VOLK and GNU Radio

Tom Rondeau
GNU Radio maintainer

Boston, MA
October 1, 2011
What’s the problem we are trying to solve?

VOLK: Vector-Optimized Library of Kernels

Better than he was before. Better... stronger... faster.
Single Instruction, Multiply Data (SIMD) Basics

```
for i in [0 : N] {
    x_i * y_i = z_i
}
```

Traditional (scalar) math. Only one multiply.
Single Instruction, Multiply Data (SIMD) Basics

Vectorized math. One instruction does four multiplies

\[
\text{for } i \text{ in } [0 : 4 : N] \quad \left\{ \begin{array}{c}
x_i \times x_{i+1} \times x_{i+2} \times x_{i+3} \\
y_i \times y_{i+1} \times y_{i+2} \times y_{i+3} \\
\end{array} \right. = \begin{array}{c} z_i \\ z_{i+1} \\ z_{i+2} \\ z_{i+3} \end{array}
\]
### SIMD Registers in x86 chips

Holds doubles, floats, ints, shorts, and chars

<table>
<thead>
<tr>
<th>128-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
Other SIMD architectures

• Intel: SIMD (SSE, AVX)
  – AVX extends to 256-bit registers
• PowerPC: AltiVec
• AMD: 3DNow!
• ARM: NEON
• Others, but mostly on dead architectures
VOLK: Set of architecture-specific kernels

- Some Math Function
- Some Math Function
- Some Math Function
- Some Math Function
- Some Math Function

- Architecture 1
- Architecture 2
- Architecture 3
- Architecture 4
- Architecture 5

- generic
Runtime engine finds best architecture for the processor and selects it.
If no suitable architecture kernel has been written, fall back on the generic kernel.
Naming Convention:
http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk
GNU Radio Implementation Issues
Memory Alignment

• SIMD instructions (generally) want to have some byte alignment
  – SSE: 16-byte aligned loads
  – AVX: 32-byte aligned loads

• Loading unaligned data can cause a seg fault.

• Using special unaligned load instructions is very time consuming
  – Aligned memory in an unaligned load is not guaranteed to be promoted
GNU Radio Implementation Issues
Memory Alignment

Initially page aligned

work() function given a start pointer

What’s the alignment?
GNU Radio Implementation Issues
Memory Alignment

Initially page aligned

Use `set_output_multiple(x)` to ensure alignment

Always properly aligned

Given an output multiple commensurate with the alignment and data type, we can keep alignment.