Autotuning (2.5/2): TCE & Empirical compilers

Prof. Richard Vuduc

Georgia Institute of Technology

CSE/CS 8803 PNA: Parallel Numerical Algorithms

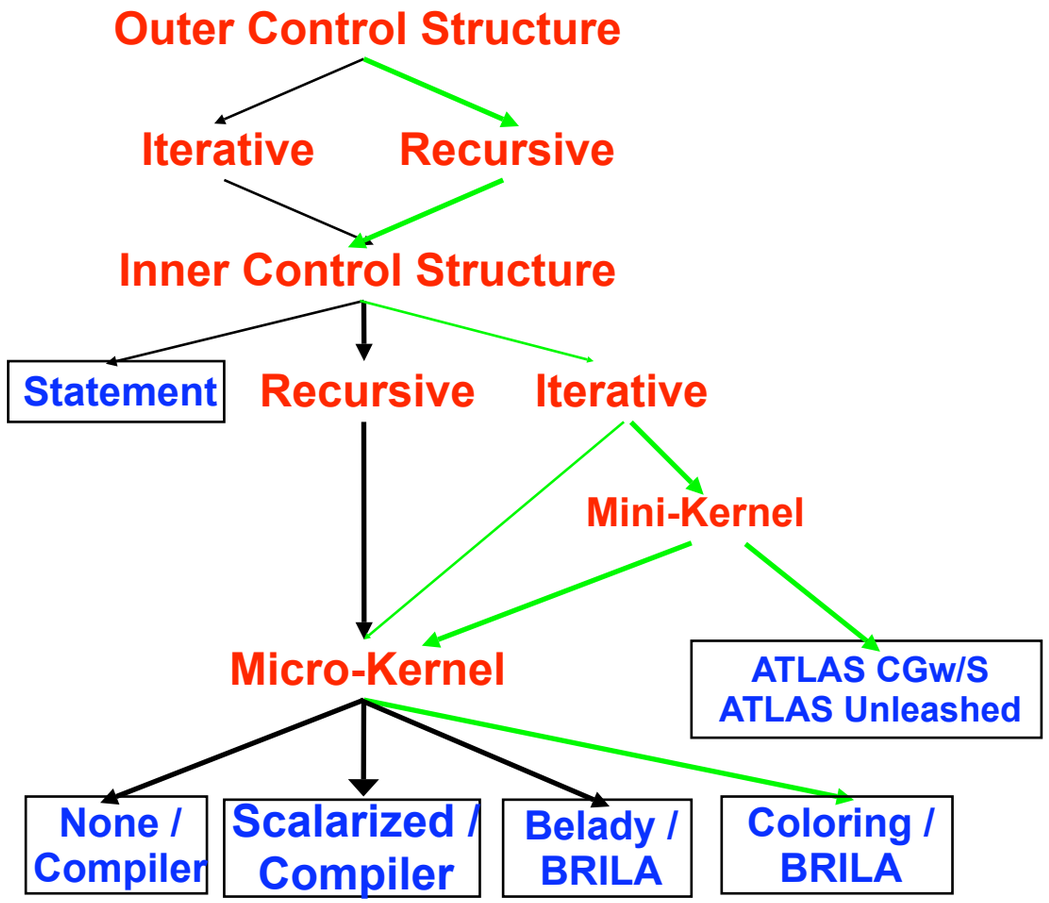
[L.19] Tuesday, March 11, 2008

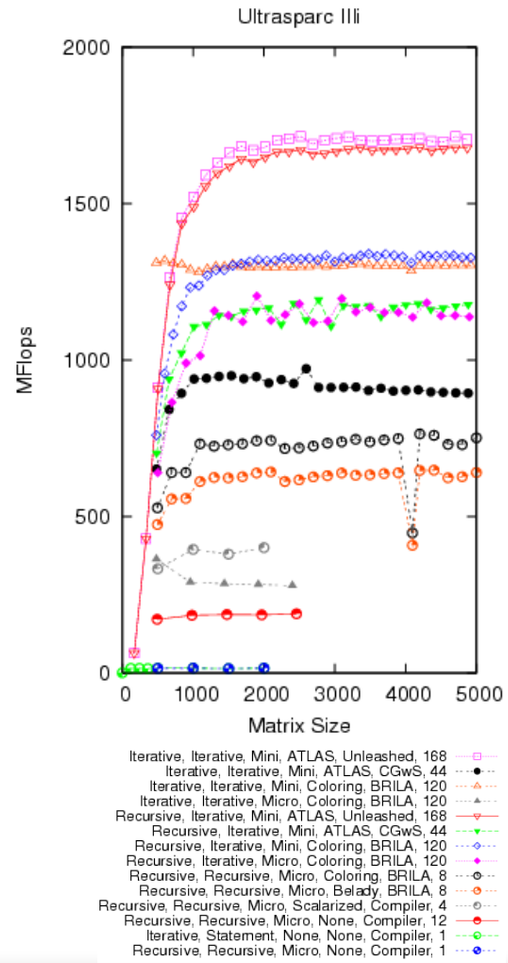
Today's sources

- CS 267 at UCB (Demmel & Yelick)
- Papers from various autotuning projects
  - PHiPAC, ATLAS, FFTW, SPIRAL, TCE
    - See: Proc. IEEE 2005 special issue on Program Generation, Optimization, and Platform Adaptation
- Me (for once!)

