Automatic Performance Tuning

Jeremy Johnson

Dept. of Computer Science

Drexel University

Outline

- Scientific Computation Kernels
  - Matrix Multiplication
  - Fast Fourier Transform (FFT)

- Automated Performance Tuning
  (IEEE Proc. Vol. 93, No. 2, Feb. 2005)
  - ATLAS
  - FFTW
  - SPIRAL

Matrix Multiplication and the FFT

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

$$y_k = \sum_{l=0}^{N-1} \omega_N^{kl} x_l$$

$$N = RS, k = k_2 S + k_1, l = l_1 R + l_2$$

$$y_{k_2 S + k_1} = \sum_{l=0}^{S-1} \omega_S^{k1l1} \omega_N^{kl} \left(\sum_{l=0}^{R-1} \omega_R^{k_2 l_2} x_{k_2 S + k_1} \right)$$

Basic Linear Algebra Subprograms (BLAS)

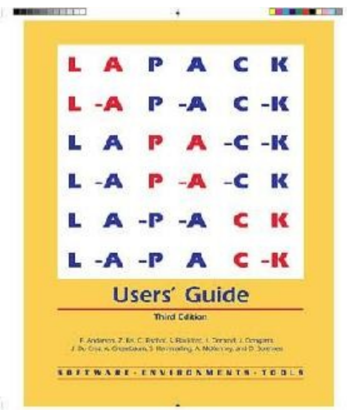
- **Level 1 – vector-vector, O(n) data, O(n) operations**
- **Level 2 – matrix-vector, $O(n^2)$ data, $O(n^2)$ operations**
- **Level 3 – matrix-matrix, $O(n^2)$ data, $O(n^3)$ operations = data reuse = locality!**

- **LAPACK built on top of BLAS (level 3)**
  - **Blocking (for the memory hierarchy) is the single most important optimization for linear algebra algorithms**

- **GEMM – General Matrix Multiplication**

  - **SUBROUTINE DGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)**

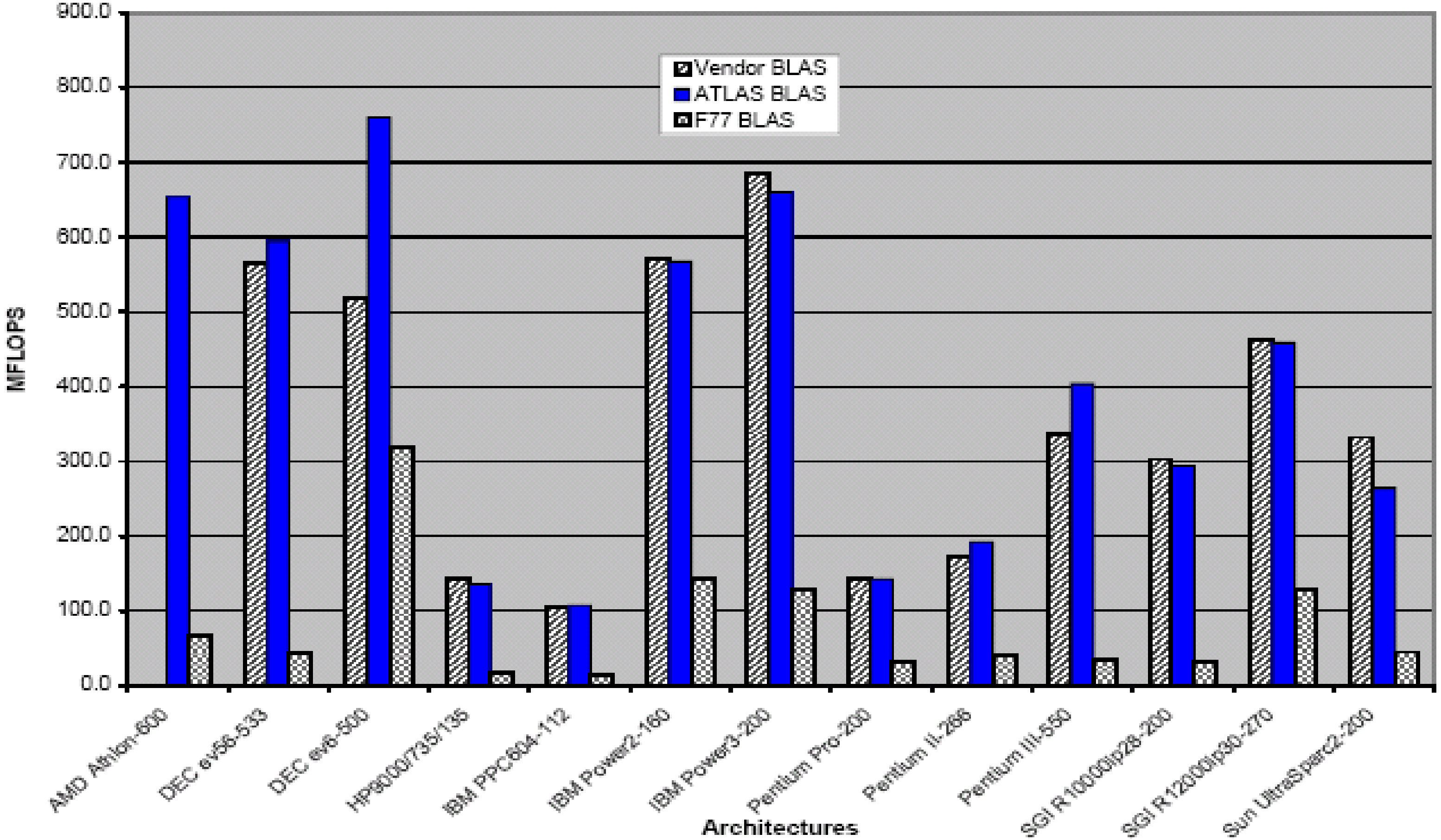
  - **C := alpha*op(A)*op(B) + beta*C,**
  - **where op(X) = X or X'**



