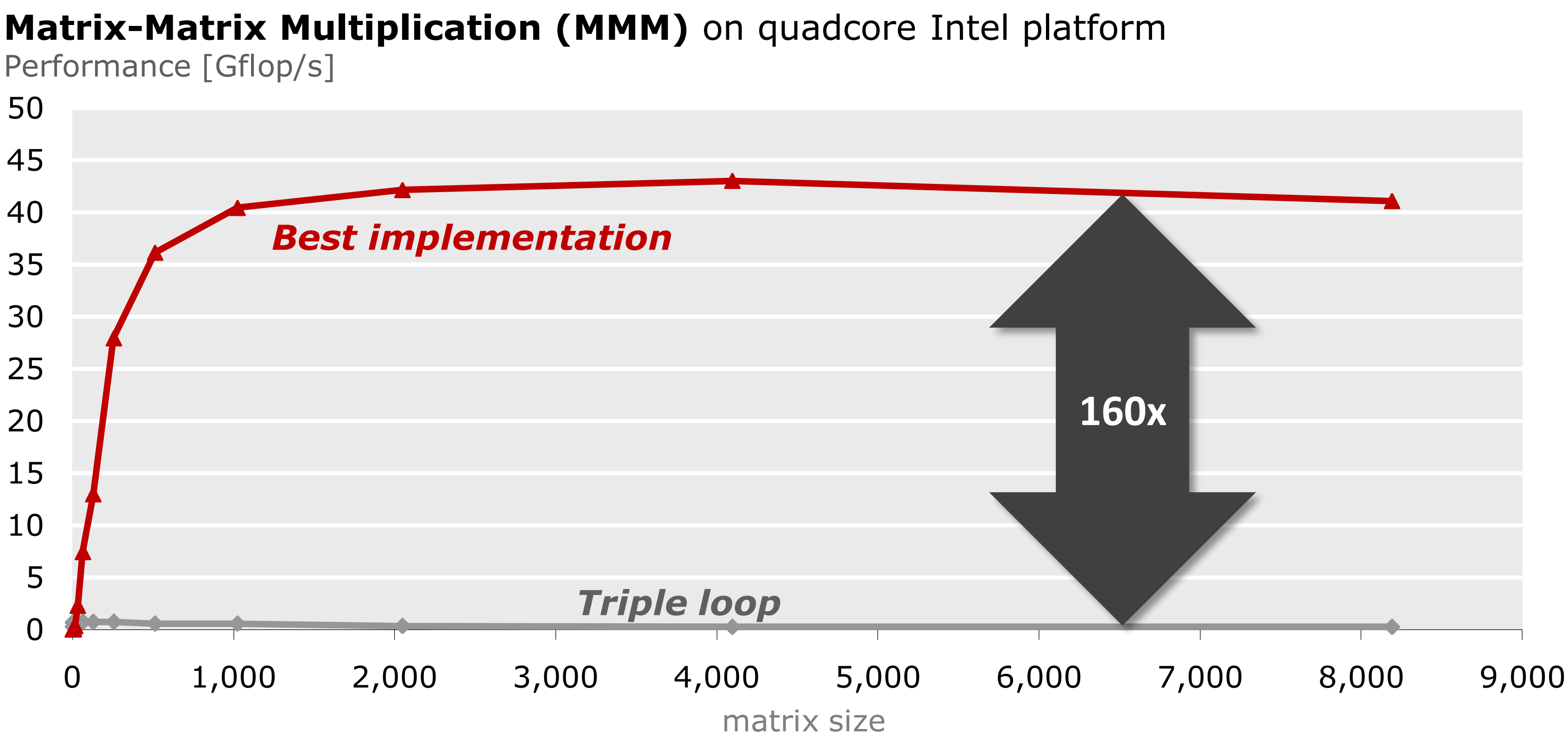
Autotuning

Markus Püschel

Electrical and Computer Engineering
Carnegie Mellon University

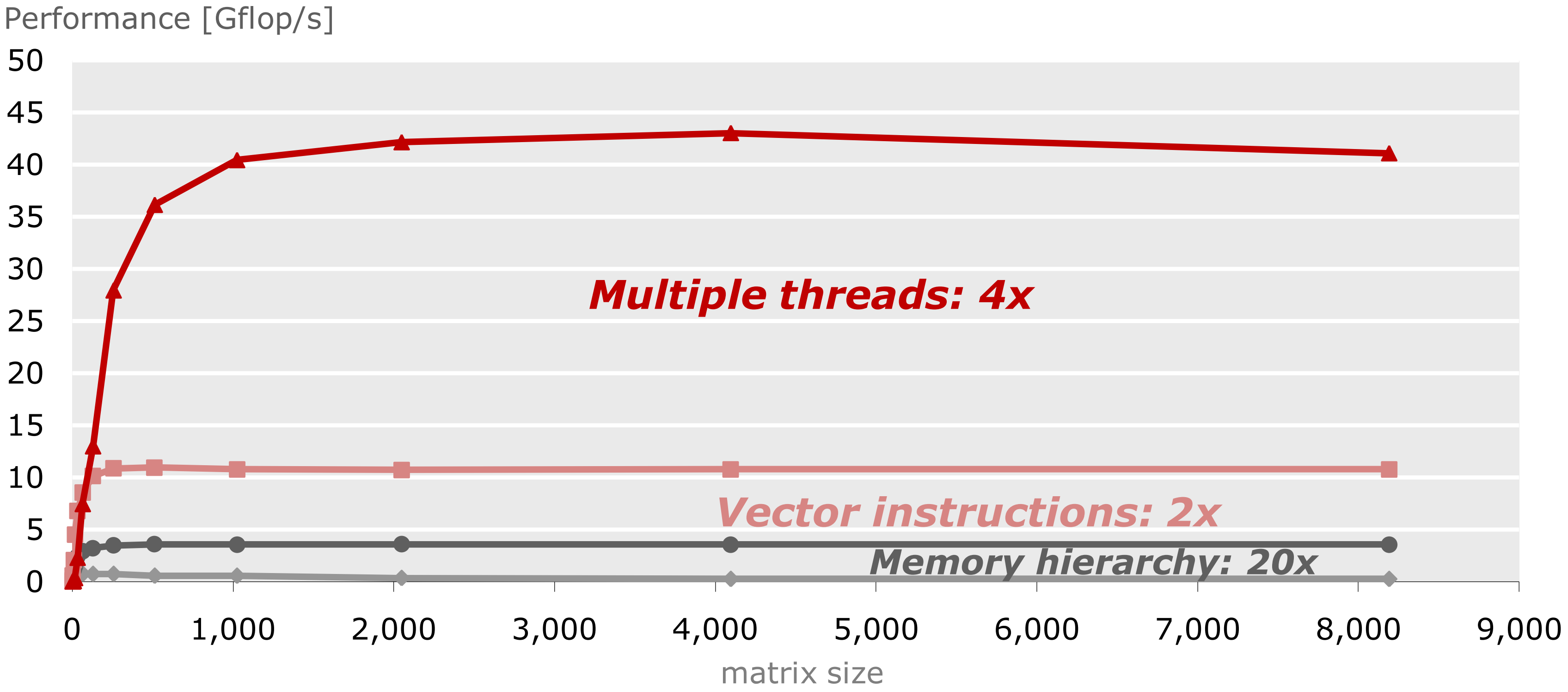