Review:
Autotuners

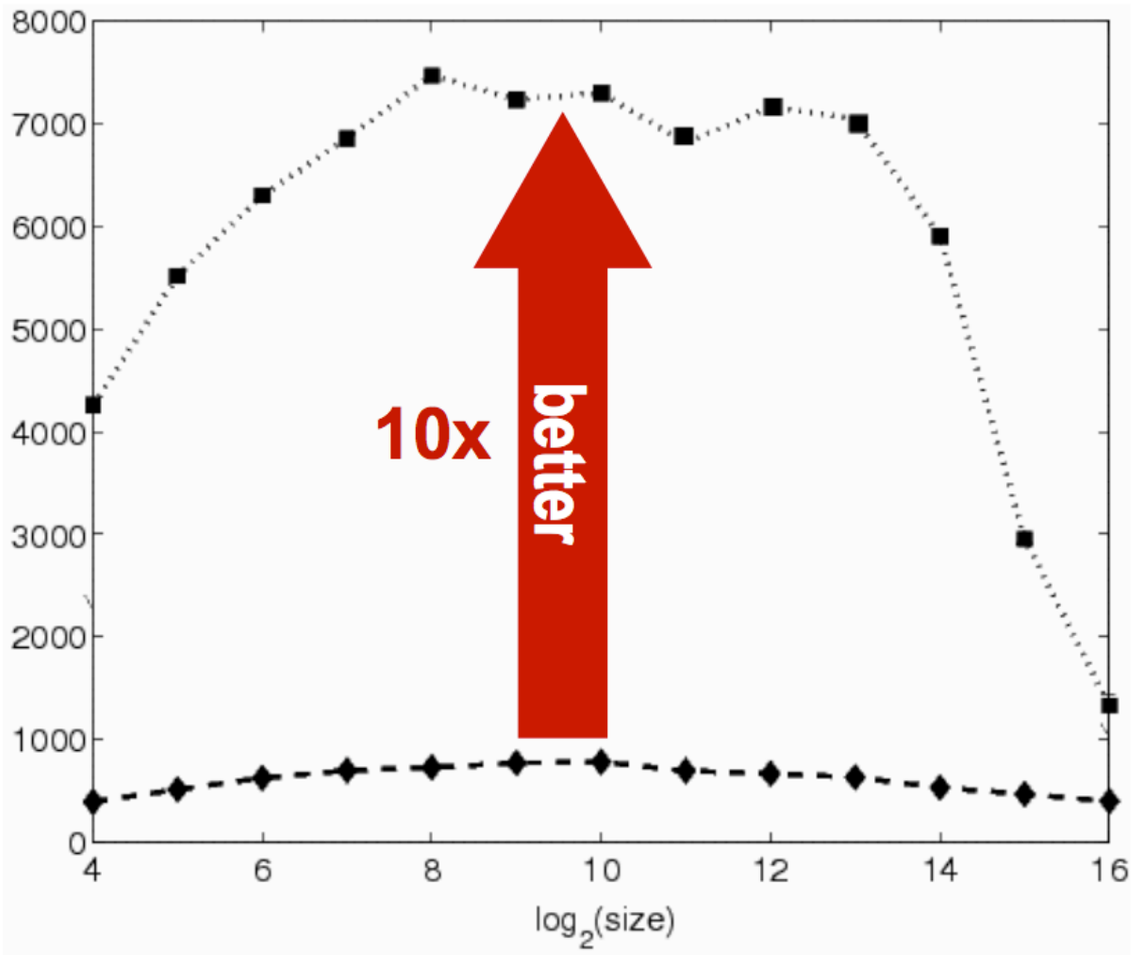
Performance-engineering challenges





Motivation for performance tuning

pseudo
Mflop/s



best available implementation (FFTW, Intel IPP, Spiral)

roughly the same operations count

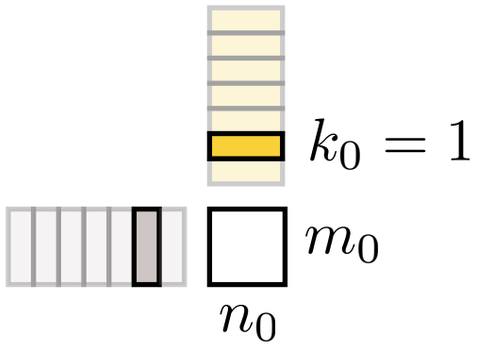
reasonable implementation (Numerical recipes, GNU scientific library)

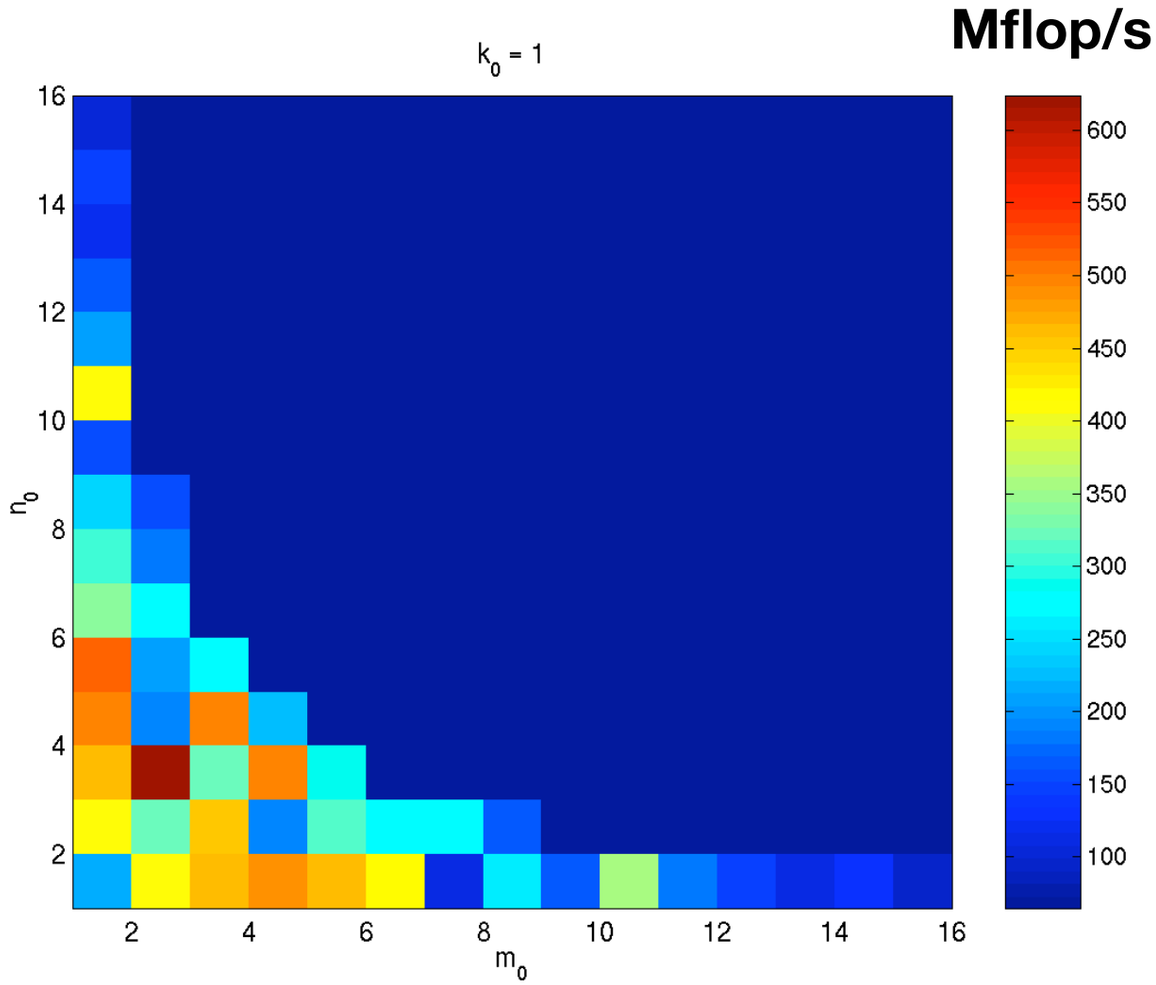
Source: J. Johnson (2007), CScADS autotuning workshop

