Meet Stevephanie, the Computer Spiral: Automatic Library Generation

Franz Franchetti

Electrical and Computer Engineering
Carnegie Mellon University

The Current Architecture Space

before 2000
- Core2 Duo
- Core2 Extreme
- Nvidia GPUs
- ClearSpeed

2008
- Cell BE
  8+1 cores
- Sun Niagara
  32 threads
- IBM Cyclone
  80 cores
- SGI RASC
  Itanium + FPGA
- ATI/AMD merger
  CPU+GPU fusion
- Virtex 5
  FPGA+ 4 CPUs
- Xtreme DATA
  Opteron + FPGA
- IBM Cyclops64
  80 cores

2010 and later
- Larrabee
- 32 threads
- Multicore
- Programmability
Example: DFT on “Nice” Quadcore

Discrete Fourier Transform (single precision): 2 x Core2 Extreme 3 GHz
Performance [Gflop/s]

What’s going on?
Discrete Fourier Transform (DFT) on 2 x Core 2 Duo 3 GHz

Gflop/s

Multiple threads: 2x
SIMD vector instructions: 3x
Memory hierarchy: 5x

High performance is hard even on “simple” architectures
Current Solution

- Legions of programmers implement and optimize the same functionality for every platform and whenever a new platform comes out.
Better: Automatic Performance Tuning

- Automate (parts of) the implementation or optimization

- Research efforts
  - Linear algebra: Phipac/ATLAS, LAPACK, Sparsity/Bebop/OSKI, Flame
  - Tensor computations (TCE)
  - PDE/finite elements: Fenics
  - Adaptive sorting
  - Fourier transform: FFTW, UHFFT
  - Linear transforms: Spiral
  - ...

Promising new area but more work needed
In particular for parallelism…

Proceedings of the IEEE special issue, Feb. 2005
Spiral

- Library generator for linear transforms (DFT, DCT, DWT, filters, ....) \textit{and recently more ...}

- Wide range of platforms supported: scalar, fixed point, \textit{vector, parallel, Verilog, GPU}

- Research Goal: \textit{“Teach” computers to write fast libraries}
  - Complete automation of implementation and optimization
  - Conquer the “high” algorithm level for automation

- When a new platform comes out: Regenerate a retuned library

- When a new platform paradigm comes out (e.g., CPU+GPU): Update the tool rather than rewriting the library

\textit{Intel is using Spiral to generate parts of their MKL and IPP libraries}
Vision Behind Spiral

**Current**

- Numerical problem
- Algorithm selection
- Implementation
- Compilation

**Future**

- Numerical problem
- Algorithm selection
- Implementation
- Compilation

*C code a singularity: Compiler has no access to high level information*

*Challenge: conquer the high abstraction level for complete automation*
Main Idea: Joint Mathematical Abstraction

Model: common abstraction
= spaces of matching formulas

Architectural parameter:
Vector length,
#processors, ...

Kernel:
problem size,
algorithm choice
How Spiral Works

Spiral:
Complete automation of the implementation and optimization task

Basic idea:
Declarative representation of algorithms

Rewriting systems to generate and optimize algorithms

Problem specification (transform)

- Algorithm Generation
- Algorithm Optimization
- Implementation
- Code Optimization
- Compilation
- Compiler Optimizations

Fast executable

Spiral: Complete automation of the implementation and optimization task

Basic idea:
Declarative representation of algorithms

Rewriting systems to generate and optimize algorithms
Generating, Not Finding Parallelism

- Shared Memory (SMP, CMP)
- Vector SIMD (SSE, VMX, Double FPU...)
- Message Passing (Clusters)
- DMA+Streaming (Cell BE)
- Graphics Processors (GPUs)
- Pipelining+Streaming (FPGAs)
- HW/SW partitioning (CPU+FPGA)

One methodology optimizes for all types of parallelism
Program Generation in Spiral (Sketched)