Why Autotuning?



- **Same (mathematical) operation count ($2n^3$)**
- **Compiler underperforms by 160x**

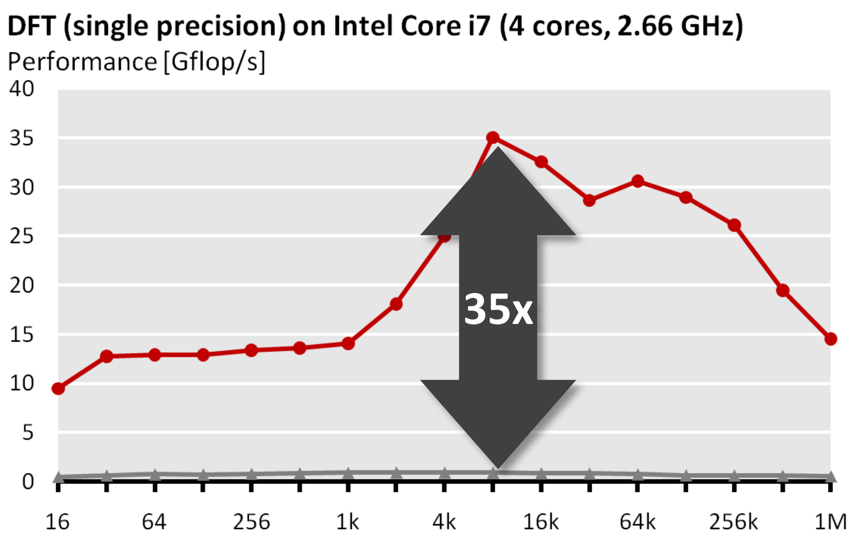
Why Autotuning?