Context for autotuning

- Problem: HPC needs detailed low-level machine knowledge
- Autotuning methodology
  - Identify and generate a space of implementations
  - Search (modeling, experiments) to choose the best one
- Early idea seedlings
  - Polyalgorithms
  - Profile and feedback-directed compilation
  - Domain- and architecture-specific code generators

Example: What a search space looks like





- Platform: Sun Ultra IIi
  - 16 double regs
  - 667 Mflop/s peak
  - Unrolled, pipelined inner-kernel
  - Sun cc v5.0 compiler

Source: PHiPAC Project at UC Berkeley (1997)

Cooley-Tukey FFT algorithm: Encoding in FFTW's codelet generator

$$y[k] \quad \leftarrow \quad \mathrm{DFT}_N(x,k) \equiv \sum_{j=0}^{N-1} x[j] \cdot \omega_N^{-kj} \qquad\qquad x, y \in \mathbb{C}^N$$

$$y[k_1 + k_2 \cdot N_1] \quad \leftarrow \quad \sum_{n_2=0}^{N_2-1} \left[\left(\sum_{n_1}^{N_1-1} x[n_1 \cdot N_2 + n_2] \cdot \omega_{N_1}^{-k_1 n_1} \right) \cdot \omega_N^{-k_1 n_2} \right] \cdot \omega_{N_2}^{-k_2 n_2}$$