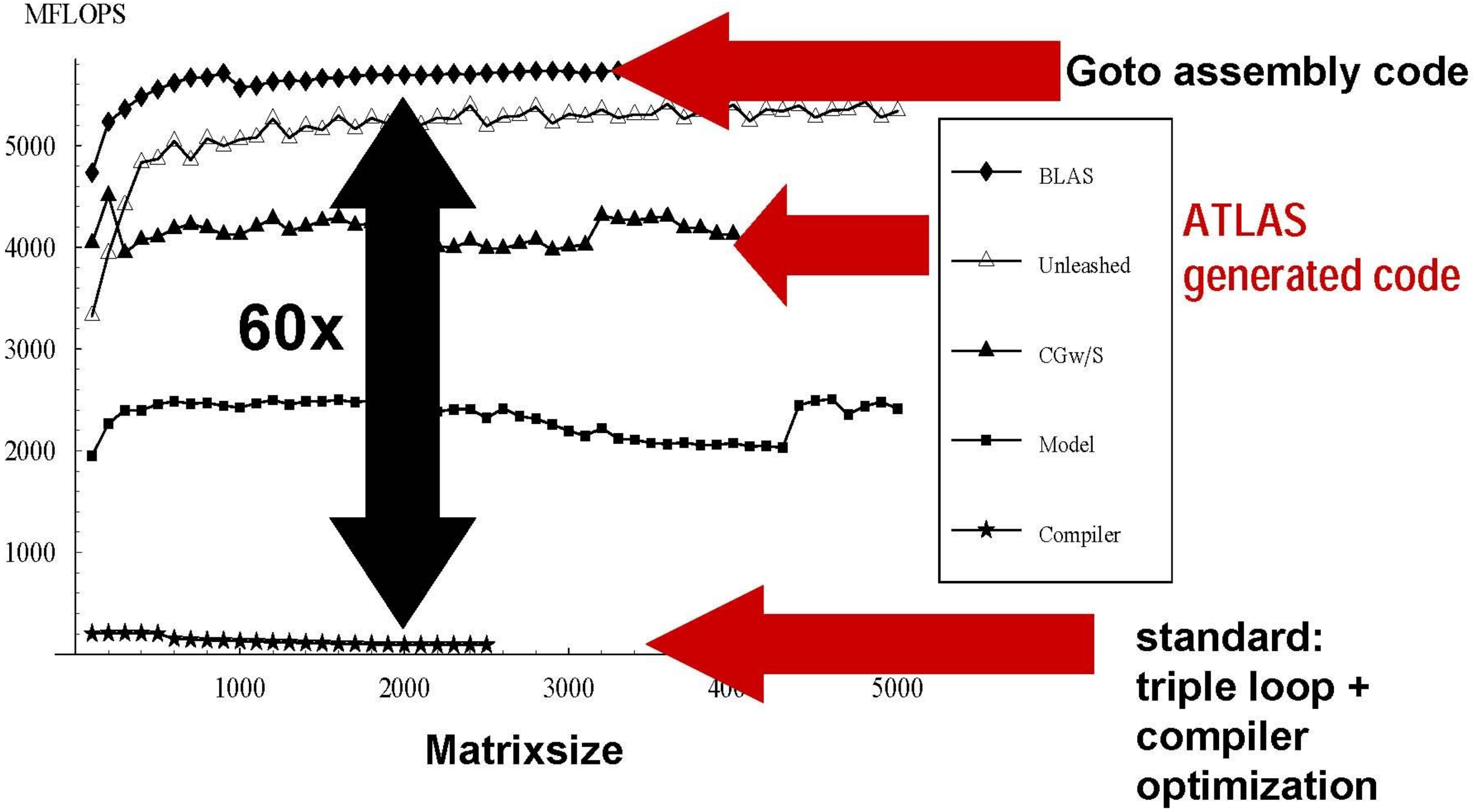
DGEMM

```
…
*          Form  C := alpha*A*B + beta*C.
*
           DO 90, J = 1, N
              IF( BETA.EQ.ZERO )THEN
                 DO 50, I = 1, M
                    C( I, J ) = ZERO
   50            CONTINUE
              ELSE IF( BETA.NE.ONE )THEN
                 DO 60, I = 1, M
                    C( I, J ) = BETA*C( I, J )
   60            CONTINUE
              END IF
              DO 80, L = 1, K
                 IF( B( L, J ).NE.ZERO )THEN
                    TEMP = ALPHA*B( L, J )
                    DO 70, I = 1, M
                       C( I, J ) = C( I, J ) + TEMP*A( I, L )
   70               CONTINUE
                 END IF
   80         CONTINUE
   90      CONTINUE
…
```

Matrix Multiplication Performance



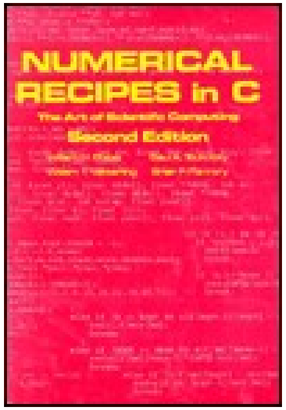
Matrix Multiplication Performance



Numeric Recipes

- **Numeric Recipes in C – The Art of Scientific Computing, 2nd Ed.**
  - *William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Cambridge University Press, 1992.*