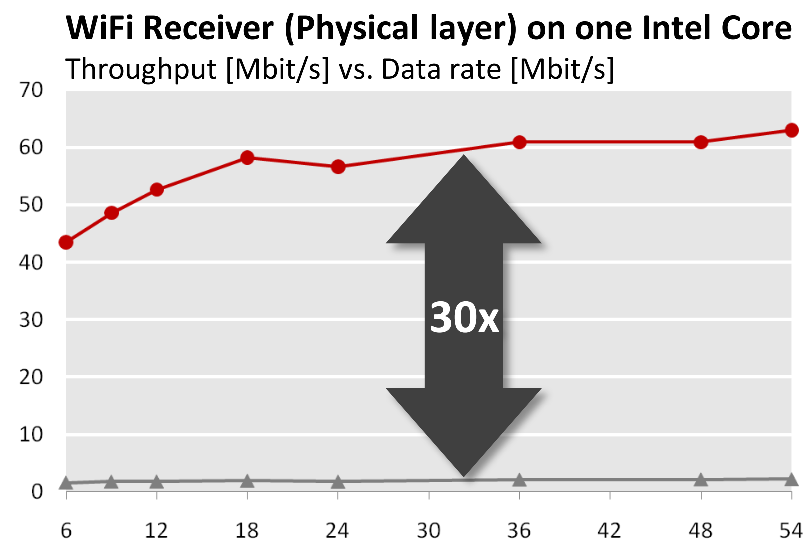
Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Extreme 3 GHz



- **Code size:**
  - Triple loop: < 1 KB
  - Best code: about 100 KB

Same for All Critical Compute Functions



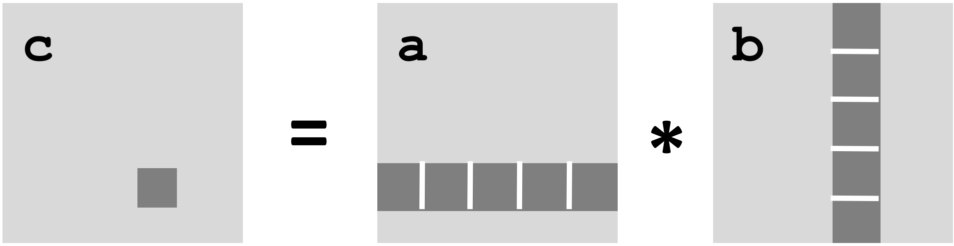


Summary of the Situation: *Huge Problem*

- **For computing functions compilers underperform by 10–100x**
- **Reason: the following are unsolved problems**
  - Memory hierarchy optimizations
  - Vectorization
  - Parallelization
- **Optimization by hand requires highly skilled programmers**
- **Performance does not port well**
- ***Solution (autotuning):*** *Automating performance optimization with tools that complement/aid the compiler or programmer*