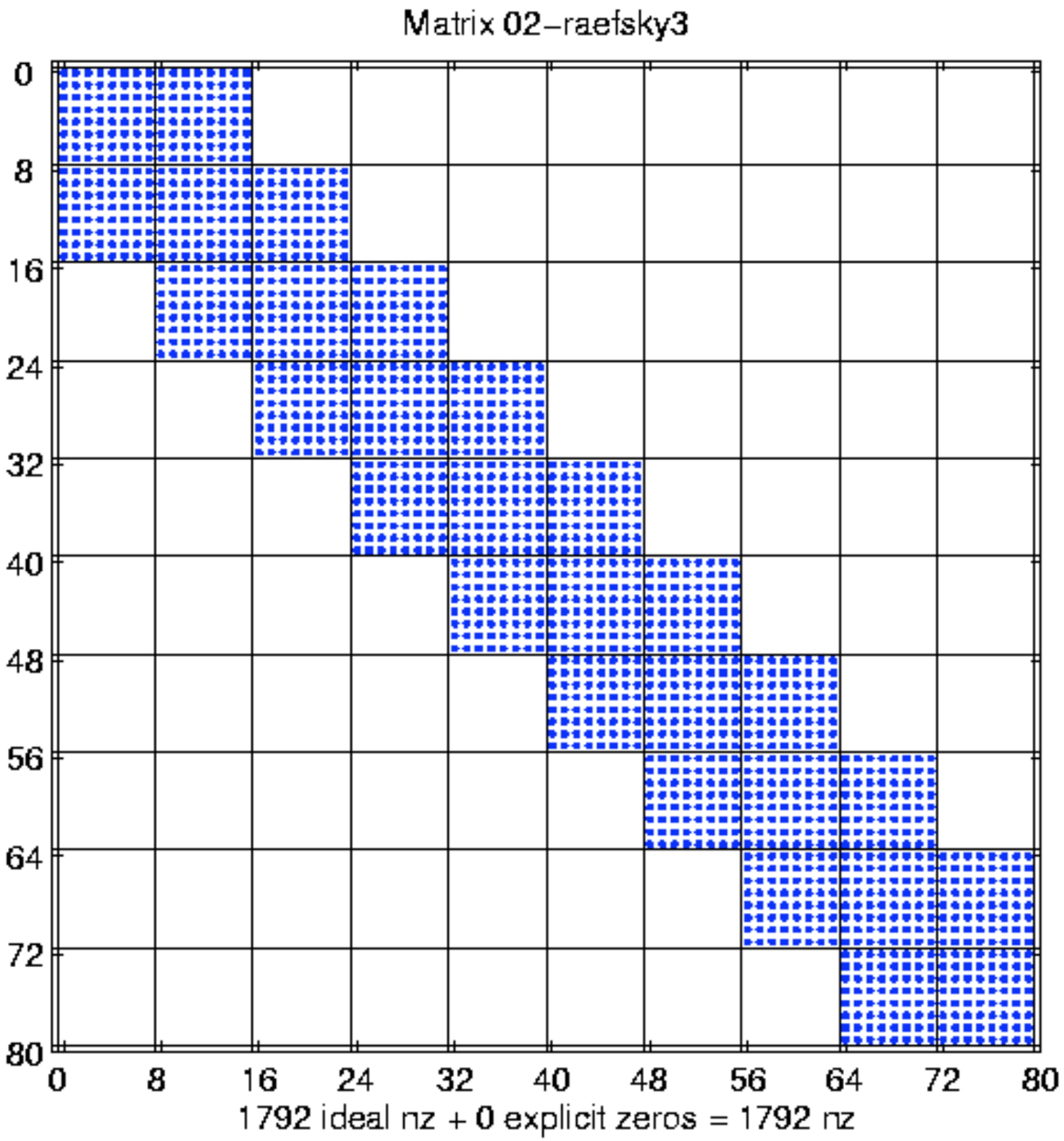
N_1-point DFT

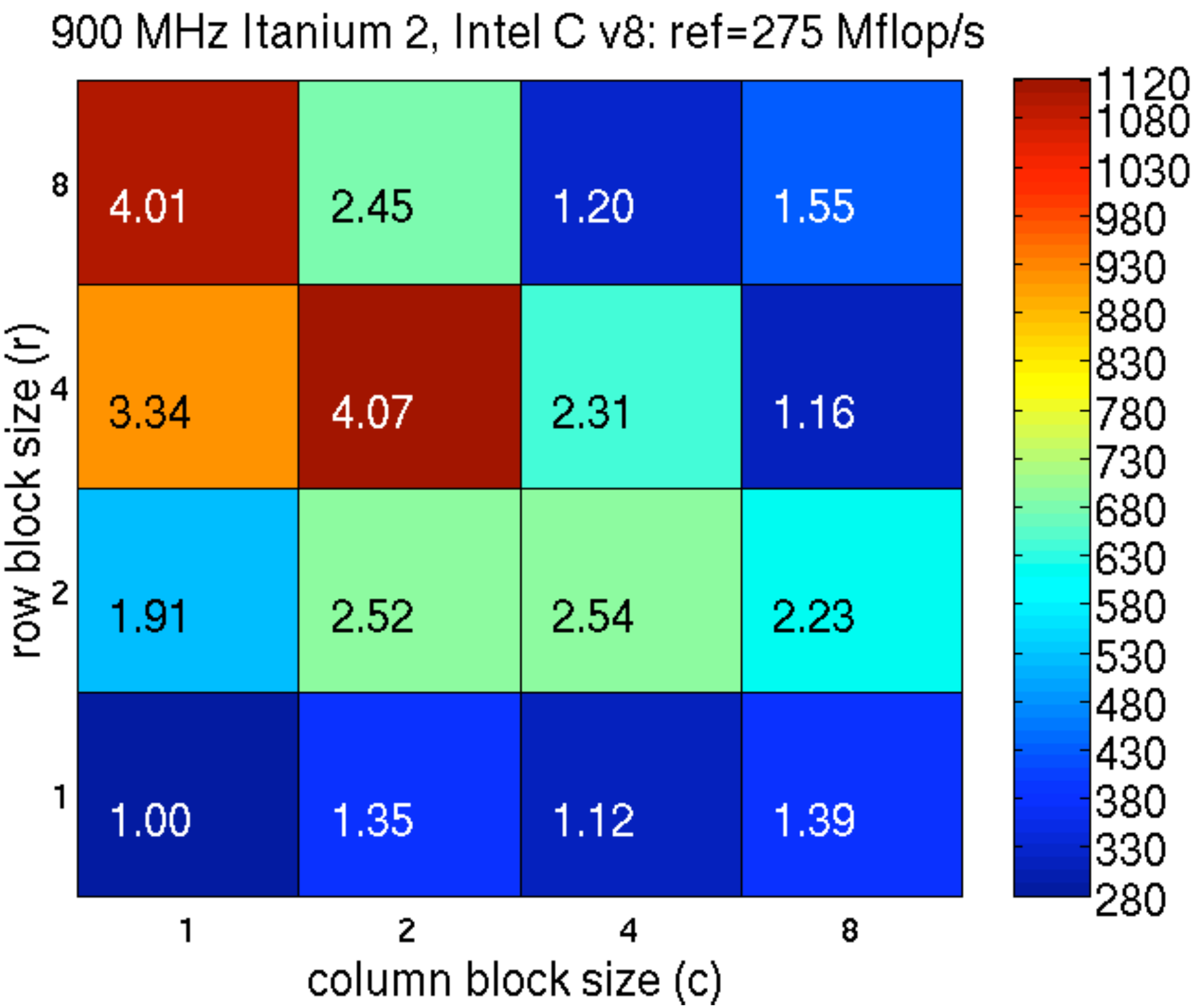
Twiddle

N_2-point DFT

```
let dftgen(N, x) ≡  fun k → ...    # DFT_N(x, k)
let cooley_tukey(N1, N2, x) ≡
  let x̂ ≡ fun n2, n1 → x(n2 + n1 · N2) in
  let G1 ≡  fun n2 → dftgen(N1, x̂(n2, __)) in
  let W ≡  fun k1, n2 → G1(n2, k1) · ω_N^(−k1 n2)  in
  let G2 ≡  fun k1 → dftgen(N2, W(k1, __))
  in
    fun k → G2(k mod N1, k div N1)
```

(Functional pseudo-code)


Matrix 02–raefsky3
0
8
16
24
32
40
48
56
64
72
80
0
8
16
24
32
40
48
56
64
72
80
1792 ideal nz + 0 explicit zeros = 1792 nz


900 MHz Itanium 2, Intel C v8: ref=275 Mflop/s
row block size (r)
column block size (c)
8
4
2
1
1
2
4
8
4.01
2.45
1.20
1.55
3.34
4.07
2.31
1.16
1.91
2.52
2.54
2.23
1.00
1.35
1.12
1.39
1120
1080
1030
980
930
880
830
780
730
680
630
580
530
480
430
380
330
280

Tensor Contraction Engine (TCE) for quantum chemistry

Tensor Contraction Engine (TCE)

- Application domain: Quantum chemistry
  - Electronic structure calculations
  - Dominant computation expressible as a “tensor contraction”
- TCE generates a complete parallel program from a high-level spec
  - Automates time-space trade-offs
  - Output
- S. Hirata (2002), and many others
- Following presentation taken from Proc. IEEE 2005 special issue