- “This book is unique, we think, in offering, for each topic considered, a certain amount of general discussion, a certain amount of analytical mathematics, a certain amount of discussion of algorithmics, and (most important) actual implementations of these ideas in the form of working computer routines.
- 1. Preliminarys
- 2. Solutions of Linear Algebraic Equations
- …
- 12. Fast Fourier Transform
- 19. Partial Differential Equations
- 20. Less Numerical Algorithms



four1

```
#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(float data[], unsigned long nn, int isign)
```

Replaces `data[1..2*nn]` by its discrete Fourier transform, if `isign` is input as 1; or replaces `data[1..2*nn]` by `nn` times its inverse discrete Fourier transform, if `isign` is input as −1. `data` is a complex array of length `nn` or, equivalently, a real array of length `2*nn`. `nn` MUST be an integer power of 2 (this is not checked for!).

```
{
    unsigned long n,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;     Double precision for the trigonomet-
    float tempr,tempi;                        ric recurrences.

    n=nn << 1;
    j=1;
    for (i=1;i<n;i+=2) {                  This is the bit-reversal section of the
        if (j > i) {                          routine.
            SWAP(data[j],data[i]);        Exchange the two complex numbers.
            SWAP(data[j+1],data[i+1]);
        }
        m=nn;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }
```

four1 (cont)

```
Here begins the Danielson-Lanczos section of the routine.
mmax=2;
while (n > mmax) {                               Outer loop executed log2 nn times.
    istep=mmax << 1;
    theta=isign*(6.28318530717959/mmax);         Initialize the trigonometric recurrence.
    wtemp=sin(0.5*theta);
    wpr = -2.0*wtemp*wtemp;
    wpi=sin(theta);
    wr=1.0;
    wi=0.0;
    for (m=1;m<mmax;m+=2) {                      Here are the two nested inner loops.
        for (i=m;i<=n;i+=istep) {
            j=i+mmax;                            This is the Danielson-Lanczos for-
            tempr=wr*data[j]-wi*data[j+1];           mula:
            tempi=wr*data[j+1]+wi*data[j];
            data[j]=data[i]-tempr;
            data[j+1]=data[i+1]-tempi;
            data[i] += tempr;
            data[i+1] += tempi;
        }
        wr=(wtemp=wr)*wpr-wi*wpi+wr;             Trigonometric recurrence.
        wi=wi*wpr+wtemp*wpi+wi;
    }
    mmax=istep;
}
}
```

FFT Performance

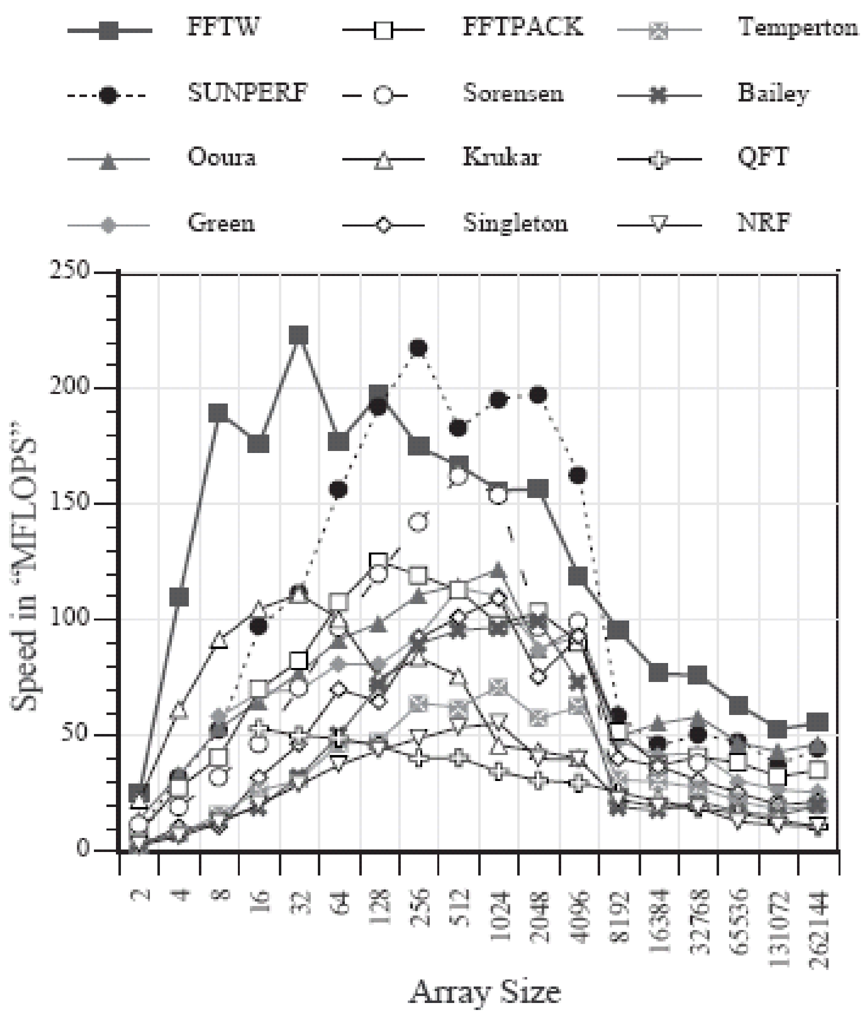
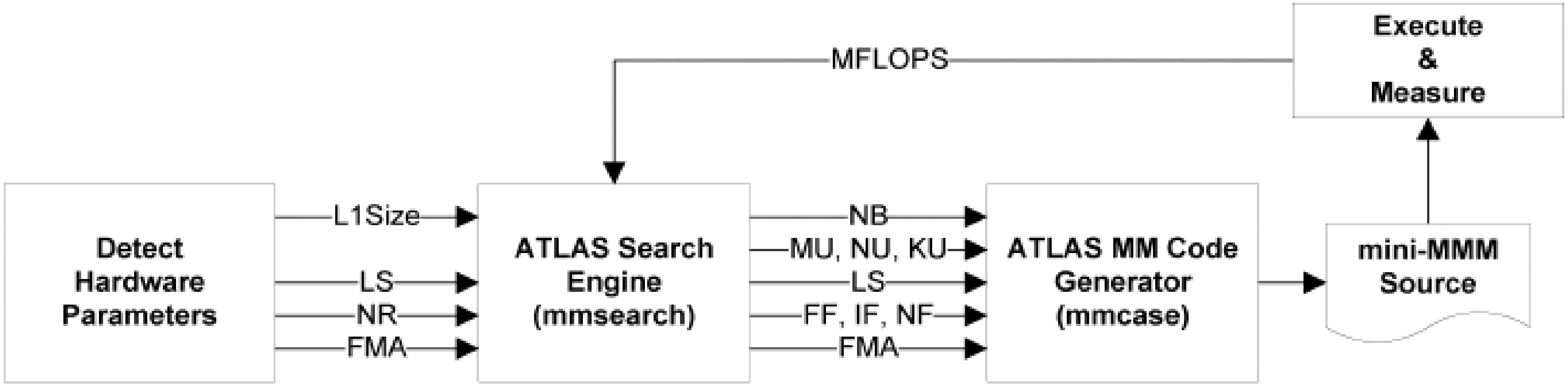


