Software Engineering Seminar

Fall 2011
Lecture 1

Instructor: Markus Püschel
TA: Georg Ofenbeck

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Today

- Course organization
- Automatic performance tuning

Course

- Number: 252-2600
- 2 credits
- Course website:
  http://people.inf.ethz.ch/markusp/teaching/252-2600-ETH-fall11/course.html

Course Goals

- Introduction to research in software engineering
- Learn how to read and understand research papers
- Learn how to give a good technical presentation to peers

- General topic this semester: *Automatic Performance Tuning*

How It Works

- **Every students gets a research paper, main advisor, and date assigned within the next week**
- **Understand the paper**
- **Create a presentation**
- **Have a meeting with main advisor (TA or me)**
- **Present at your assigned date**

Understand the Paper

- **Study it carefully**
- **Obtain and study relevant background material, e.g.,**
  - Read papers that are cited
  - Look up and understand technical terms and concepts used
- **If needed, meet with TA or instructor for help**

Create a Presentation

- **Try to follow the guidelines presented in the first lectures**
- **Should include:**
  - Clear motivation for the work
  - Clear explanation what the paper does
  - Understandable (by your fellow students) presentation of content and results
  - Brief critical discussion in the end of the contribution: strong and weak parts including limitations
- **Present the crucial content and not everything**
- **Strive for high visual quality**
- ***Acknowledge any external material* (graphics, anything included by copy-paste from other sources) on the same slide**

Meeting With Main Advisor

- **Ask some final questions**
- **Strongly recommended: bring draft of presentation for feedback**

Present at Your Assigned Date

- **30 minutes presentation + 15 minutes for questions**
- **Presentation time is strictly enforced (as in the real world)**

Grading

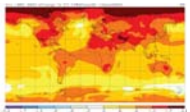
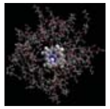
- **Quality of presentation and question handling**
  - How well you understood the paper
  - How understandable you presented it
  - How effectively your slides communicated (includes visual quality)
- **I understand that the papers have varying difficulty and will take that into account**
- **Presence and participation**
  - Presence will be recorded
  - If you miss many classes you fail (“many” starts very early for me)
  - Conflicts (military duties etc.): questionnaire

Today

- Course organization
- **Automatic performance tuning**
  - Problem and motivation
  - A glimpse of some research efforts

Computing

Scientific Computing

Physics/biology simulations

Consumer Computing


Audio/image/video processing

Embedded Computing

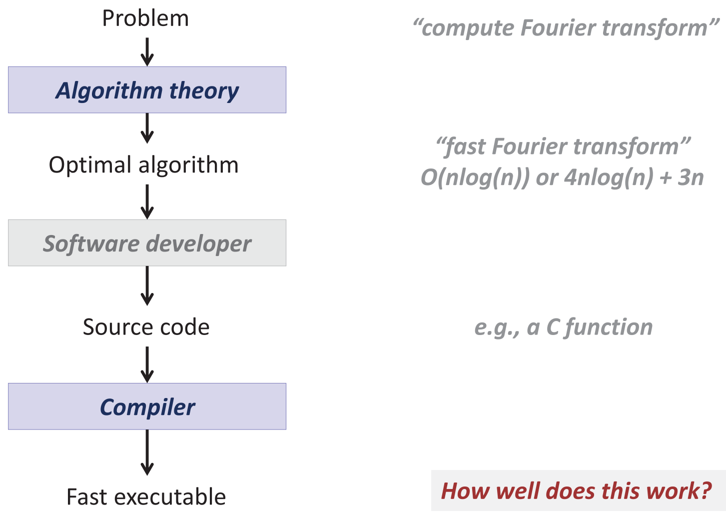

Signal processing, communication, control

- **Unlimited need for performance**
- **Large set of applications, but …**
- **Relatively small set of critical components (100s to 1000s)**
  - Matrix multiplication
  - Discrete Fourier transform (DFT)
  - Viterbi decoder
  - Shortest path computation
  - Stencils
  - Solving linear system
  - ….

