Garp: A MIPS Processor with a Reconfigurable Coprocessor

John R. Hauser and John Wawrzynek
University of California, Berkeley
Quick Idea #1

MEMS microphone post-processing accelerator

Seamlessly Interfacing MEMS Microphones with Blackfin® Processors

The microphones are clocked at a rate of 350 kHz. To connect two such microphones, the mute of the microphones have to be clocked at half the rate of the clock at which the rate of the clock at which the clock input to the microphone and the SPORT receives the data at 3 MHz. This is followed by two 2:1 FIR half band filters with 300 taps divided by 2:1 decimator.

The data coming out of the microphone after feeding it with the sine tone and noise generated in the sigma delta modulation process in the sigma delta modulation process is sent to the decimation process. The figures are taken from Figure 4. The microphones are connected to the Blackfin processor and the MEMS microphone interface between the microphones and the Blackfin processor is done by implementing the code associated with the interface. The reconstructed audio is sent to a DAC for audio output purpose.

Note

The figures provide a visual representation of the setup and the processing sequence. The data coming out of the microphone is shown in Figure 5, demonstrating the output of the CIC decimation stage.

Table

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>FIR Low Pass Filter</th>
<th>2:1 HB Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tap</td>
<td>36</td>
<td>560</td>
</tr>
</tbody>
</table>

The experimental evaluation platform has been selected to illustrate the interfacing of MEMS microphones to Blackfin® processors. Any of the existing Blackfin evaluation boards can be used for this purpose. For more information on the implementation, refer to the Blackfin processor documentation.
A 280 mV-to-1.1 V 256b Reconfigurable SIMD Vector Permutation Engine With 2-Dimensional Shuffle in 22 nm Tri-Gate CMOS

Steven K. Hsu, Member, IEEE, Amit Agarwal, Member, IEEE, Mark A. Anders, Member, IEEE,

Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached

Kevin Lim
HP Labs

David Meisner
Facebook

Ali G. Saidi
ARM R&D

Systolic Sorting on a Mesh-Connected Network

HANS-WERNER LANG, MANFRED SCHIMMLER, HARTMUT SCHMECK, AND HEIKO SCHRÖDER

FINITE AUTOMATA BASED COMPRESSION OF BI-LEVEL AND SIMPLE COLOR IMAGES

KAREL CULIK II* and VLADIMIR VALENTA

Department of Computer Science, University of South Carolina, Columbia, SC 29208, U.S.A.

e-mail: culik@cs.sc.edu

Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing

Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, Mark A. Horowitz