PhiPac/ATLAS: MMM Generator

Whaley, Bilmes, Demmel, Dongarra, …

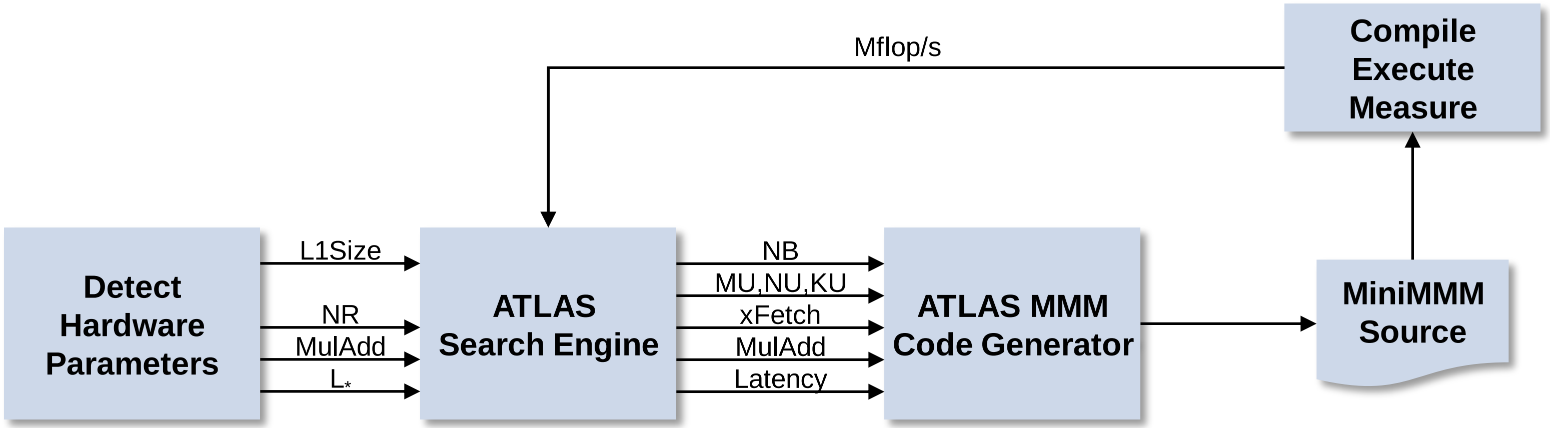


Blocking improves locality

```
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];
}
```