Classes of Performance-Critical Functions

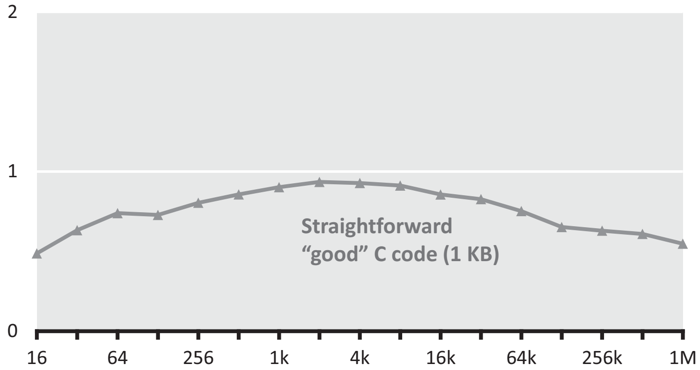
- **Transforms**
- **Filters/correlation/convolution/stencils/interpolators**
- **Dense linear algebra functions**
- **Sparse linear algebra functions**
- **Coder/decoders**
- ***… many others***

How Hard Is It to Get Fast Code?



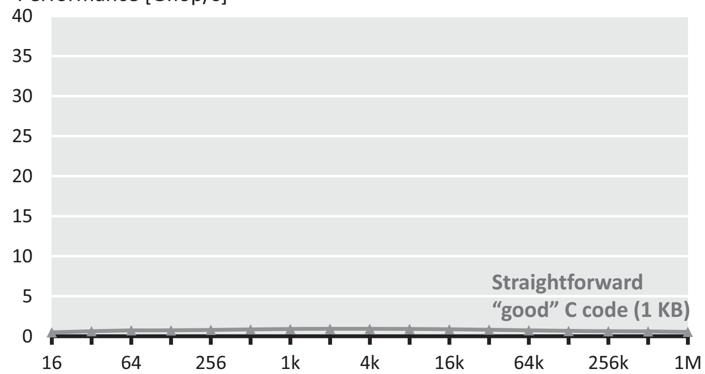
The Problem: Example 1

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

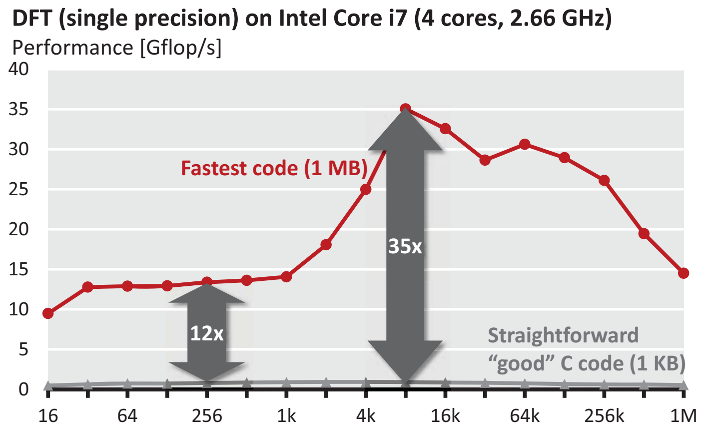


The Problem: Example 1