Motivation: Simplify program development

hbar[a,b,i,j] == sum[f[b,c] * t[i,j,a,c], c] - sum[f[k,c] * t[k,b] * t[i,j,a,c], k,c] + sum[f[a,c] * t[i,j,c,b], c] - sum[f[k,c] * t[k,a] * t[i,j,c,b], k,c] - sum[f[k,j] * t[i,k,a,b], k] - sum[f[k,c] * t[j,c] * t[i,k,a,b], k,c] - sum[f[k,i] * t[j,k,b,a], k] - sum[f[k,c] * t[i,c] * t[j,k,b,a], k,c] + sum[t[i,c] * t[j,d] * v[a,b,c,d], c,d] + sum[t[i,j,c,d] * v[a,b,c,d], c,d] + sum[t[j,c] * v[a,b,i,c], c] - sum[t[k,b] * v[a,k,i,j], k] + sum[t[i,c] * v[b,a,j,c], c] - sum[t[k,a] * v[b,k,j,i], k] - sum[t[k,d] * t[i,j,c,b] * v[k,a,c,d], k,c,d] - sum[t[i,c] * t[j,k,b,d] * v[k,a,c,d], k,c,d] - sum[t[j,c] * t[k,b] * v[k,a,c,i], k,c] + 2 * sum[t[j,k,b,c] * v[k,a,c,i], k,c] - sum[t[j,k,c,b] * v[k,a,c,i], k,c] - sum[t[i,c] * t[j,d] * t[k,b] * v[k,a,d,c], k,c,d] + 2 * sum[t[k,d] * t[i,j,c,b] * v[k,a,d,c], k,c,d] - sum[t[k,b] * t[i,j,c,d] * v[k,a,d,c], k,c,d] - sum[t[j,d] * t[i,k,c,b] * v[k,a,d,c], k,c,d] + 2 * sum[t[i,c] * t[j,k,b,d] * v[k,a,d,c], k,c,d] - sum[t[i,c] * t[j,k,d,b] * v[k,a,d,c], k,c,d] - sum[t[j,k,b,c] * v[k,a,i,c], k,c] - sum[t[i,c] * t[k,b] * v[k,a,j,c], k,c] - sum[t[i,k,c,b] * v[k,a,j,c], k,c] - sum[t[i,c] * t[j,d] * t[k,a] * v[k,b,c,d], k,c,d] - sum[t[k,d] * t[i,j,a,c] * v[k,b,c,d], k,c,d] - sum[t[k,a] * t[i,j,c,d] * v[k,b,c,d], k,c,d] + 2 * sum[t[j,d] * t[i,k,a,c] * v[k,b,c,d], k,c,d] - sum[t[j,d] * t[i,k,c,a] * v[k,b,c,d], k,c,d] - sum[t[i,c] * t[j,k,d,a] * v[k,b,c,d], k,c,d] - sum[t[i,c] * t[k,a] * v[k,b,c,j], k,c] + 2 * sum[t[i,k,a,c] * v[k,b,c,j], k,c] - sum[t[i,k,c,a] * v[k,b,c,j], k,c] + 2 * sum[t[k,d] * t[i,j,a,c] * v[k,b,d,c], k,c,d] - sum[t[j,d] * t[i,k,a,c] * v[k,b,d,c], k,c,d] - sum[t[j,c] * t[k,a] * v[k,b,i,c], k,c] - sum[t[j,k,c,a] * v[k,b,i,c], k,c] - sum[t[i,k,a,c] * v[k,b,j,c], k,c] + sum[t[i,c] * t[j,d] * t[k,a] * t[l,b] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[k,b] * t[l,d] * t[i,j,a,c] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[k,a] * t[l,d] * t[i,j,c,b] * v[k,l,c,d], k,l,c,d] + sum[t[k,a] * t[l,b] * t[i,j,c,d] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[j,c] * t[l,d] * t[i,k,a,b] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[j,d] * t[l,b] * t[i,k,a,c] * v[k,l,c,d], k,l,c,d] + sum[t[j,d] * t[l,b] * t[i,k,c,a] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[i,c] * t[l,d] * t[j,k,b,a] * v[k,l,c,d], k,l,c,d] + sum[t[i,c] * t[l,a] * t[j,k,b,d] * v[k,l,c,d], k,l,c,d] + sum[t[i,c] * t[l,b] * t[j,k,d,a] * v[k,l,c,d], k,l,c,d] + sum[t[i,k,c,d] * t[j,l,b,a] * v[k,l,c,d], k,l,c,d] + 4 * sum[t[i,k,a,c] * t[j,l,b,d] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[i,k,c,a] * t[j,l,b,d] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[i,k,a,b] * t[j,l,c,d] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[i,k,a,c] * t[j,l,d,b] * v[k,l,c,d], k,l,c,d] + sum[t[i,k,c,a] * t[j,l,d,b] * v[k,l,c,d], k,l,c,d] + sum[t[i,c] * t[j,d] * t[k,l,a,b] * v[k,l,c,d], k,l,c,d] + sum[t[i,j,c,d] * t[k,l,a,b] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[i,j,c,b] * t[k,l,a,d] * v[k,l,c,d], k,l,c,d] - 2 * sum[t[i,j,a,c] * t[k,l,b,d] * v[k,l,c,d], k,l,c,d] + sum[t[j,c] * t[k,b] * t[l,a] * v[k,l,c,i], k,l,c] + sum[t[l,c] * t[j,k,b,a] * v[k,l,c,i], k,l,c] - 2 * sum[t[l,a] * t[j,k,b,c] * v[k,l,c,i], k,l,c] + sum[t[l,a] * t[j,k,c,b] * v[k,l,c,i], k,l,c] - 2 * sum[t[k,c] * t[j,l,b,a] * v[k,l,c,i], k,l,c] + sum[t[k,a] * t[j,l,b,c] * v[k,l,c,i], k,l,c] + sum[t[k,b] * t[j,l,c,a] * v[k,l,c,i], k,l,c] + sum[t[j,c] * t[l,k,a,b] * v[k,l,c,i], k,l,c] + sum[t[i,c] * t[k,a] * t[l,b] * v[k,l,c,j], k,l,c] + sum[t[l,c] * t[i,k,a,b] * v[k,l,c,j], k,l,c] - 2 * sum[t[l,b] * t[i,k,a,c] * v[k,l,c,j], k,l,c] + sum[t[l,b] * t[i,k,c,a] * v[k,l,c,j], k,l,c] + sum[t[i,c] * t[k,l,a,b] * v[k,l,c,j], k,l,c] + sum[t[j,c] * t[l,d] * t[i,k,a,b] * v[k,l,d,c], k,l,c,d] + sum[t[j,d] * t[l,b] * t[i,k,a,c] * v[k,l,d,c], k,l,c,d] + sum[t[j,d] * t[l,a] * t[i,k,c,b] * v[k,l,d,c], k,l,c,d] - 2 * sum[t[i,k,c,d] * t[j,l,b,a] * v[k,l,d,c], k,l,c,d] - 2 * sum[t[i,k,a,c] * t[j,l,b,d] * v[k,l,d,c], k,l,c,d] + sum[t[i,k,c,a] * t[j,l,b,d] * v[k,l,d,c], k,l,c,d] + sum[t[i,k,a,b] * t[j,l,c,d] * v[k,l,d,c], k,l,c,d] + sum[t[i,k,c,b] * t[j,l,d,a] * v[k,l,d,c], k,l,c,d] + sum[t[i,k,a,c] * t[j,l,d,b] * v[k,l,d,c], k,l,c,d] + sum[t[k,a] * t[l,b] * v[k,l,i,j], k,l] + sum[t[k,l,a,b] * v[k,l,i,j], k,l] + sum[t[k,b] * t[l,d] * t[i,j,a,c] * v[l,k,c,d], k,l,c,d] + sum[t[k,a] * t[l,d] * t[i,j,c,b] * v[l,k,c,d], k,l,c,d] + sum[t[i,c] * t[l,d] * t[j,k,b,a] * v[l,k,c,d], k,l,c,d] - 2 * sum[t[i,c] * t[l,a] * t[j,k,b,d] * v[l,k,c,d], k,l,c,d] + sum[t[i,c] * t[l,a] * t[j,k,d,b] * v[l,k,c,d], k,l,c,d] + sum[t[i,j,c,b] * t[k,l,a,d] * v[l,k,c,d], k,l,c,d] + sum[t[i,j,a,c] * t[k,l,b,d] * v[l,k,c,d], k,l,c,d] - 2 * sum[t[l,c] * t[i,k,a,b] * v[l,k,c,j], k,l,c] + sum[t[l,b] * t[i,k,a,c] * v[l,k,c,j], k,l,c] + sum[t[l,a] * t[i,k,c,b] * v[l,k,c,j], k,l,c] + v[a,b,i,j]