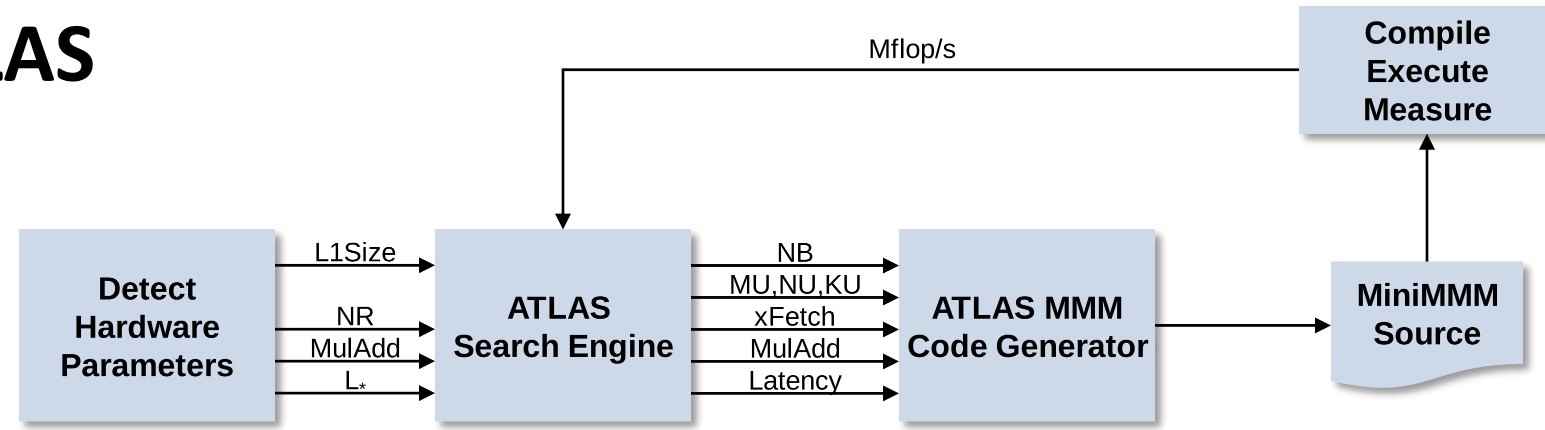
PhiPac/ATLAS: MMM Generator



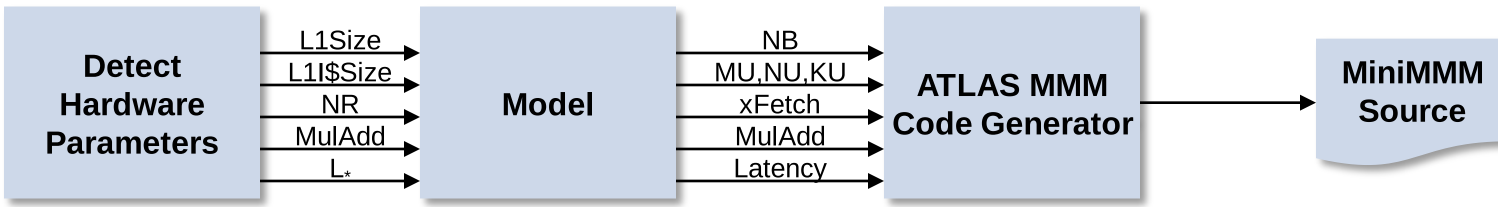
- ***Techniques:***
  - Program generation (here: template-based)
  - Feedback-driven search over a set of parameters

source: Pingali, Yotov, Cornell U.

ATLAS



Model-Based ATLAS (Yotov et al.)



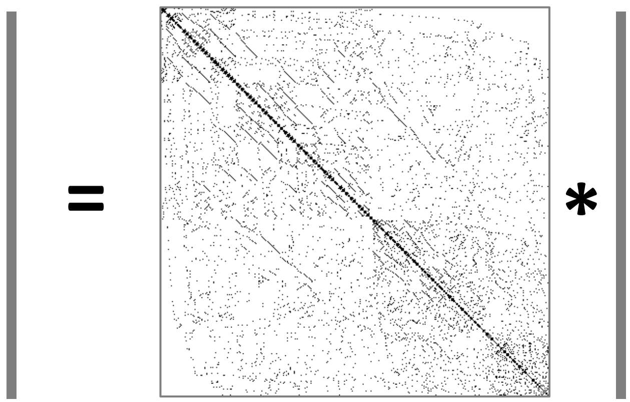
$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + 3 \left\lceil \frac{N_B \times N_U}{B_1} \right\rceil + \left\lceil \frac{M_U}{B_1} \right\rceil \times N_U \leq \frac{C_1}{B_1}$$

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

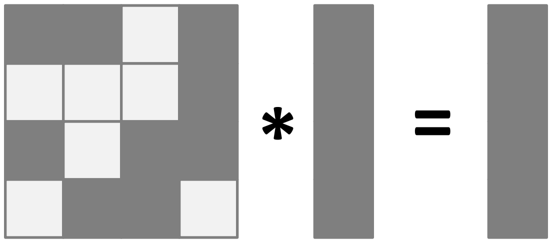
source: Pingali, Yotov, Cornell U.

OSKI: Sparse Matrix-Vector Multiplication

Vuduc, Im, Yelick, Demmel

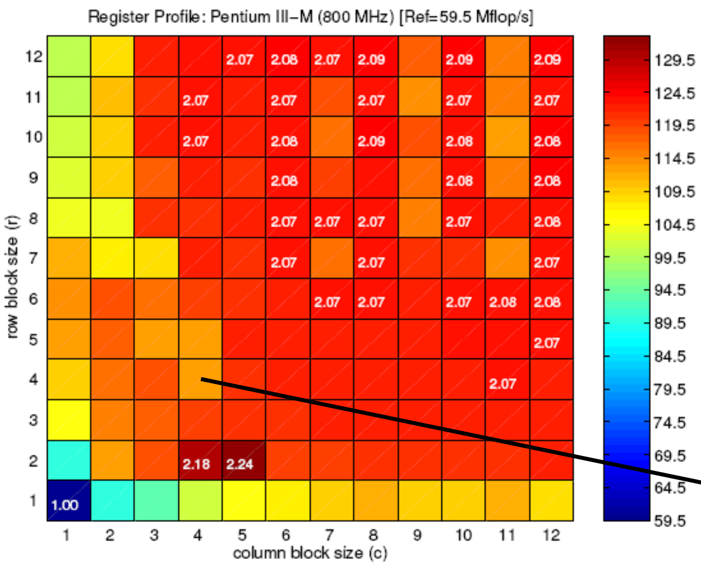


- **Blocking for registers:**
  - Improves locality (reuse of input vector)
  - But creates overhead (zeros in block)