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

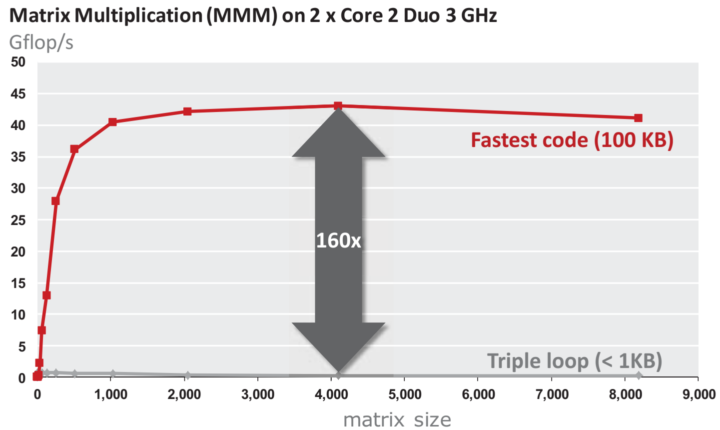


The Problem: Example 1



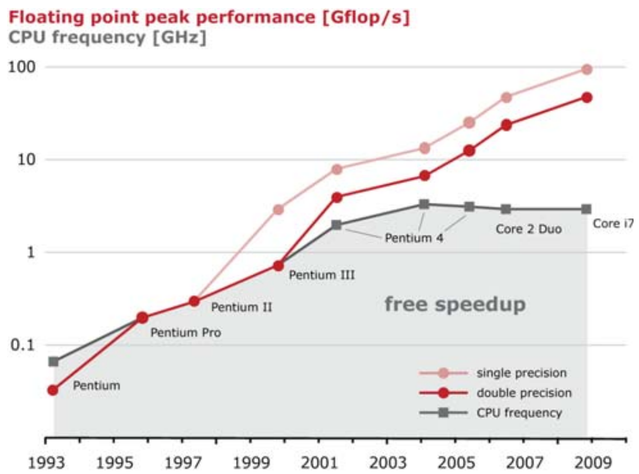
- Vendor compiler, best flags
- Roughly same operations count

The Problem: Example 2

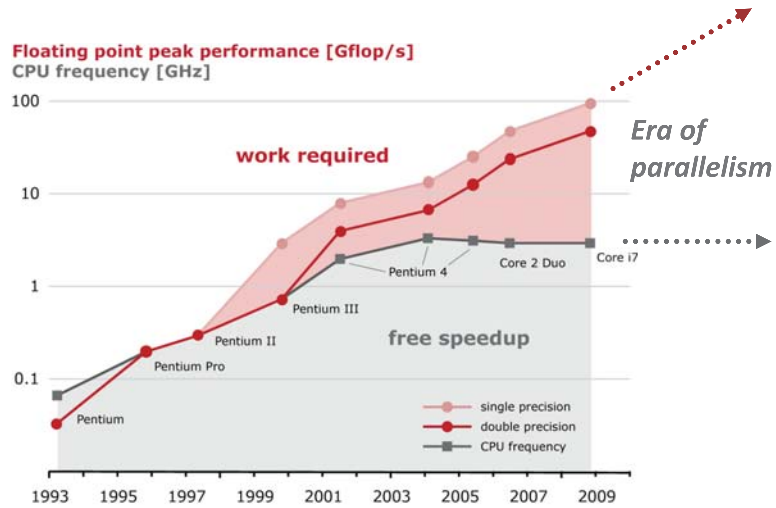


- Vendor compiler, best flags
- Exact same operations count ($2n^3$)
- ***What is going on?***

Evolution of Processors (Intel)



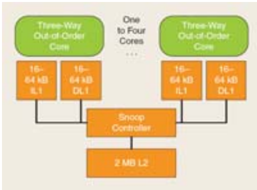
Evolution of Processors (Intel)



