An Overview of FLEET

CS-152
FLEET

- Brainchild of Ivan Sutherland
  - Fleshed out in collaboration with Berkeley graduate students
- A one-instruction, clockless processor
  - Alternatively: an asynchronous transport-triggered VLIW machine
- Designed to take advantage of asynchronous circuits
- But first, I’d like to talk about...
- Ancient bridges were made of stone
- Required arch-and-keystone design
Steel

- Much stronger building material than stone
Early Steel Bridges

- Arch-and-keystone not required
- Regardless, early steel bridges were built as imitations of stone bridges.
Modern Steel Bridges

- Steel made suspension bridges possible
- Took a while for people to realize that new materials enabled new designs
What does this have to do with Computer Architecture?

- Synchronous circuits
  - Single-cycle, pipelined, superscalar, Tomasulo, etc

- Asynchronous circuits
  - Early async processors imitated synchronous architectures
  - FLEET is a first draft of what “suspension bridges” might look like
Crash Course on Asynchronous Circuits
KLA

- Kinetic Learning Activity
IF \( \text{predecessor} \neq \text{successor} \)
THEN
\begin{verbatim}
copy \text{predecessor}
\end{verbatim}
Muller C Element

- Majority gate with output looped back
- A two-voter election with “incumbent advantage” in event of a tie
Micropipelines

- Chain of Muller C-Elements
- Each element connects to an inverted input on predecessor, non-inverted on successor
Micropipelines

“each stage of the control...follows a very simple stage state rule:

\[
\text{IF } \text{predecessor } \neq \text{successor} \\
\text{THEN } \\
\quad \text{copy } \text{predecessor’s state} \\
\text{ELSE} \\
\quad \text{hold } \text{present state}
\]

[IES’89]
Micropipelines

IF \( \text{predecessor} \neq \text{successor} \)
THEN
\( \text{copy \ predecessors'} \text{\ state} \)
ELSE
\( \text{hold \ present \ state} \)
Micropipelines

IF \texttt{predecessor} \neq \texttt{successor} THEN
\hspace{1em}\texttt{copy \ predecessor's state}
ELSE
\hspace{1em}\texttt{hold \ present \ state}

- Any “disagreements” propagate to the right
IF $\textit{predecessor} \neq \textit{successor}$
    THEN
        copy $\textit{predecessor}$’s state
    ELSE
        hold $\textit{present}$ state

• Any “disagreements” propagate to the right
Micropipelines

IF predecessor != successor
THEN
  copy predecessor’s state
ELSE
  hold present state

- Any “disagreements” propagate to the right
Micropipelines

IF `predecessor` != `successor`
  THEN
    copy `predecessor`'s state
  ELSE
    hold present state

- Any “disagreements” propagate to the right
Micropipelines

IF predecessor != successor
    THEN
        copy predecessor's state
    ELSE
        hold present state

• Any “disagreements” propagate to the right
• System is stable when pipeline contains
  • Zero or more agreeing stages
  • Followed by zero or more disagreeing stages
Micropipelines

- Transition signaling

- A transition on a stage’s output will:
  - Acknowledge data from the previous segment
  - Signal data ready to next segment
Other Styles Exist

- Too many for this talk
Advantages of Async

- Average (not worst) case timing -- no "timing closure"
- Early completion with some circuit styles
- Separates correctness from performance
- No clock tree
- Low power
- Better modularity
- Better EMI profile
- Extremely robust to variations Process, Temperature, Voltage
Caltech MiniMIPS

Designed 1995-1998

250% the performance of the best synchronous MIPS on the same fabrication process

Performance scaled “automatically” with voltage and temperature changes
Communication Matters

- “Metal rod” model of VLSI wires
- Communication is what matters, so put the programmer in charge of it
- One instruction: MOVE
Mike Holenderski’s Animation

http://research.cs.berkeley.edu/class/fleet/docs/fleet_animation.swf
MIPS Pipeline vs FLEET SHIPs
Inboxes and Outboxes

As an amendment to earlier memo AM-/, formerly "bar" token inputs and outputs now have inboxes and outboxes.

In the diagram below, dashed lines indicate connections which always carry tokens, while solid horizontal lines carry data in the case of data in; boxes "outboxes" and tokens in the case of token inboxes "outboxes."
Code “Bags”

- Unordered sets of instructions
- Lets us start fetching the next “block” of code as soon as the current “block” starts executing
- Unlike sequential-instruction ISAs
Anatomy of a FLEET Instruction

Recall that if a codebag contains two or more instructions which have the same source address, those instructions are guaranteed to be executed at their source ship in the order in which they appear in the codebag.

This property is important for understanding the usefulness of some instruction forms. In particular, many instruction forms are only useful as a "barrier" in the stream of instructions being sent to a particular ship.

An instruction tells a box what action to take. This action involves possibly copying or consuming a datum from its input and possibly sending that datum on its output. The instruction may also specify whether or not the box must wait for a token on its input before performing this action, and whether it should emit a token on its output to announce that the action has been completed.

The field - formerly (can now hold only values and . A is no longer valid; its role has been subsumed by the other instruction fields.

A conceptual Fleet instruction format is shown below.

<table>
<thead>
<tr>
<th>Source Address</th>
<th>DataIn Ignore/Copy/Take</th>
<th>Triggered</th>
<th>Count</th>
<th>Ack</th>
<th>DataOut</th>
<th>Destination Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>1</td>
<td>log₂ MAXMOVE</td>
<td>1</td>
<td>1</td>
<td>log₂ NUMDESTINATIONS</td>
<td></td>
</tr>
</tbody>
</table>

Anatomy of a FLEET Instruction
## Instruction Forms

<table>
<thead>
<tr>
<th>Triggered</th>
<th>Not Triggered</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataIn</td>
<td>DataIn</td>
</tr>
<tr>
<td>Ignore</td>
<td>Copy</td>
</tr>
<tr>
<td>nop</td>
<td>wait</td>
</tr>
<tr>
<td>Ack</td>
<td>nop+ack</td>
</tr>
<tr>
<td>DataOut</td>
<td>copy</td>
</tr>
<tr>
<td>move (or accept)</td>
<td></td>
</tr>
<tr>
<td>DataOut+Ack</td>
<td>copy+ack</td>
</tr>
</tbody>
</table>
Synchronous SHIPs

- The fact that the switch fabric is asynchronous is fairly essential to FLEET
- However, SHIPs can be internally synchronous
- Pauseable clocks are an attractive option
More Information

- Website:
  - http://research.cs.berkeley.edu/class/fleet/
- Graduate seminar meets in this room tomorrow, 10am-noon
- Project presentations are a week from tomorrow