Computer Generation of Platform-Adapted Physical Layer Software

Yevgen Voronenko (SpiralGen, CMU)
Christian Berger (CMU)
Volodymyr Arbatov (CMU)
Ronghui Peng (SpiralGen)
Franz Franchetti (CMU)
Markus Püschel (CMU)
SpiralGen at a Glance

- **Software & Service**
  For computationally intensive apps
  Performance optimization service | Optimized components | Performance evaluation

- **Value proposition**
  - *Faster time-to-market*
  - *Quick customization*
  - *Platform independence w/ no penalty*

- **Target applications:**
  - Core signal/image/video processing
  - Medical Imaging

- **Target platforms:**
  - Multi-core DSPs / GPPs
The Future is Parallel

- General purpose/high performance platforms
  - Intel Core i7 (4-8 cores, 128-B SIMD)
  - Intel Larrabee (16, 512-B)
  - ARM Cortex (1-4, -)
  - IBM Cell (9, 128-B)
  - Nvidia G200 (240)

- DSP software platforms
  - Tilera Tile64 (64, 3-way)
  - Sandbridge SB3500 (4, 128-B)
  - Freescale MSC8156 (6, ?)
  - ClearSpeed
  - StreamProcessors


Can waveform porting be substantially accelerated?
Rethinking the Software Radio

- Trend for performance-driven apps?

- Key driver: reducing the chip count on board

- Advantages:
  - Reduced software partitioning effort + easier porting
  - Lower power + longer battery life

- Magic bullet: single-chip multi-layer SDR solution

- Example: DSP/GPP/FPGA board – 25W, Intel board – 30W

*Multi-core GPPs are real. But can compilers retarget the code?*
The Answer is No

DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)
Performance [Gflop/s]

Multiple threads: 3x
Vector instructions: 3x
Memory hierarchy: 5x

Only guru programmers can do the job
WiFi Transmitter Example

WiFi transmitter on Dualcore Intel Atom
Run time per OFDM symbol [micro seconds] vs. Data rate [Mbit/s]

- Straightforward C code but minimizing op count
- Best standard C code
- Computer generated

Parallelism: 2 threads, 4-16 way SIMD

Compilers fail to optimize: 50x
SpiralGen’s Core Technology: Automation Tools

Application specification
Algorithm knowledge

Platform description
Instruction set architecture

_\text{mm\_set1\_epi8}(x) = \ldots
_\text{mm\_xor\_si128}(x,y) = \ldots
_\text{mm\_avg\_epu8}(x,y) = \ldots
_\text{mm\_cmpeq\_epi8}(x,y) = \ldots
_\text{mm\_unpacklo\_epi8}(x,y) = \ldots
\ldots

Spiral frees up guru programmers
Proof-of-Concept: Spiral and Intel IPP 6.0

Spiral-generated code in IPP (since 6.0, fall 2008):
- 1M lines of code
- 3984 C functions, 1203 functions are user visible
- Functionality: Cross product of
  Transforms: DFT (fwd+inv), RDFT (fwd+inv), DCT2, DCT3, DCT4, DHT, WHT
Sizes: 2–64 (DFT, RDFT, DHT); 2-powers (DCTs, WHT)
Precision: single, double
Data type: scalar, SSE, AVX (DFT, DCT), LRB (DFT)

Written by a computer

Can we do the same for SDR?
DFT on Intel Multicore

Complex DFT (Intel Core i7, 2.66 GHz, 4 cores)
Performance [Gflop/s] vs. input size

Vectorized, threaded, general-size, adaptive library
Spiral Outperforms Human Programmers

DCT on 2.66 GHz Core2 (single-precision, 4-way SSSE3) performance [Gflop/s], opcount = 2.5 n ld n, Windows XP 32-bit, Intel C++ 10.1

Spiral generated

5–6x

Intel, FFTW

GEMM (acc. packed sq. NN), single-threaded, single precision
Performance [Gflop/s] Dual Intel Xeon 5160, 3000 MHz, icc 10.1

MKL 10.0

Spiral-generated library

GotoBLAS 1.26

DGEMM

SAR Image Formation on Intel platforms performance [Gflop/s]

Performance Gain of Various Generated Viterbi
Speedup over Karn's C implementation
How Can Machine-Generated Code Outperform Hand-Optimized?