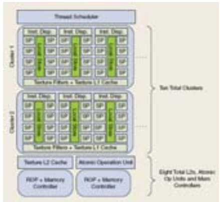
And There Will Be Variety …

Core i7

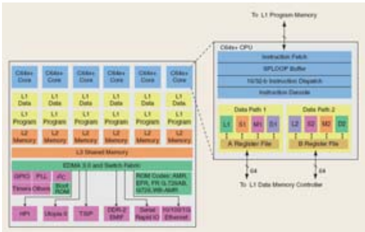
Arm Cortex A9



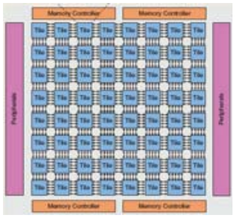
Nvidia G200



TI TNETV3020



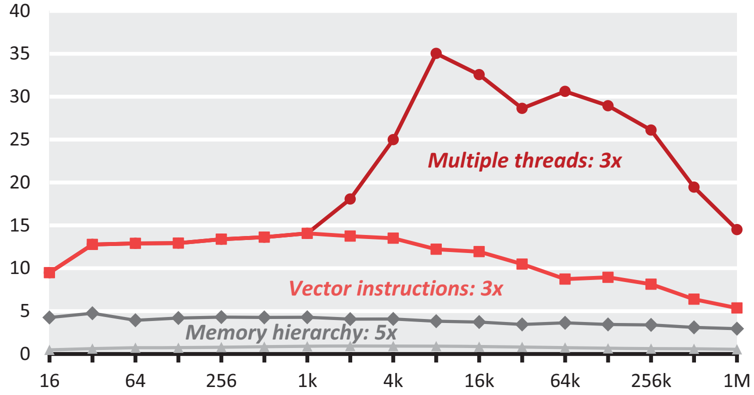
Tilera Tile64



Source: IEEE SP Magazine, Vol. 26, November 2009

DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

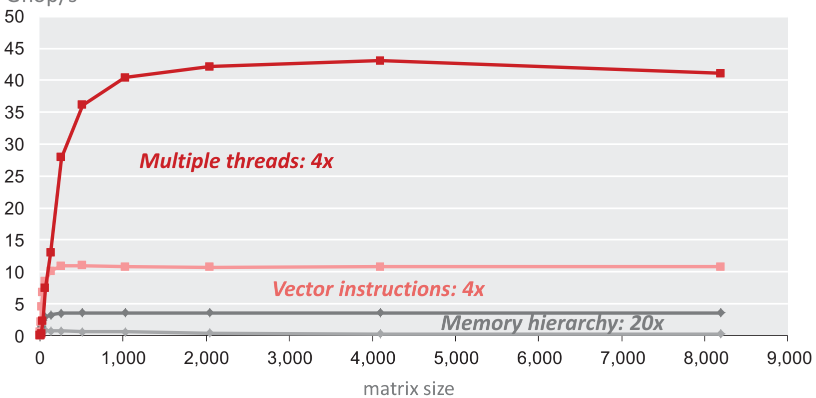
Performance [Gflop/s]



- Compiler doesn't do the job
- Doing by hand: ***nightmare***

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz

Gflop/s



- Compiler doesn't do the job
- Doing by hand: ***nightmare***

Summary and Facts I

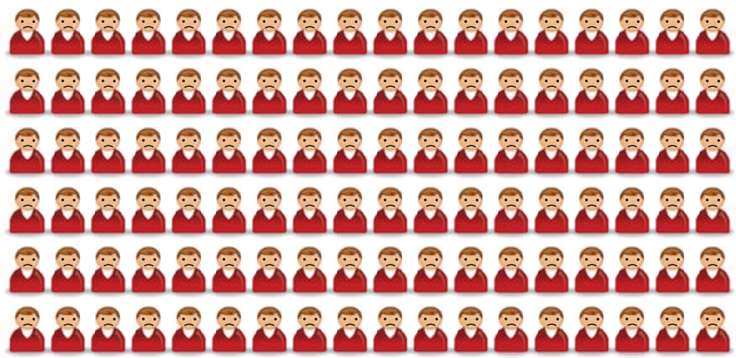
- **Implementations with same operations count can have vastly different performance (up to 100x and more)**
  - Code style
  - A cache miss can be 100x more expensive than an operation
  - Vector instructions
  - Multiple cores = processors on one die
- **Minimizing operations count ≠ maximizing performance**
- **End of free speed-up for legacy code**
  - Future performance gains through increasing parallelism

Summary and Facts II

- **It is very difficult to write the fastest code**
  - Tuning for memory hierarchy
  - Vector instructions
  - Efficient parallelization (multiple threads)
  - Requires expert knowledge in algorithms, coding, and architecture
- **Fast code can be large**
  - Can violate "good" software engineering practices
- **Compilers often can't do the job**
  - Code style
  - Often intricate changes in the algorithm required
  - Parallelization/vectorization still unsolved
- **Highest performance is in general non-portable**