Source: Baumgartner, et al. *(2005)*

Rewriting to reduce operation counts

Naïvely, ≈ 4 × N^{10} flops

$$S_{abij} = \sum_{c,d,e,f,k,l} A_{acik} \times B_{befl} \times C_{dfjk} \times D_{cdel}$$

$$\Downarrow$$

$$S_{abij} = \sum_{c,k} \left(\sum_{d,f} \left(\sum_{e,l} B_{befl} \times D_{cdel} \right) \times C_{dfjk} \right) \times A_{acik}$$

Assuming associativity and distributivity, ≈ 6 × N^6 flops, but also requires temporary storage.

Source: Baumgartner, et al. *(2005)*

Operation and storage minimization *via* loop fusion

$$
\begin{aligned}
T^{(1)}_{bcdf} &= \sum_{e,l} B_{befl} \times D_{cdel} \\
T^{(2)}_{bcjk} &= \sum_{d,f} T^{(1)}_{bcdf} \times C_{dfjk} \\
S_{abij} &= \sum_{c,k} T^{(2)}_{bcjk} \times A_{acik}
\end{aligned}
$$

$T1 = T2 = S = 0$

for b, c, d, e, f, l **do**

$T1[b, c, d, f] \mathrel{+}= B[b, e, f, l] \cdot D[c, d, e, l]$

for b, c, d, f, j, k **do**

$T2[b, c, j, k] \mathrel{+}= T1[b, c, d, f] \cdot C[d, f, j, k]$

for a, b, c, i, j, k **do**

$S[a, b, i, j] \mathrel{+}= T2[b, c, j, k] \cdot A[a, c, i, k]$

Operation and storage minimization *via* loop fusion

$$T^{(1)}_{bcdf} = \sum_{e,l} B_{befl} \times D_{cdel}$$

$$T^{(2)}_{bcjk} = \sum_{d,f} T^{(1)}_{bcdf} \times C_{dfjk}$$

$$S_{abij} = \sum_{c,k} T^{(2)}_{bcjk} \times A_{acik}$$

$T1 = T2 = S = 0$
for b, c, d, e, f, l **do**
  $T1[b, c, d, f] \mathrel{+}= B[b, e, f, l] \cdot D[c, d, e, l]$
for b, c, d, f, j, k **do**
  $T2[b, c, j, k] \mathrel{+}= T1[b, c, d, f] \cdot C[d, f, j, k]$
for a, b, c, i, j, k **do**
  $S[a, b, i, j] \mathrel{+}= T2[b, c, j, k] \cdot A[a, c, i, k]$

$S = 0$
for b, c **do**
  $T1f \leftarrow 0,\ T2f \leftarrow 0$
  for d, f **do**
    for e, l **do**
      $T1f \mathrel{+}= B[b, e, f, l] \cdot D[c, d, e, l]$
    for j, k **do**
      $T2f[j, k] \mathrel{+}= T1f \cdot C[d, f, j, k]$
  for a, i, j, k **do**
    $S[a, b, i, j] \mathrel{+}= T2f[j, k] \cdot A[a, c, i, k]$