Figure 4: Comparison of double precision 1D complex FFTs on a Sun HPC 5000 (167MHz UltraSPARC-I). Compiled with `[cc|f77] -native -fast -xO5 -dalign`. SunOS 5.5.1, Sun WorkShop Compilers version 4.2.

Atlas Architecture and Search Parameters



- N_B – L1 data cache tile size
- NCN_B – L1 data cache tile size for non-copying version
- M_U, N_U – Register tile size
- K_U – Unroll factor for k' loop
- L_S – Latency for computation scheduling
- FMA – 1 if fused multiply-add available, 0 otherwise
- F_F, I_F, N_F – Scheduling of loads

Yotov et al., Is Search Really Necessary to Generate High-Performance BLAS?, Proc. IEEE, Vol. 93, No. 2, Feb. 2005

ATLAS Code Generation

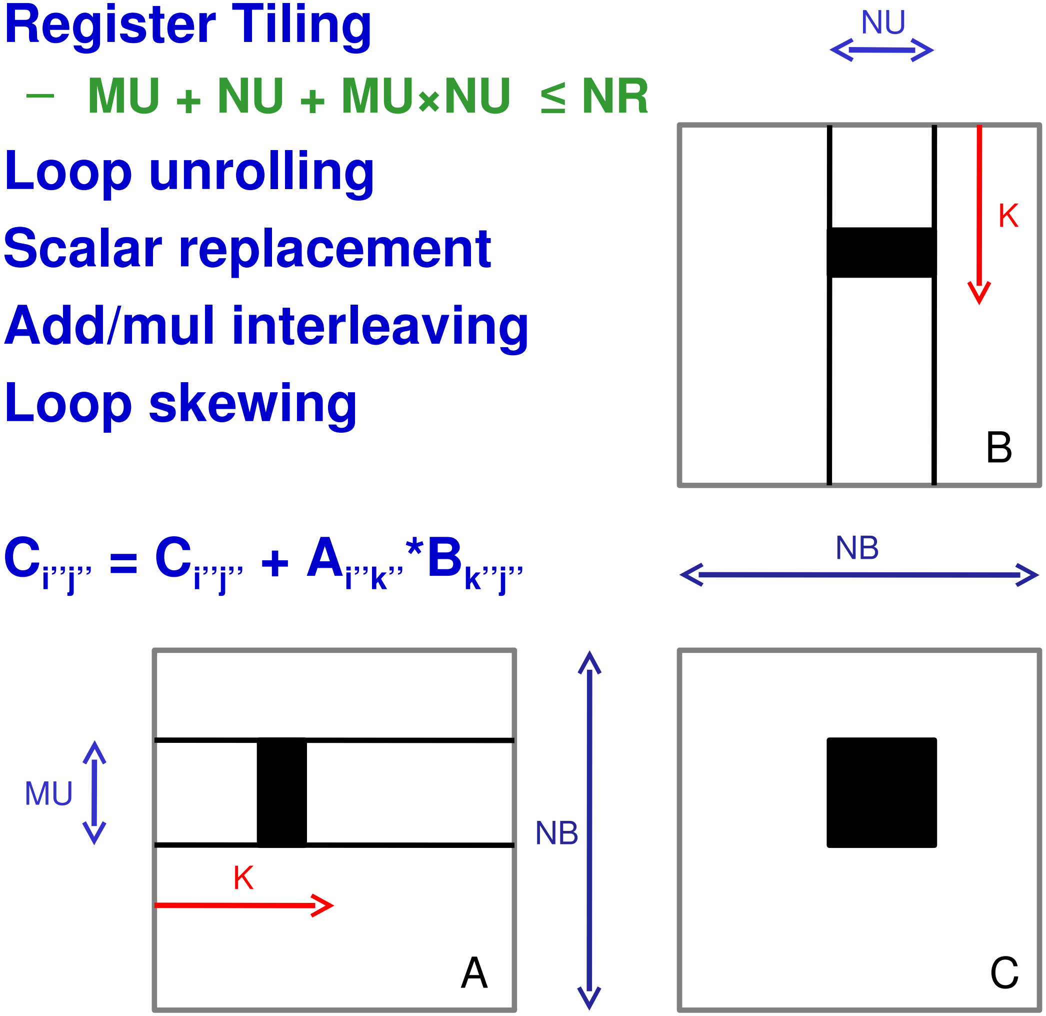
- **Optimization for locality**
  - **Cache tiling, Register tiling**