Performance/Productivity **Challenge**

Current Solution



- ***Legions*** of programmers implement and optimize the ***same functionality*** for ***every platform*** and whenever a ***new platform*** comes out

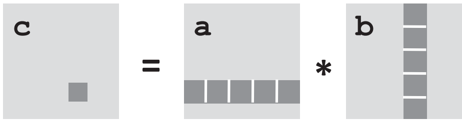
Better Solution: Autotuning

- **Automate (parts of) the implementation or optimization**

- **Relatively recent research area (since late nineties)**
- **Techniques used:**
  - Program generation
  - Empirical search over alternatives for the fastest
  - Machine learning
  - Performance models
  - Adaptive libraries
  - Domain-specific languages
  - Rewriting systems

PhiPac/ATLAS: MMM Generator

Whaley, Bilmes, Demmel, Dongarra, …

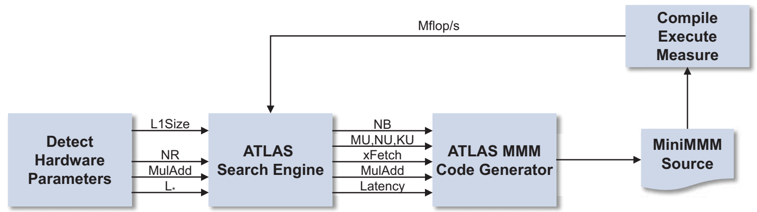


Blocking improves locality

```
c = (double *) calloc(sizeof(double), n*n);

/* Multiply n x n matrices a and b  */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=B)
        for (j = 0; j < n; j+=B)
            for (k = 0; k < n; k+=B)
                /* B x B mini matrix multiplications */
                for (i1 = i; i1 < i+B; i++)
                    for (j1 = j; j1 < j+B; j++)
                        for (k1 = k; k1 < k+B; k++)
                            c[i1*n+j1] += a[i1*n + k1]*b[k1*n + j1];
}
```

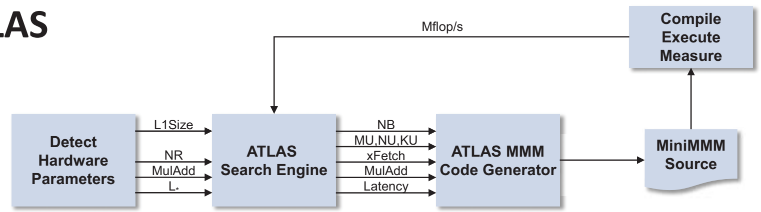
PhiPac/ATLAS: MMM Generator



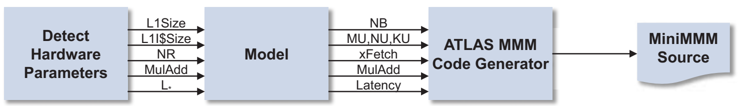
- ***Techniques:***
  - Program generation (here: template-based)
  - Feedback-driven search over a set of parameters

source: Pingali, Yotov, Cornell U.

ATLAS



Model-Based ATLAS (Yotov et al.)



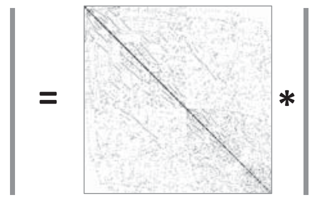
$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + 3\left\lceil \frac{N_B \times N_U}{B_1} \right\rceil + \left\lceil \frac{M_U}{B_1} \right\rceil \times N_U \leq \frac{C_1}{B_1}$$

- ***Techniques:***
  - Hardware parameter based model

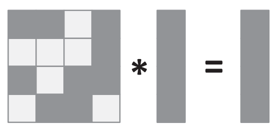
source: Pingali, Yotov, Cornell U.