- **Spiral** explores both:
  - Algorithms
  - Code restructurings

![Graph showing performance vs. man-hours for Spiral and three engineers (Eng 1, Eng 2, Eng 3). Spiral typically explores both algorithms and code restructurings, potentially reaching the best possible performance for machine, problem, and algorithm.Eng 1 and Eng 2 usually explore one or two aspects, while Eng 3 focuses on the best possible problem performance.](image)
Spiral: Concept

Application specification
Algorithm knowledge

Platform description
Instruction set architecture

_\text{mm\_set1\_epi8}(x) = \ldots
_\text{mm\_xor\_si128}(x,y) = \ldots
_\text{mm\_avg\_epu8}(x,y) = \ldots
_\text{mm\_cmpeq\_epi8}(x,y) = \ldots
_\text{mm\_unpacklo\_epi8}(x,y) = \ldots
\ldots

Optimized implementation
(performance/power/energy)
Spiral: How It Works I

Application specification
Algorithm knowledge

\[
\begin{align*}
\text{DFT}_n &\rightarrow P_{k/2,2m}^T (\text{DFT}_{2m} \oplus (I_{k/2-1} \otimes_i C_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT'}_k \otimes I_m) \\
\text{rDFT}_{2n}(u) &\rightarrow I_{2m}^n \left(I_{k \otimes i} \right| \text{rDFT}_{2m}((i + u)/k) \left| \text{rDHT}_{2m}((i + u)/k) \right) (rDFT_{2k}(u) \otimes I_m) \\
\text{RDFT-3n} &\rightarrow (Q_{k/2,m}^T \otimes I_2) (I_{k \otimes i} \text{rDFT}_{2m}(i + 1/2)/k) (\text{RDFT-3k} \otimes I_m)
\end{align*}
\]

Platform description
Instruction set architecture

\[
\begin{align*}
A_m \otimes I_n &\rightarrow \left( \underbrace{L_{mp}^m \otimes I_{n/p}}_{\text{smp}(p,\mu)} \right) \left( I_p \otimes (A_m \otimes I_{n/p}) \right) \left( L_{p}^{mp} \otimes I_{n/p} \right) \\
I_m \otimes A_n &\rightarrow I_p \otimes \underbrace{(I_{m/p} \otimes A_n)}_{\text{smp}(p,\mu)} \\
(P \otimes I_n) &\rightarrow (P \otimes I_{n/\mu}) \otimes I_{\mu} \underbrace{\text{smp}(p,\mu)}_{\text{smp}(p,\mu)}
\end{align*}
\]

Optimized implementation
(performance/power/energy)
Spiral: How It Works II

- Optimization at the high level of abstraction:
  - Overcomes compiler limitations
- Complete automation

\[
\text{DFT}_{256} \rightarrow \text{functionality} \rightarrow \text{problem specification} \\
\text{OL} \rightarrow \text{Tough optimizations by rewriting:} \\
\text{Σ-OL} \rightarrow \text{Threading} \\
\text{C code} \rightarrow \text{SIMD vectorization} \\
\text{machine code} \rightarrow \text{Streaming} \\
\text{+ threading, vector intrinsics, ...} \rightarrow \text{Locality}
\]
Example: Viterbi decoder

### Select convolutional code
Select a preset code or customize parameters

- **Custom**
- **Voyager**
- **NASA-DSN**
- **CCSDS/NASA-GSFC**
- **WiMAX**
- **CDMA IS-95A**
- **LTE (3GPP - Long Term Evolution)**
- **UWB (802.15)**
- **CDMA 2000**
- **Cassini**
- **Mars Pathfinder & Stereo**