// MMM loop nest (j, i, k)
// copy full A here
for $j \in [1 : N_B : M]$
    // copy a panel of B here
    for $i \in [1 : N_B : N]$
        // possibly copy a tile of C here
        for $k \in [1 : N_B : K]$
            // mini-MMM loop nest (j', i', k')
            for $j' \in [j : N_U : j + N_B - 1]$
            for $i' \in [i : M_U : i + N_B - 1]$
            for $k' \in [k : K_U : k + N_B - 1]$
                for $k'' \in [k' : 1 : k' + K_U - 1]$
                // micro-MMM loop nest (j'', i'')
                for $j'' \in [j' : 1 : j' + N_U - 1]$
                for $i'' \in [i' : 1 : i' + M_U - 1]$
                    $C_{i''j''} = C_{i''j''} + A_{i''k''} \times B_{k''j''}$

ATLAS Code Generation

- **Register Tiling**
  - **$MU + NU + MU \times NU \leq NR$**
- **Loop unrolling**
- **Scalar replacement**
- **Add/mul interleaving**
- **Loop skewing**

- $C_{i''j''} = C_{i''j''} + A_{i''k''} * B_{k''j''}$



mul_1

mul_2

…

mul_{Ls}

add_1

mul_{Ls+1}

add_2

…

$mul_{Mu \times Nu}$

$add_{Mu \times Nu-Ls+2}$

…

$add_{Mu \times Nu}$

ATLAS Search

- Estimate Machine Parameters (C_1, N_R, FMA, L_S)
  - Used to bound search
- Orthogonal Line Search (fix all parameters except one and search for the optimal value of this parameter)
  - Search order
    - NB
    - MU, NU
    - KU
    - LS
    - FF, IF, NF
    - NCNB
    - Cleanup codes

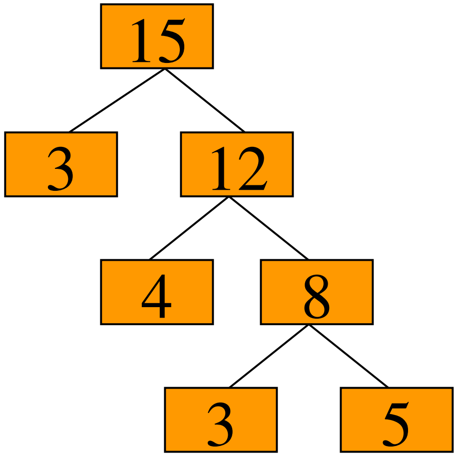
Using FFTW

```
fftw_plan plan;
int n = 1024;
COMPLEX A[n], B[n];

/* plan the computation */
plan = fftw_create_plan(n);

/* execute the plan */
fftw(plan, A);

/* the plan can be reused */
fftw(plan, B);
```