OSKI: Sparse Matrix-Vector Multiplication

Vuduc, Im, Yelick, Demmel

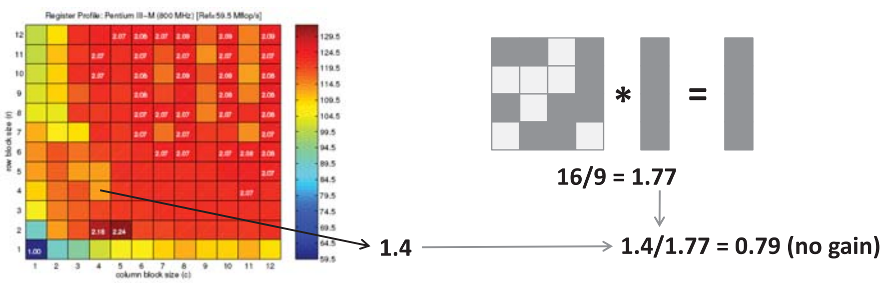


- **Blocking for registers:**
  - Improves locality (reuse of input vector)
  - But creates overhead (zeros in block)



OSKI: Sparse Matrix-Vector Multiplication

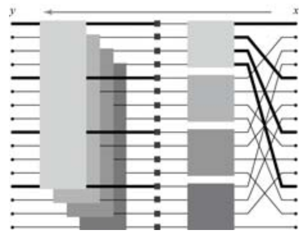
Gain by blocking (dense MVM) **Overhead by blocking**



- ***Techniques:***
  - Measurement-based model
  - Data structure adaptation

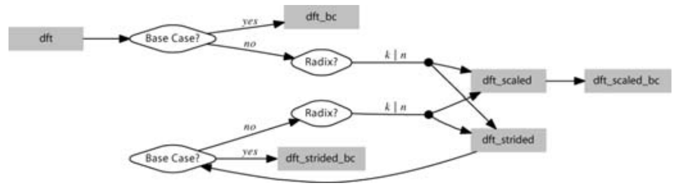
FFTW: Discrete Fourier Transform

Frigo, Johnson



```
void dft(int n, cpx *y, cpx *x) {
    if (use_dft_base_case(n))
        dft_bc(n, y, x);
    else {
        int k = choose_dft_radix(n);
        for (int i=0; i < k; ++i)
            dft_strided(m, k, t + m*i, x + m*i);
        for (int i=0; i < m; ++i)
            dft_scaled(k, m, precomp_d[i], y + i, t + i);
    }
}
```

Choices used for adaptation



Vectorization, threading, etc. add more choices

FFTW: Discrete Fourier Transform

Installation

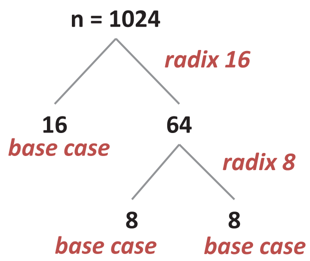
```
configure/make
```

Usage

```
d = dft(n)
d(x,y)
```

Twiddles

Search for fastest computation strategy



- ***Techniques:***
  - Adaptive library
  - Dynamic programming search
  - Not explained: Program generator for basic blocks

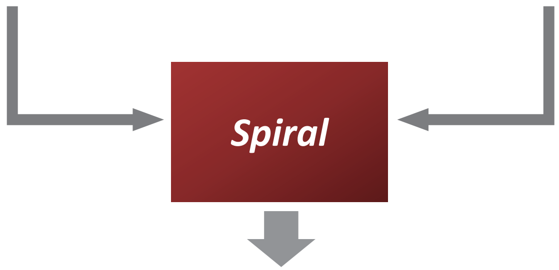
Spiral: Linear Transforms & More

Franchetti, Voronenko, Püschel, Xiong, Singer, Moura, Johnson, Padua, …

Algorithm knowledge

Platform description

_mm_set1_epi8(x) = …
_mm_xor_si128(x,y) = …
_mm_avg_epu8(x,y) = …
_mm_cmpeq_epi8(x,y) = …
_mm_unpacklo_epi8(x,y) = …
…



Optimized implementation

(regenerated for every new platform)