Transform
user specified

Fast algorithm
in SPL
many choices

Optimization at all abstraction levels

parallelization
vectorization
loop optimizations
constant folding scheduling

\[ \text{DFT}_8 \]

\[ (\text{DFT}_2 \otimes I_4) T^8_4 (I_2 \otimes ((\text{DFT}_2 \otimes I_2) \cdot T^4_2 (I_2 \otimes \text{DFT}_2) L^4_2)) L^8_2 \]

\[ \sum (S_j \text{DFT}_2 G_j) \sum \left( \sum \left( S_{k,l} \text{diag}(t_{k,l}) \text{DFT}_2 G_l \right) \right) \sum \left( S_m \text{diag}(t_m) \text{DFT}_2 G_{k,m} \right) \]

\[
\text{void sub}(\text{double } *y, \text{double } *x) \{
\text{double } f0, f1, f2, f3, f4, f7, f8, f10, f11;
\text{f0} = x[0] - x[3];
\text{f1} = x[0] + x[3];
\text{f2} = x[1] - x[2];
\text{f3} = x[1] + x[2];
\text{f4} = f1 - f3;
\text{y}[0] = f1 + f3;
\text{y}[2] = 0.7071067811865476 + f1;
\text{f7} = 0.9238795325112867 * \text{f0};
\text{f8} = 0.3826834323650898 * \text{f2};
\text{y}[1] = \text{f7} + \text{f8};
\text{f10} = 0.3826834323650898 * \text{f0};
\text{f11} = (-0.9238795325112867) * \text{f2};
\text{y}[3] = \text{f10} + \text{f11};
\} \]
Behind The Scenes: Domain-Specific Rewriting

DFT_n → (DFT_k ⊗ I_m) T_m^n (I_k ⊗ DFT_m) L_k^n, n = km
DFT_n → P_n (DFT_k ⊗ DFT_m) Q_n, n = km, gcd(k,m) = 1
DFT_p → R_p^T (I_1 ⊕ DFT_{p-1}) D_p (I_1 ⊕ DFT_{p-1}) R_p, p prime

- “Teaches” Spiral about existing algorithm knowledge (~200 journal papers)
- “Teaches” Spiral about hardware properties
- “Teaches” Spiral about program transformations

Algorithm Transformations
Formulas Encode Program Properties

- Embarrassingly parallel, load balanced computation

\[ y = \left( I_p \otimes A \right) x \]

- False sharing

\[ y = L^\otimes_4 x \]

Span space of false-sharing free programs for empirical tuning
Going Beyond Transforms

- **Transform** = 
  linear operator with one vector input and one vector output

- **Key ideas:**
  - Generalize to (possibly nonlinear) operators with several inputs and several outputs
  - Generalize SPL (including tensor product) to OL (operator language)
  - Generalize rewriting systems for parallelizations
Expressing Kernels as Operator Formulas

Matrix-Matrix Multiplication

\[ \text{MMM}_{1,1,1} \rightarrow (\cdot)_1 \]
\[ \text{MMM}_{m,n,k} \rightarrow (\otimes)_{m/m_n \times 1} \otimes \text{MMM}_{m,n,k} \]
\[ \text{MMM}_{m,n,k} \rightarrow \text{MMM}_{m,nb,k} \otimes (\otimes)_{1 \times n/nb} \]
\[ \text{MMM}_{m,n,k} \rightarrow ((\Sigma_{l/b} \circ (\cdot)_{l/b}) \otimes \text{MMM}_{m,n,b,k}) \circ ((L^{m/k}_{l/b} \otimes I_{kn}) \times I_{kn}) \]
\[ \text{MMM}_{m,n,k} \rightarrow (L^{m/n}_m \otimes I_{n_k}) \circ ((\otimes)_{1 \times n/nb} \otimes \text{MMM}_{m,nb,k}) \circ (I_{kn} \times (L^{n/n_k} \otimes I_{n_k})) \]

JPEG 2000 (Wavelet, EBCOT)

JPEG 2000 Compression

- DWT
- quantization
- entropy coding (EBCOT + MQ)

Viterbi Decoder

\[ \mathbf{V}_{\text{vec}(v)} \rightarrow (\prod (L \times I) \circ (I \otimes C)) \circ \text{Id} \]
\[ \mathbf{V}_{\text{vec}(v)} \rightarrow (\prod (L \otimes I) \circ (I \otimes C)) \circ \text{Id} \]
\[ \times \rightarrow (\prod (L \otimes I) \circ (I \otimes C)) \circ \text{Id} \]
\[ \prod (L \otimes I) \circ (I \otimes (B \otimes I_r)) \circ (\bar{L} \times I) \]

