A programmable baseband processor design for software defined radios

Sridhar Rajagopal
Scott Rixner and Joseph R. Cavallaro

This work is supported by Nokia, TI, TATP and NSF
Motivation

Flexibility needed in future high data rate systems

Mobile
- Switching between different protocols
- Switching between different parameters

Base-station also

Software Defined Radio (SDR)
Real-time Computation Requirements

Estimation, Detection and Decoding in a 4Mbps W-CDMA cellular multiuser system

- **FAST FADING** (estimation every 10 bits)
- **MEDIUM FADING** (estimation every 100 bits)
- **SLOW FADING** (estimation every 1000 bits)

ALUs required for real-time at 500 MHz

Number of W-CDMA Cellular Users

Data rates per user:

- **Add**
- **Multiply**
Real-time design challenge for SDRs

- At least 20 multiplications and additions required per clock cycle for 32-user system
- Current generation single processor DSPs lack that required horsepower
- Real-time implementations can be achieved in VLSI

What is needed to make it programmable?
Our contributions

• Target a real-time communications processor design using a streaming media processor simulator
• Target a processor design with high functional unit utilization with minimum memory stalls
• Study sophisticated baseband algorithms on a streaming processor
Streaming processor simulator

- GPP simulator not the right one
- Media processor/DSP simulators more relevant
- Explore space of stream processor architectures with isim
  - stream processor simulator from MIT/Stanford
  - Cycle-accurate simulation
  - Flexible machine description language (read by both simulator and compiler)
  - Vary number and design of functional units
  - Vary memory, register sizes
  - etc.
Imagine architecture – Base example
Stream Programming

- **StreamC**
  - Executes on host processor
  - C++
  - Controls stream transfers between main memory and SRF

```cpp
void main() {
    Stream<int> a(256);
    Stream<int> b(256);
    Stream<int> c(256);
    Stream<int> d(1024);
    ...
    example1(a, b, c);
    example2(c, d);
    ...
}
```

- **KernelC**
  - Executes on clusters
  - C-like Syntax
  - Kernel computation
  - Compiled by iscd

```cpp
KERNEL example1(
    istream<int> a,
    istream<int> b,
    ostream<int> c)
{
    loop_stream(a) {
        int ai, bi, ci;
        a >> ai;
        b >> bi;
        ci = ai * 2 + bi * 3;
        c << ci;
    }
}
```
Algorithms

• Multiuser estimation
  – replaced matrix inversion by gradient descent

• Multiuser detection
  – Parallel Interference Cancellation (PIC)
  – Pipelined algorithm that avoids block-based detection

• Viterbi decoding
  – Trellis structures suited for decoding
  – Register exchange for survivor memory
  – No traceback latency
Stream Dataflow

- Estimation bits
  - Correlation update kernel
  - Matrix mult kernel
  - Iteration update kernel
  - Computation

- Data rearrangement
  - Detection bits
  - Matched filter kernel
  - Matrix mult kernel
  - Matrix transpose
  - PIC kernel
  - Communication

- Buffer
  - Viterbi kernel
  - Decoding

Multiuser
Channel Estimation

Multiuser Detection

Decoding
**Base implementation**

- 32 cycle loop
- Executed on all 8 clusters
- Complexity
  - $O(N^3)$ multiplies
  - $O(N^3)$ adds
- 100% multiplier utilization in the loop
- Divider is unnecessary!

```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>ADD0</th>
<th>ADD1</th>
<th>ADD2</th>
<th>MUL0</th>
<th>MUL1</th>
<th>DIV0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Communication**
- (waiting for input)
- (input ready but FU busy)

**Inner Loop**
Replacing divider with multiplier

- 22 cycle loop
- Executed on all 8 clusters
- 97% multiplier utilization in the loop
- 85% adder utilization in the loop

- Changing functional units
  - Supported by simulator/compiler
  - Architecturally realistic
## Kernel Computation Time

<table>
<thead>
<tr>
<th>Algorithm Kernel</th>
<th>Functional unit utilization (3 +, 2 *)</th>
<th>Execution Time (cycles)</th>
<th>Functional unit utilization (3 +, 3 *)</th>
<th>Execution Time (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70%, 100%</td>
<td>1224</td>
<td>78.6%, 78%</td>
<td>1064</td>
</tr>
<tr>
<td>2</td>
<td>53%, 91%</td>
<td>22720</td>
<td>85%, 99%</td>
<td>14360</td>
</tr>
<tr>
<td>3</td>
<td>55%, 42%</td>
<td>1058</td>
<td>55%, 28%</td>
<td>1058</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>14464</strong></td>
</tr>
<tr>
<td>Glue</td>
<td>59%, 91%</td>
<td>7468</td>
<td>78%, 84%</td>
<td>5573</td>
</tr>
<tr>
<td>Matrices</td>
<td>63%, 96%</td>
<td>12192</td>
<td>68%, 71%</td>
<td>11084</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>16657</strong></td>
</tr>
<tr>
<td>Detect</td>
<td>67%, 100%</td>
<td>366</td>
<td>90%, 90%</td>
<td>275</td>
</tr>
<tr>
<td>7</td>
<td>67%, 96%</td>
<td>996</td>
<td>89%, 84%</td>
<td>760</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>1035</strong></td>
</tr>
<tr>
<td>Decode</td>
<td>13%, 2%</td>
<td>8044</td>
<td>13%, 1.4%</td>
<td>8044</td>
</tr>
</tbody>
</table>

Opt. for Computations

Opt. for min. communication

Time available at 128 Kbps for each of 32 users at 500 MHz: 4000 cycles
Estimation – Detection execution

Cycle

Kernels

Memory transfers

High FU util. but comm. overhead

Stalls waiting for data from memory

Estimation

Detection (10 bits)
Real-time Performance

Initialization

Decode (32 bits)

Cycle | Kernels | Memory transfers

<table>
<thead>
<tr>
<th>MC_D</th>
<th>MEM_D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>6000</td>
<td></td>
</tr>
<tr>
<td>7000</td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td></td>
</tr>
<tr>
<td>9000</td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>11000</td>
<td></td>
</tr>
<tr>
<td>12000</td>
<td></td>
</tr>
<tr>
<td>13000</td>
<td></td>
</tr>
<tr>
<td>14000</td>
<td></td>
</tr>
<tr>
<td>15000</td>
<td></td>
</tr>
<tr>
<td>16000</td>
<td></td>
</tr>
<tr>
<td>17000</td>
<td></td>
</tr>
<tr>
<td>18000</td>
<td></td>
</tr>
</tbody>
</table>

Low comm. overhead but low FU util.

Rate 1/2
Constraint Length 5
Real-Time Performance

Real-Time at 500 MHz
DSP comparisons

![Graph showing execution time for different kernels and memories.]
Future Work

• Achieve real-time rates
  – Additional functional units (that can be used efficiently!)
  – Eliminate communication stalls between kernels
  – Support for matrix transposes and bit-level operations

• Power and area constraints
  – Low power stream processing
  – Scaling the architecture for handsets

• Scalability with data rates
  – Boundaries of the architecture

• Handset algorithms
Conclusions

• Future wireless communications algorithms
  - exceed the capabilities of current DSPs
  - require flexibility to change algorithms and parameters
  - require efficient use of resources because of delay, area, and power constraints

• Architectural developments are needed for future SDRs
  - Stream processing is a promising approach
  - Additional hardware acceleration, akin to Viterbi coprocessor on C64?