Spiral: Linear Transforms & More

Algorithm knowledge

$$\mathbf{DFT}_n \to P_{k/2,2m}^{\top}\left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m}\,\mathbf{rDFT}_{2m}(i/k)\right)\right)\left(\mathbf{RDFT}'_k \otimes I_m\right)$$

$$\left|\begin{matrix}\mathbf{rDFT}_{2n}(u)\\ \mathbf{rDHT}_{2n}(u)\end{matrix}\right| \to L_m^{2n}\left(I_k \otimes_i \left|\begin{matrix}\mathbf{rDFT}_{2m}((i+u)/k)\\ \mathbf{rDHT}_{2m}((i+u)/k)\end{matrix}\right|\right)\left(\left|\begin{matrix}\mathbf{rDFT}_{2k}(u)\\ \mathbf{rDHT}_{2k}(u)\end{matrix}\right| \otimes I_m\right)$$

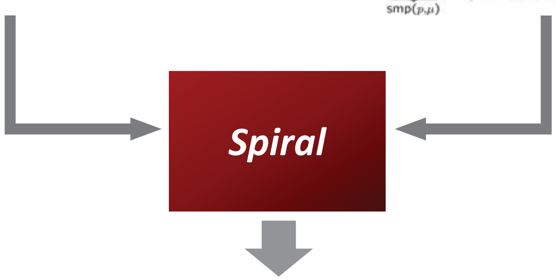
$$\mathbf{RDFT\text{-}3}_n \to (Q_{k/2,m}^{\top} \otimes I_2)\,(I_k \otimes_i \mathbf{rDFT}_{2m})(i+1/2)/k))\,(\mathbf{RDFT\text{-}3}_k \otimes I_m)$$

Platform description

$$\underbrace{A_m \otimes \mathrm{I}_n}_{\mathrm{smp}(p,\mu)} \to \underbrace{\left(\mathrm{L}_m^{mp} \otimes \mathrm{I}_{n/p}\right)\left(\mathrm{I}_p \otimes (A_m \otimes \mathrm{I}_{n/p})\right)\left(\mathrm{L}_p^{mp} \otimes \mathrm{I}_{n/p}\right)}_{\mathrm{smp}(p,\mu)}$$

$$\underbrace{\mathrm{I}_m \otimes A_n}_{\mathrm{smp}(p,\mu)} \to \mathrm{I}_p \otimes_{\parallel} \left(\mathrm{I}_{m/p} \otimes A_n\right)$$

$$\underbrace{(P \otimes \mathrm{I}_n)}_{\mathrm{smp}(p,\mu)} \to \left(P \otimes \mathrm{I}_{n/\mu}\right)\bar{\otimes}\,\mathrm{I}_\mu$$



Optimized implementation
(regenerated for every new platform)

Program Generation in Spiral (Sketched)

Transform
user specified

$$\mathsf{DFT}_8$$

Algorithm rules

Fast algorithm in SPL
many choices

$$(\mathsf{DFT}_2 \otimes I_4)\,T_4^8\,(I_2 \otimes ((\mathsf{DFT}_2 \otimes I_2) \cdot T_2^4\,(I_2 \otimes \mathsf{DFT}_2)\,L_2^4))\,L_2^8$$

∑-SPL

$$\sum\left(S_j\,\mathsf{DFT}_2\,G_j\right)\sum\left(\sum\left(S_{k,l}\,\mathrm{diag}\left(\mathrm{t}_{k,l}\right)\mathsf{DFT}_2\,G_l\right)\sum\left(S_m\,\mathrm{diag}\left(\mathrm{t}_m\right)\mathsf{DFT}_2\,G_{k,m}\right)\right)$$

Optimized implementation

Optimization at all abstraction levels

- parallelization vectorization
- loop optimizations
- constant folding scheduling ……

- *Techniques:*
  - Domain-specific language (declarative, mathematical, point-free)
  - Rewriting for optimization
  - Search techniques
  - …

This Seminar

- **A collection of papers in the domain of autotuning**
- **Somewhat interdisciplinary**
- **More detailed problem motivation: read first 7 pages of this http://spiral.ece.cmu.edu:8080/pub-spiral/pubfile/paper_100.pdf**
- **For a more complete introduction to performance optimization, take the course: *How to write fast numerical code***