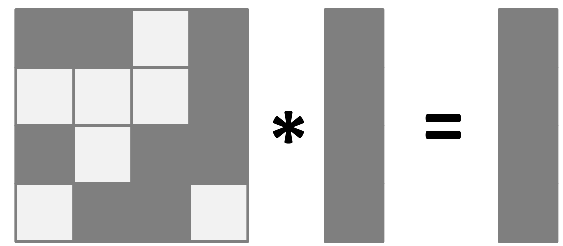
OSKI: Sparse Matrix-Vector Multiplication

Gain by blocking (dense MVM)



1.4

Overhead by blocking



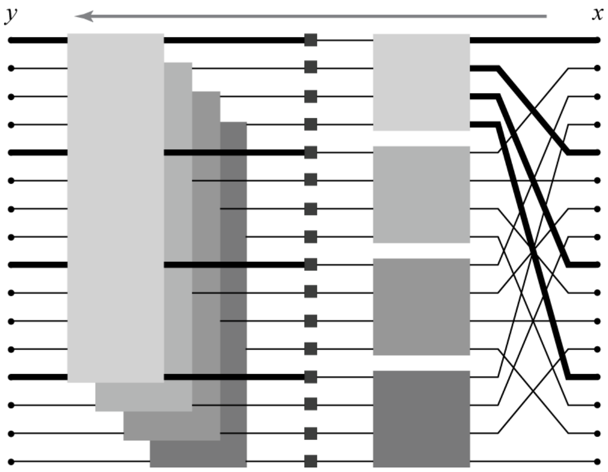
16/9 = 1.77

1.4/1.77 = 0.79 (no gain)

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

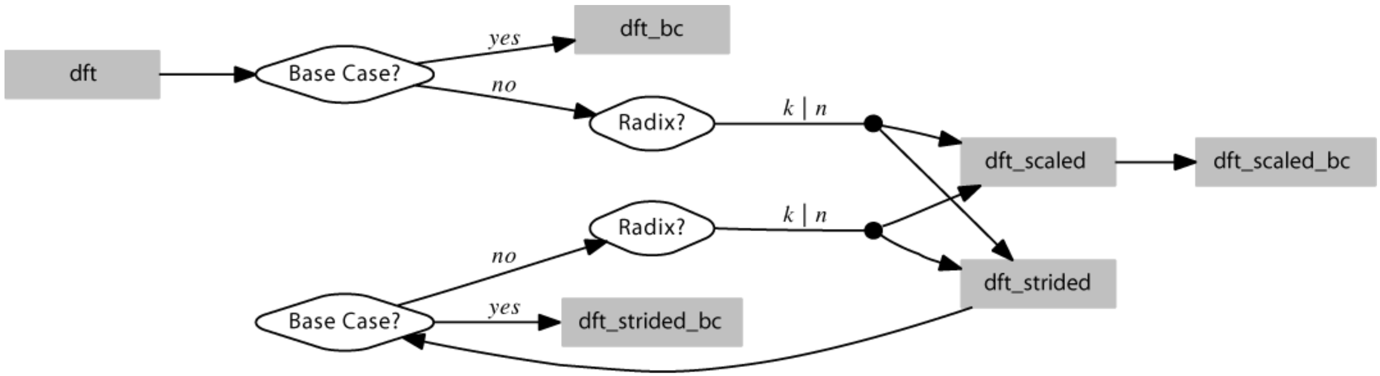
FFTW: Discrete Fourier Transform

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

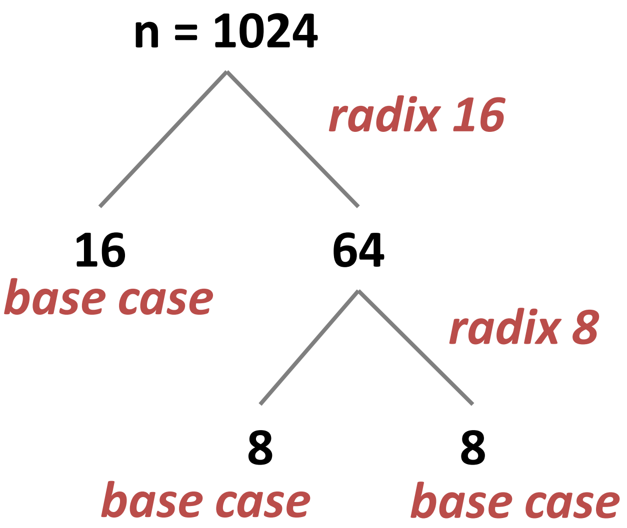
```
configure/make
```

Usage

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

Twiddles

Search for fastest computation strategy

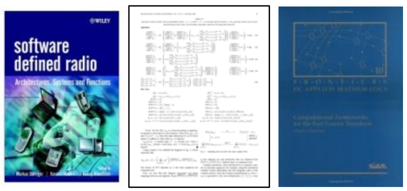


- ***Techniques:***
  - (Online) Adaptive library
  - Dynamic programming search
  - Not explained: Program generator for basic blocks