#### rate: $1/2$
- **code rate**
- **constraint length**

#### $K$:
- **polynomials**
- **polynomials for the code in decimal notation**

### Select implementation options
- **frame length**: 2048
- **unpadded frame length in bits**
- **Vectorization level**: SSE 16-way
- **type of code**

### Buttons
- **Generate Code**
- **Reset**

---

"Click": Push-button code generation
“Teaching” Spiral SDR: Algorithm Knowledge

\[ WiFiTX_{k,r,m} \rightarrow \left( I_k \otimes (\text{CP-iDFT}_{64} \circ \text{SymbM}_m \circ \text{Interl}_m) \right) \circ \text{ConvEnc}_{r,\ell} \circ \text{Scamb}_{i_0,\ell} \]

\[
\text{Scamb}_{i_0,\ell} \rightarrow (I_\ell \otimes_i C_i) := \bigoplus_{i=i_0}^{i_0+\ell-1} C_i
\]

\[
\text{ConvEnc}_{r,\ell} \rightarrow \left( T_{r,\ell+6} L_{\ell+6}^{2(\ell+6)} \begin{bmatrix} C_0,\ell \\ C_1,\ell \end{bmatrix} \right)
\]

\[
\text{Interl}_m \rightarrow \left( (I_{16} \otimes_i (I_3 \otimes W_i^m)) L_{16}^{48m} \right)
\]

\[
\text{SymbM}_m \rightarrow G \circ [I_{48} \otimes S_m]
\]

\[
\text{CP-iDFT}_N \rightarrow \left( \frac{1}{\sqrt{N}} P_N \text{iDFT}_N \right)
\]

\text{plus operator definitions and additional rules}
“Teaching” Spiral SDR: Platform Knowledge

- **Rewriting rules**
  - Parallelization
  - SIMD vectorization
  - Cross block optimizations
  - SIMD ISA specific simplification

- **Backend extensions**
  - **New data types**: bit, integer (8, 16, 32)
  - **Vectorization modes**: 128-way, 16-way, 8-way, 4-way (ISAs)
  - **ISA bridges**: data types conversion in vector registers
  - Support for finite field arithmetic
What We Did in a Nutshell

- WiFi 802.11.g (OFDM), 8 data rate modes
  - Expressed in Spiral’s OL
  - Extended OL for mixed data type formulas
  - Extended OL for mixed vector-length formulas

- Created optimized implementations w/ Spiral
  - Threading
  - SIMD vectorization, with mixed data types / vector length
  - Auto-tuning using best algorithm search

- Benchmarked on 3 platforms
  - Intel Atom
  - Intel Core 2
  - Intel Core i7
802.11b Receiver: Generated vs. Hand-tuned

Runtime per symbol [ms]

Lower is better

- SpiralGen, generated: 2.1x
- SORA, Microsoft Research: 1.4x

6 Mb/s: 2.1x more energy efficient (Mflops/watt)

24 Mb/s: 1.4x

54 Mb/s: 1.1x
Compute Time Distribution
Auto-generated receiver on Core i7

Real-time bound = 4 micro-sec

Lower is better

Viterbi decoding is 88% of runtime
Compute-bound Bandwidth
Auto-generated WiFi receiver/transmitter (parallel)

Receiver
Throughput [Mbit/s]

Transmitter
Throughput [Mbit/s]
Ability to Cross-Block Optimize

Spiral Transmitter on Atom
Run time per symbol [micro seconds] vs. Data rate [Mbit/s]

Best partially generated (1 thread)

Best fully generated (1 thread)
Conclusions

- The generated code is *very fast* (often faster than any human written code)

- Spiral supports *different types of architectures* including vector, multicore and distributed platforms

- Spiral enables *very fast transition to new architectures*