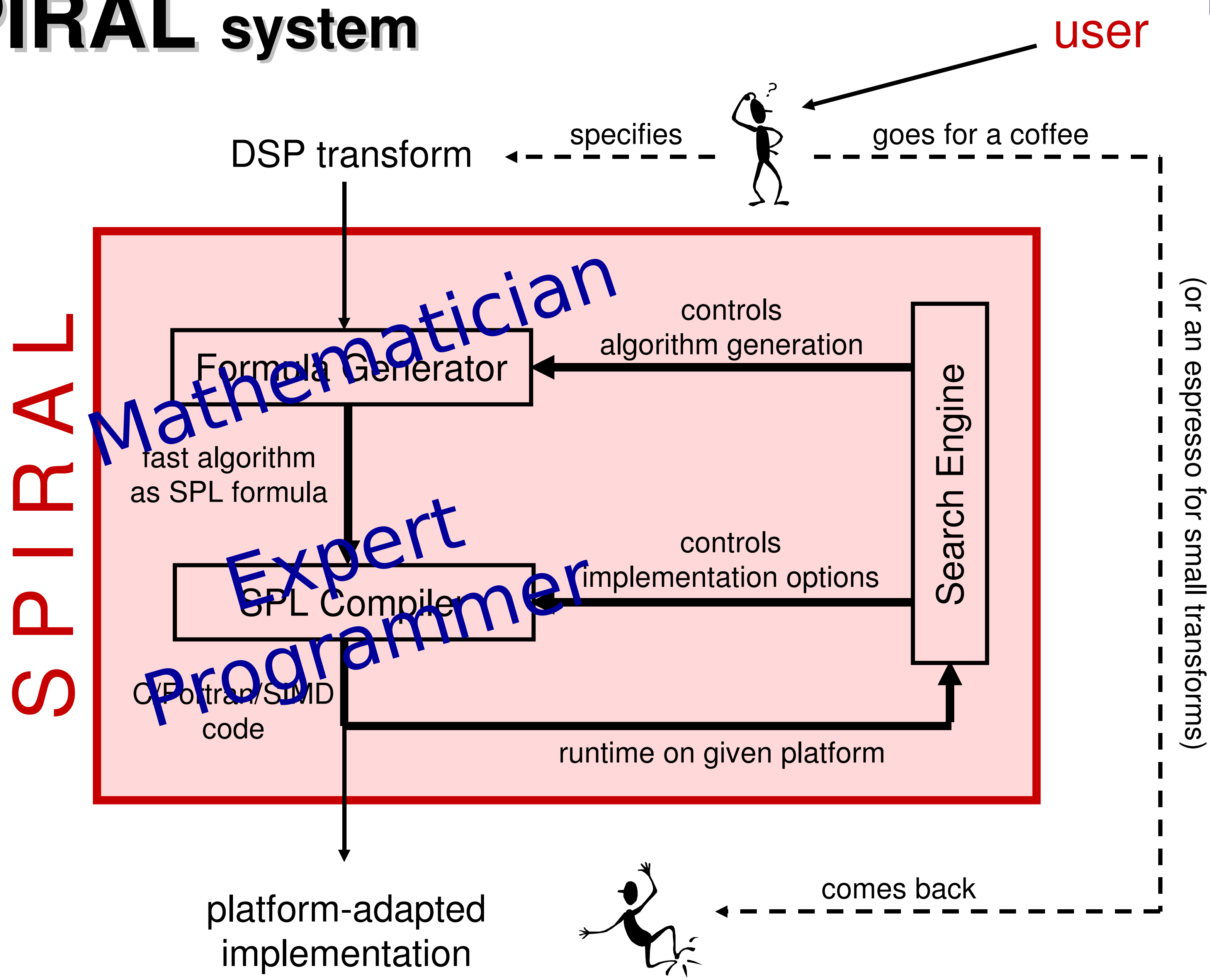
FFTW Infrastructure

- Use dynamic programming to find an efficient way to combine code sequences.
- Combine code sequences using divide and conquer structure in FFT
- Codelets (optimized code sequences for small FFTs)
- Plan encodes divide and conquer strategy and stores "twiddle factors"
- Executor computes FFT of given data using algorithm described by plan.

Right Recursive



SPIRAL system



DSP Algorithms: Example 4-point DFT

Cooley/Tukey FFT (size 4):

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fourier transform

Diagonal matrix (twiddles)

$$DFT_4 = (DFT_2 \otimes I_2) \cdot T_2^4 \cdot (I_2 \otimes DFT_2) \cdot L_2^4$$

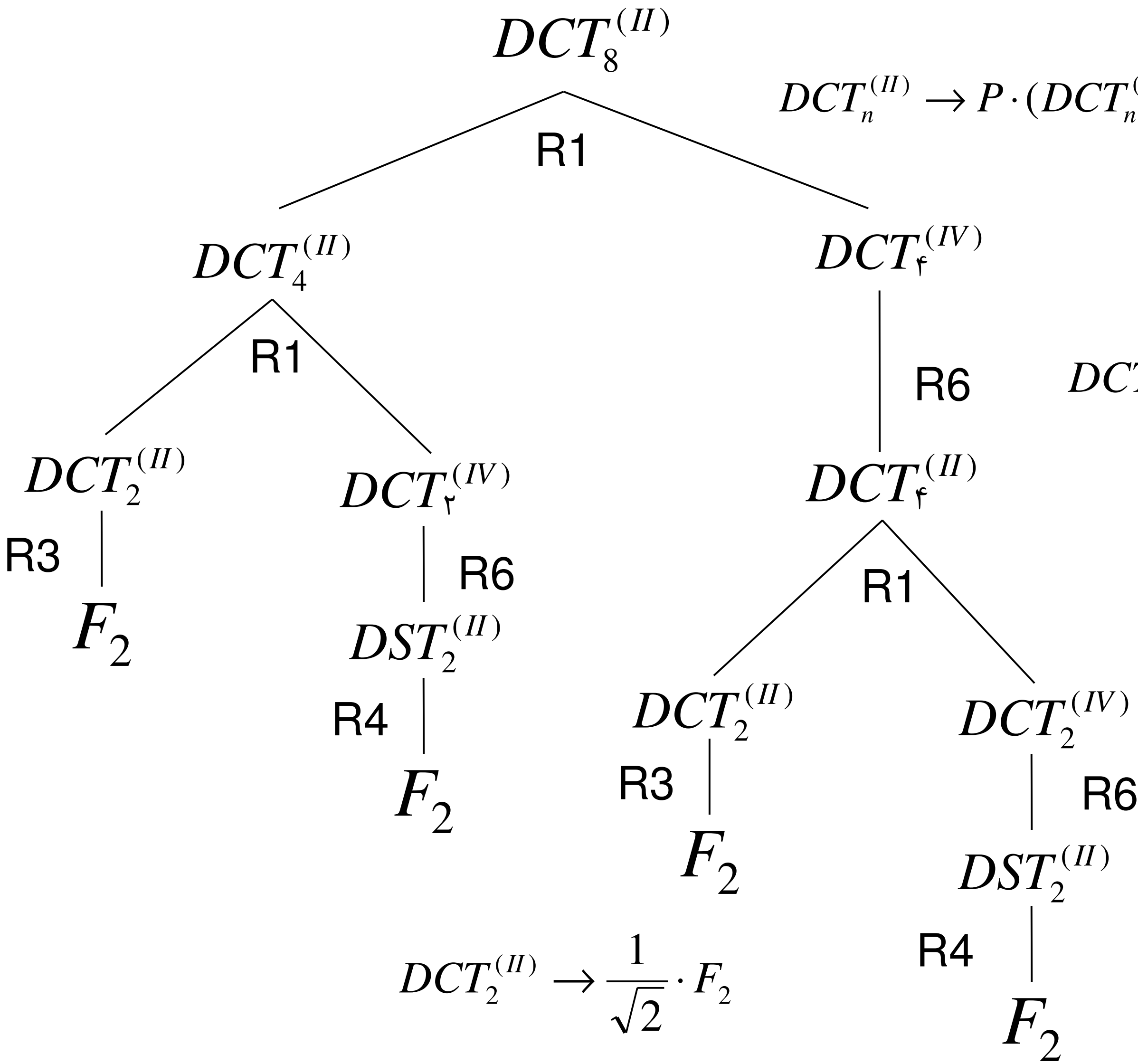
Kronecker product

Identity

Permutation

- algorithms reduce arithmetic cost O(n^2)→O(nlog(n))
- product of structured sparse matrices
- mathematical notation exhibits structure