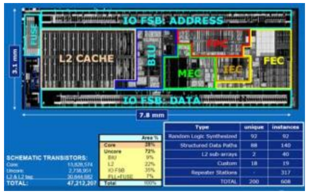
Spiral: Linear Transforms & More

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) = …
…

Spiral

Optimized implementation

(regenerated for every new platform)

Spiral: Linear Transforms & More

Algorithm knowledge

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

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

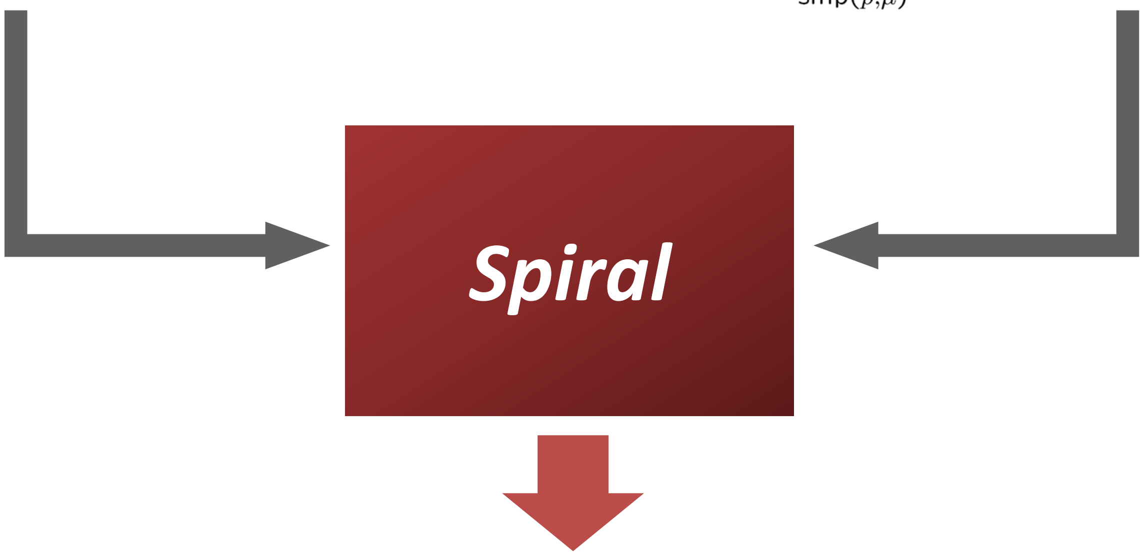
$$\mathbf{RDFT\text{-}3}_n \to (Q_{k/2,m}^{\top} \otimes I_2)\,(I_k \otimes_i \mathbf{rDFT}_{2m})(i+1/2)/k))\,(\mathbf{RDFT\text{-}3}_k \otimes I_m)$$

Platform description

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

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

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



Optimized implementation

(regenerated for every new platform)

Program Generation in Spiral (Sketched)

Transform
user specified

$$\mathrm{DFT}_8$$

Algorithm rules

Fast algorithm in SPL
many choices

$$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, \left(I_2 \otimes \left((\mathrm{DFT}_2 \otimes I_2) \cdot T^4_2\, (I_2 \otimes \mathrm{DFT}_2)\, L^4_2\right)\right) L^8_2$$