Time-space trade-offs

Max index of
***a*—*f*: *O*(1000)**
***i*—*k*: *O*(100)**

for a, e, c, f **do**

  for i, j **do**

    $X_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$ ← **"Contraction" of *T* over *i, j***

for c, e, b, k **do**

  $T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$

for a, f, b, k **do**

  $T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$

Integrals, O(1000) flops

for c, e, a, f **do**

  for b, k **do**

    $Y_{ceaf} \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$ ← **"Contraction" over $T^{(1)}$ and $T^{(2)}$**

for c, e, a, f **do**

  $E \mathrel{+}= X_{aecf} \cdot Y_{ceaf}$

Time-space trade-offs

Max index of
***a*—*f*: *O*(1000)**
***i*—*k*: *O*(100)**

$$\textbf{for } a, e, c, f \textbf{ do}$$
$$\quad \textbf{for } i, j \textbf{ do}$$
$$\quad\quad X_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$$
$$\textbf{for } c, e, b, k \textbf{ do}$$
$$\quad T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$$
$$\textbf{for } a, f, b, k \textbf{ do}$$
$$\quad T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$$
$$\textbf{for } c, e, a, f \textbf{ do}$$
$$\quad \textbf{for } b, k \textbf{ do}$$
$$\quad\quad Y_{ceaf} \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$$
$$\textbf{for } c, e, a, f \textbf{ do}$$
$$\quad E \mathrel{+}= X_{aecf} \cdot Y_{ceaf}$$

Same indices
⇒ Loop fusion candidates

Time-space trade-offs

for a, e, c, f **do**

  for i, j **do**

    $X_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$

for c, e, b, k **do**

  $T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$

for a, f, b, k **do**

  $T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$

for c, e, a, f **do**

  for b, k **do**

    $Y_{ceaf} \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$

for c, e, a, f **do**

  $E \mathrel{+}= X_{aecf} \cdot Y_{ceaf}$

→

for a, e, c, f **do**

  for i, j **do**

    $X_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$

for a, c, e, f, b, k **do**

  $T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$

for a, e, c, f, b, k **do**

  $T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$

→

for c, e, a, f **do**

  for b, k **do**

    $Y_{ceaf} \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$

→

for c, e, a, f **do**

  $E \mathrel{+}= X_{aecf} \cdot Y_{ceaf}$

Add extra flops

Time-space trade-offs

for a, e, c, f **do**

  for i, j **do**

    $X_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$

for c, e, b, k **do**

  $T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$

for a, f, b, k **do**

  $T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$

for c, e, a, f **do**

  for b, k **do**

    $Y_{ceaf} \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$

for c, e, a, f **do**

  $E \mathrel{+}= X_{aecf} \cdot Y_{ceaf}$

⇐ Fused

for a, e, c, f **do**

  for i, j **do**

    $x \mathrel{+}= T_{ijae} \cdot T_{ijcf}$

  for b, k **do**

    $T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$

    $T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$

    $y \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$

  $E \mathrel{+}= x \cdot y$

Tiled & partially fused

for a, e, c, f **do**

  for i, j **do**

    $X_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$

for c, e, b, k **do**

  $T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$

for a, f, b, k **do**

  $T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$

for c, e, a, f **do**

  for b, k **do**

    $Y_{ceaf} \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$

for c, e, a, f **do**

  $E \mathrel{+}= X_{aecf} \cdot Y_{ceaf}$

for a^B, e^B, c^B, f^B **do**

  for a, e, c, f **do**

    for i, j **do**

      $\hat{X}_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$

  for b, k **do**

    for c, e **do**

      $\hat{T}^{(1)}_{ce} \leftarrow f_1(c, e, b, k)$

    for a, f **do**

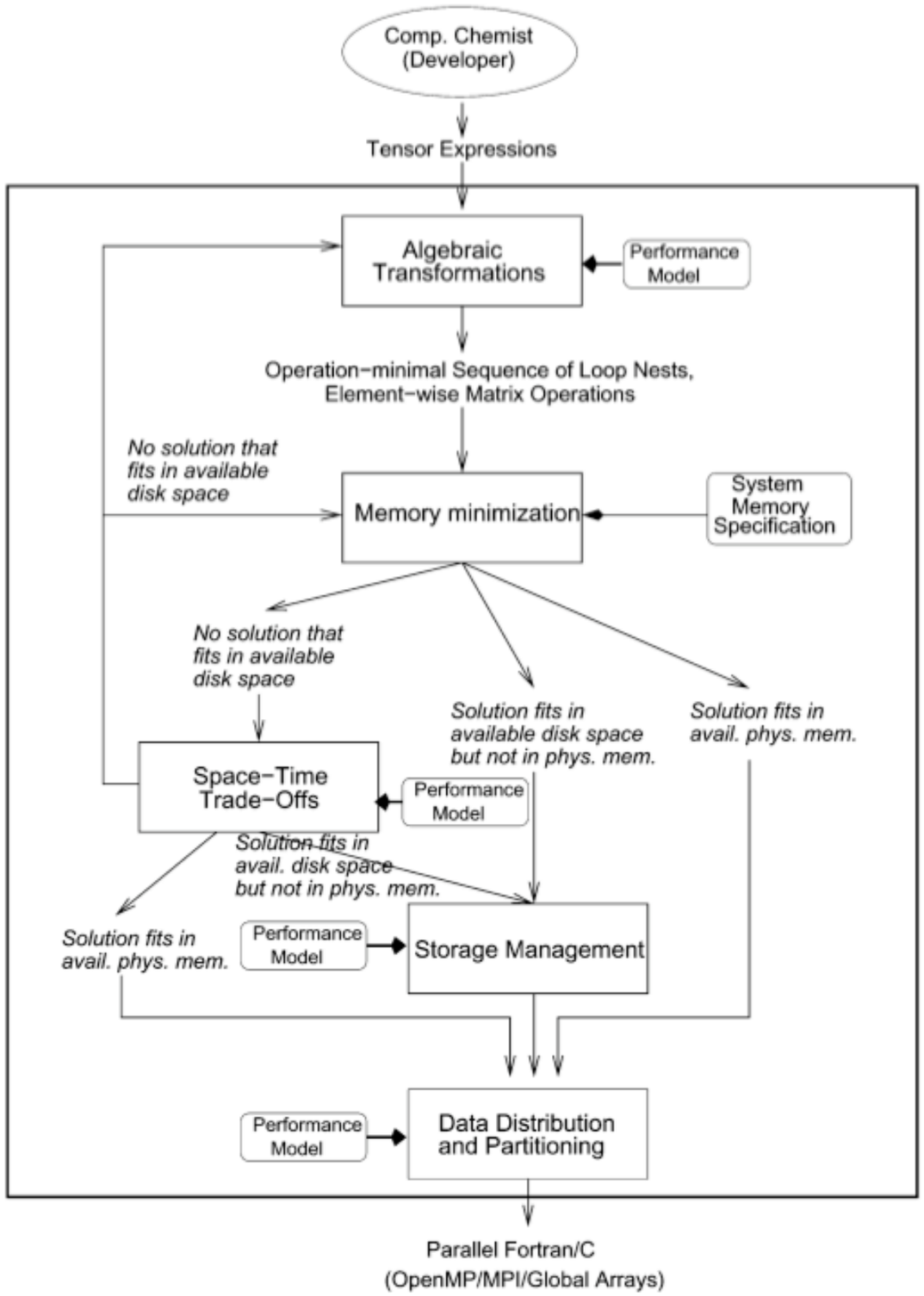
      $\hat{T}^{(2)}_{af} \leftarrow f_2(a, f, b, k)$