Algorithms = Ruletrees = Formulas

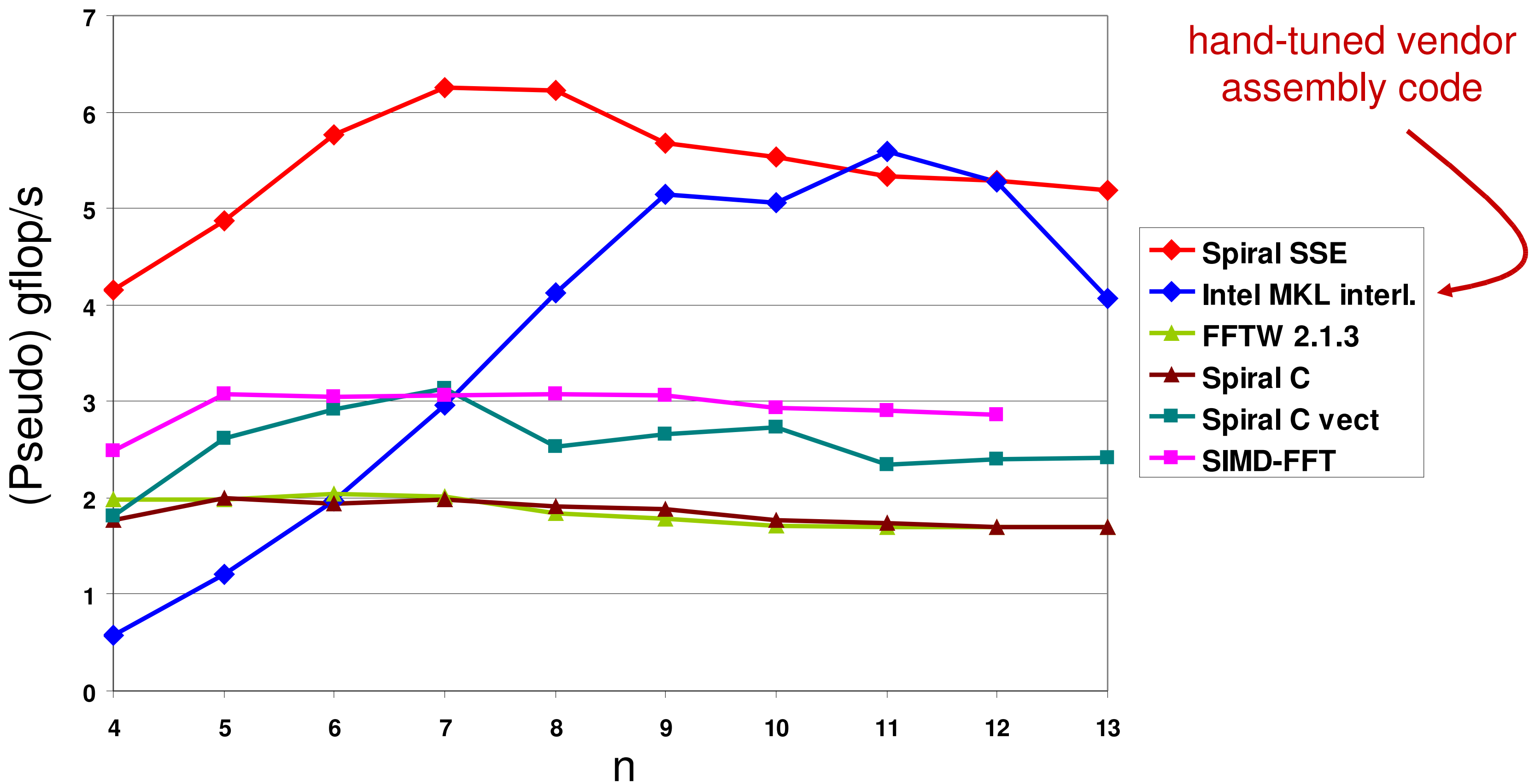


$$DCT_n^{(II)} \rightarrow P \cdot (DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)}) \cdot (F_2 \otimes I_{n/2})$$