∑-SPL

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

Optimized implementation

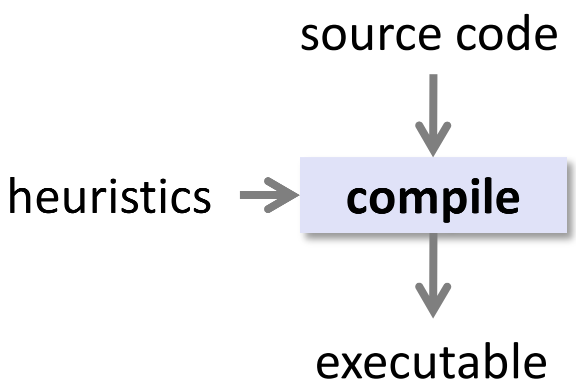
Optimization at all abstraction levels

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

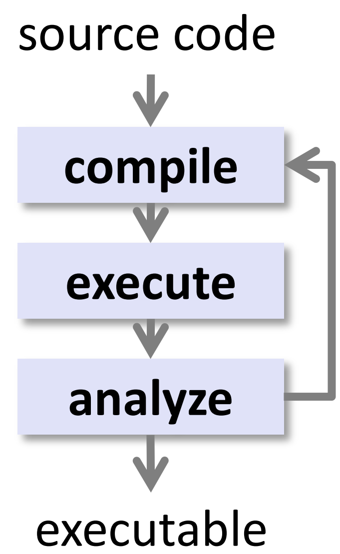
- ***Techniques:***
  - Domain-specific language (declarative, mathematical, point-free)
  - Rewriting for optimization
  - Search techniques
  - …

Adaptive Compilation

Traditional:



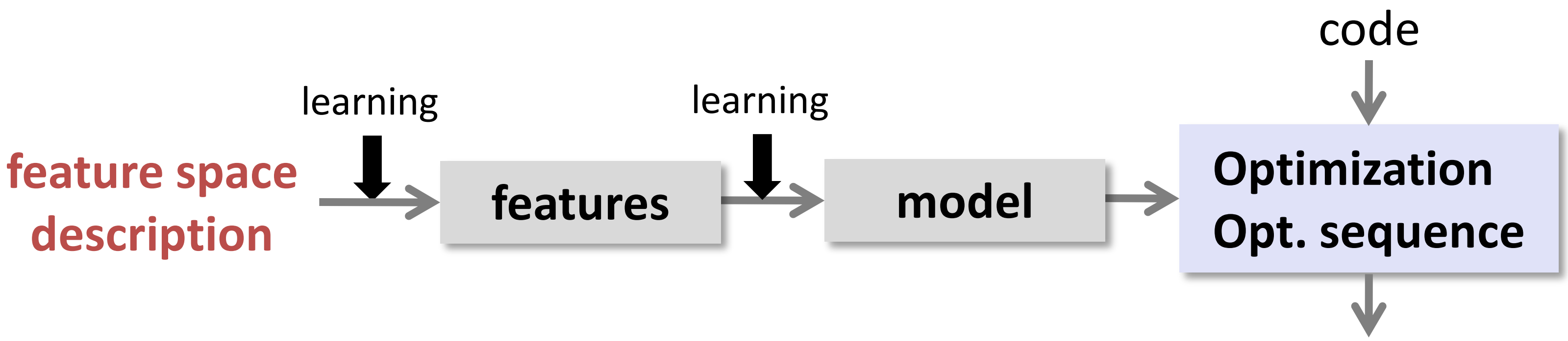
Iterative:



- Optimization ordering: Cooper et al., Kulkami et al., Triantifyllis et al.,
- Parameter selection: Kisuki et al., Stephenson et al.
- ***Techniques:***
  - Feedback-driven search based on measurements

Adaptive Compilation

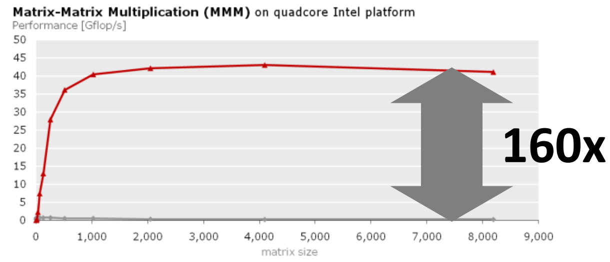
Moss, Boyle, Cavazos, Stephenson, Beaty, …



- ***Techniques:***
  - Machine learning

Autotuning: Conclusions

- Is necessary



- Search over space of parameters

Need for a broader set of techniques:

- Program generation
- Domain-specific languages (declarative)
- Models
- Machine learning
- Data structures