Synthetic Aperture Radar (SAR)

- preprocessing
- matched filtering
- interpolation
- 2D iFFT

SAR \[ \rightarrow \text{2D-iDFT} \circ \text{Interpl} \circ \text{MatchFilt} \circ \text{prep} \]
2D-iDFT \[ \rightarrow \text{iDFT} \otimes \text{iDFT} \]
MatchFilt \[ \rightarrow \text{Filt} \circ (I \times C_f) \]
Filt \[ \rightarrow (I \otimes (\cdot)) \]
Interpl \[ \rightarrow (\Sigma \otimes 1) \circ (I \otimes J_{S_x \otimes g_y}) \circ \text{Filt} \circ ((I \otimes 1 \otimes 1) \times I) \circ (I \times C_{t \otimes g_3}) \]

\[ h \circ (f \times f) \circ (G_1 \times C_{-2} \times G_1 \times G_7 \times C_{-2} \times G_7) \circ L^2_1 \circ (\frac{1}{1}) \times (\frac{1}{1}) \]
\[ h \circ (f \times f) \circ (G_3 \times C_{-2} \times G_3 \times G_5 \times C_{-2} \times G_5) \circ L^2_1 \circ (\frac{1}{1}) \times (\frac{1}{1}) \]
\[ f : \text{mul}_2 \circ (I \times \text{sub}_2) \circ (I \times C_1 \times \text{mul}_2) \]
\[ h : \text{min}_2 \circ (C_1 \times \text{max}_2) \circ (C_{-1} \times \text{sum}_2) \]
General-Size Library

Input:

- Transform: $\text{DFT}_n$
- Algorithms: $\text{DFT}_{km} \rightarrow (\text{DFT}_b \otimes \text{I}_m) \text{T}_m^{km} (\text{I}_b \otimes \text{DFT}_m) \text{L}_k^{km}$
  
  $\text{DFT}_2 \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
- Vectorization: 2-way SSE
- Threading: Yes

Output:

- Optimized library (10,000 lines of C++)
- For general input size
  (not collection of fixed sizes)
- Vectorized
- Multithreaded
- With runtime adaptation mechanism
- Performance competitive with hand-written code
Generating General-Size Libraries

\[ \text{DFT}_{km} \rightarrow (\text{DFT}_k \otimes I_m)T_{km}^{km} (I_k \otimes \text{DFT}_m)L_{km}^{km} \]

Spiral Library Generator

Recursion

```c
void compute(Y, X, ...) {
    for(i=0..k-1) c1->compute(...)
    for(j=0..m-1) c2->compute(...)
}
```

Base cases

```c
void dft2 (...) {
    y[0]=x[0]+x[1]
    y[1]=x[0]-x[1]
}
```

Target Infrastructure

```c
fftw_plan_dft_1d(...) {
    hand->buf=
    fftw_malloc(...);
    ...
}
```

Spiral generates whole library: FFTW equivalent and more
Benchmark: DFT on “Nice” Quadcore

DFT (single precision): on 3 GHz 2 × Core 2 Extreme
performance [Gflop/s]

- Spiral 5.0 SPMD
- Spiral 5.0 sequential
- Intel IPP 5.0
- FFTW 3.2 alpha SMP
- FFTW 3.2 alpha sequential

4-way vectorized + up to 4-threaded + adapted to the memory hierarchy

4 processors
2 processors
data L1$ resident
data L2$ resident
Summary

- Libraries must be well-tuned
  Nightmare: locality, multicore, special instructions,…

- Automating the tuning process is desirable
  Computer-generated code can beat hand-tuned code

- High-level abstraction is key to performance portability
  That does not imply “easy to program”

- Research challenge: Automate more domains
  Program generators must become part of the ecosystem
Acknowledgement: Spiral

www.spiral.net

Joint work with
Yevgen Voronenko
Frédéric de Mesmay
Markus Püschel

... and the Spiral team (only part shown)

This work was supported by
DARPA DESA program, NSF-NGS/ITR, NSF-ACR, Mercury Inc., and Intel