$$DCT_n^{(IV)} \rightarrow P \cdot DCT_n^{(II)} \cdot S$$

$$DCT_2^{(II)} \rightarrow \frac{1}{\sqrt{2}} \cdot F_2$$

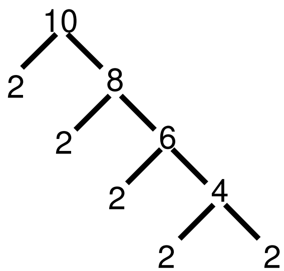
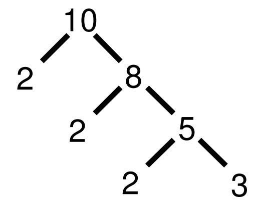
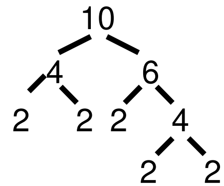
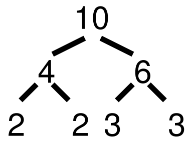
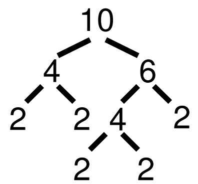
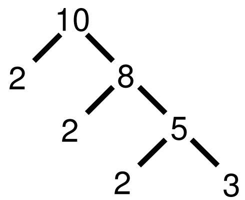
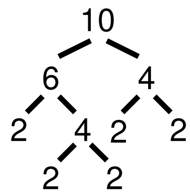
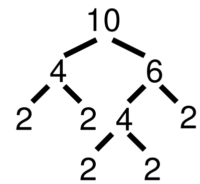
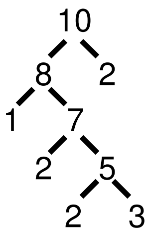
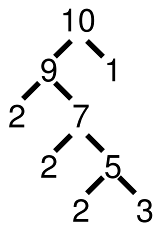
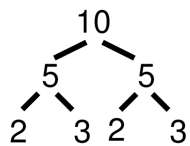
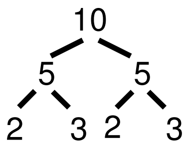
Generated DFT Vector Code: Pentium 4, SSE



DFT 2^n *single precision*, Pentium 4, 2.53 GHz, using Intel C compiler 6.0

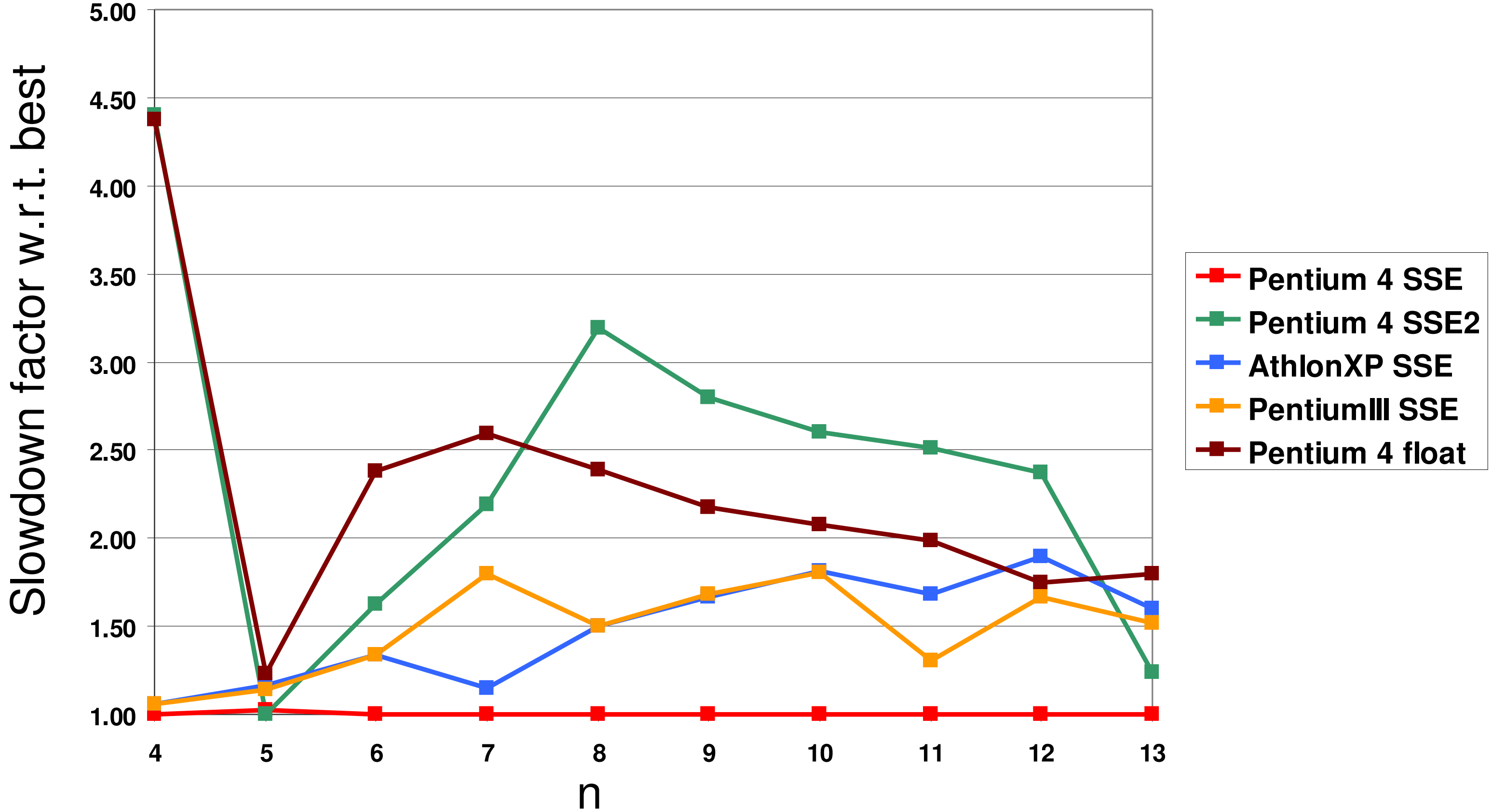
➡ speedups (to C code) up to factor of 3.1

Best DFT Trees, size 2^{10} = 1024

	Pentium 4 float	Pentium 4 double	Pentium III float	AthlonXP float
scalar				
C vect				
SIMD				

→ trees platform/datatype dependent

Crosstiming of best trees on Pentium 4



DFT 2^n *single precision*, runtime of best found of other platforms

➡ software adaptation is necessary