    for c, e, a, f **do**

      $\hat{Y}_{ceaf} \mathrel{+}= \hat{T}^{(1)}_{ce} \cdot \hat{T}^{(2)}_{af}$

for c, e, a, f **do**

  $E \mathrel{+}= \hat{X}_{aecf} \cdot \hat{Y}_{ceaf}$



- **Transform algebraically**, to minimize flops
- **Minimize temporary storage**
- **Distribute and partition data** for a parallel system
- **Search** wrt **space-time trade-off** (feedback)
- For **out-of-core** problems, apply **optimize data locality**
- Generate final program (C/ Fortran + MPI/Global-arrays)

Tensor loop nest ⇒ Expression tree

for a, e, c, f **do**

  for i, j **do**

    $X_{aecf} \mathrel{+}= T_{ijae} \cdot T_{ijcf}$

for c, e, b, k **do**

  $T^{(1)}_{cebk} \leftarrow f_1(c, e, b, k)$

for a, f, b, k **do**

  $T^{(2)}_{afbk} \leftarrow f_2(a, f, b, k)$

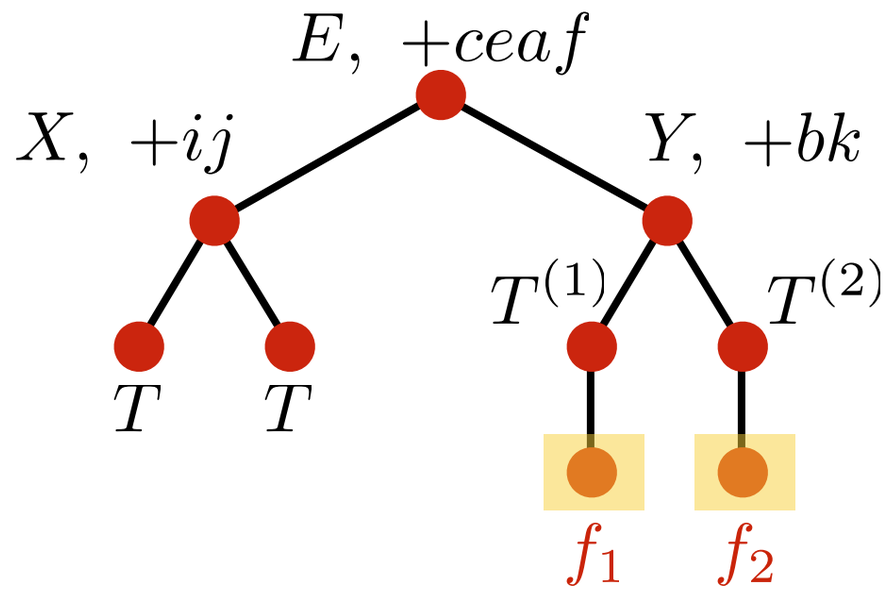
for c, e, a, f **do**

  for b, k **do**

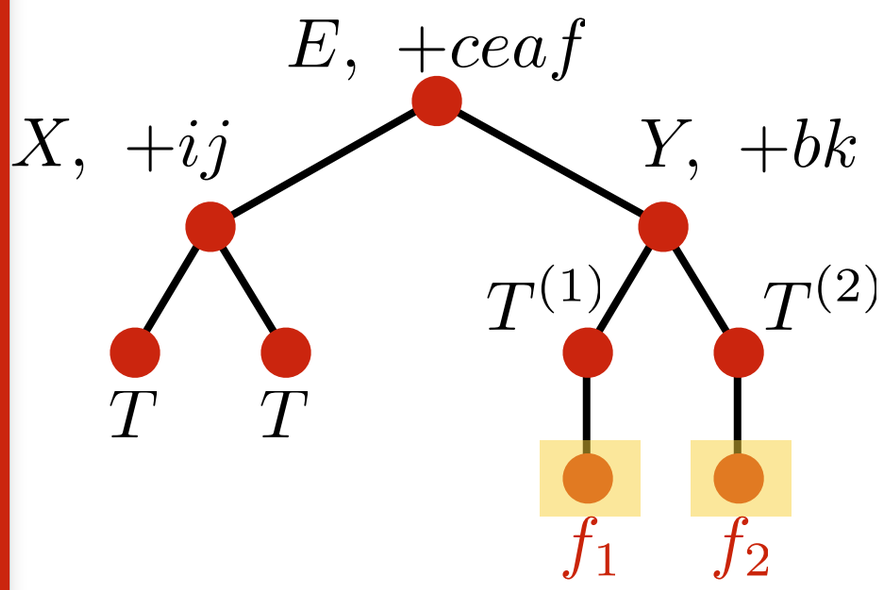
    $Y_{ceaf} \mathrel{+}= T^{(1)}_{cebk} \cdot T^{(2)}_{afbk}$

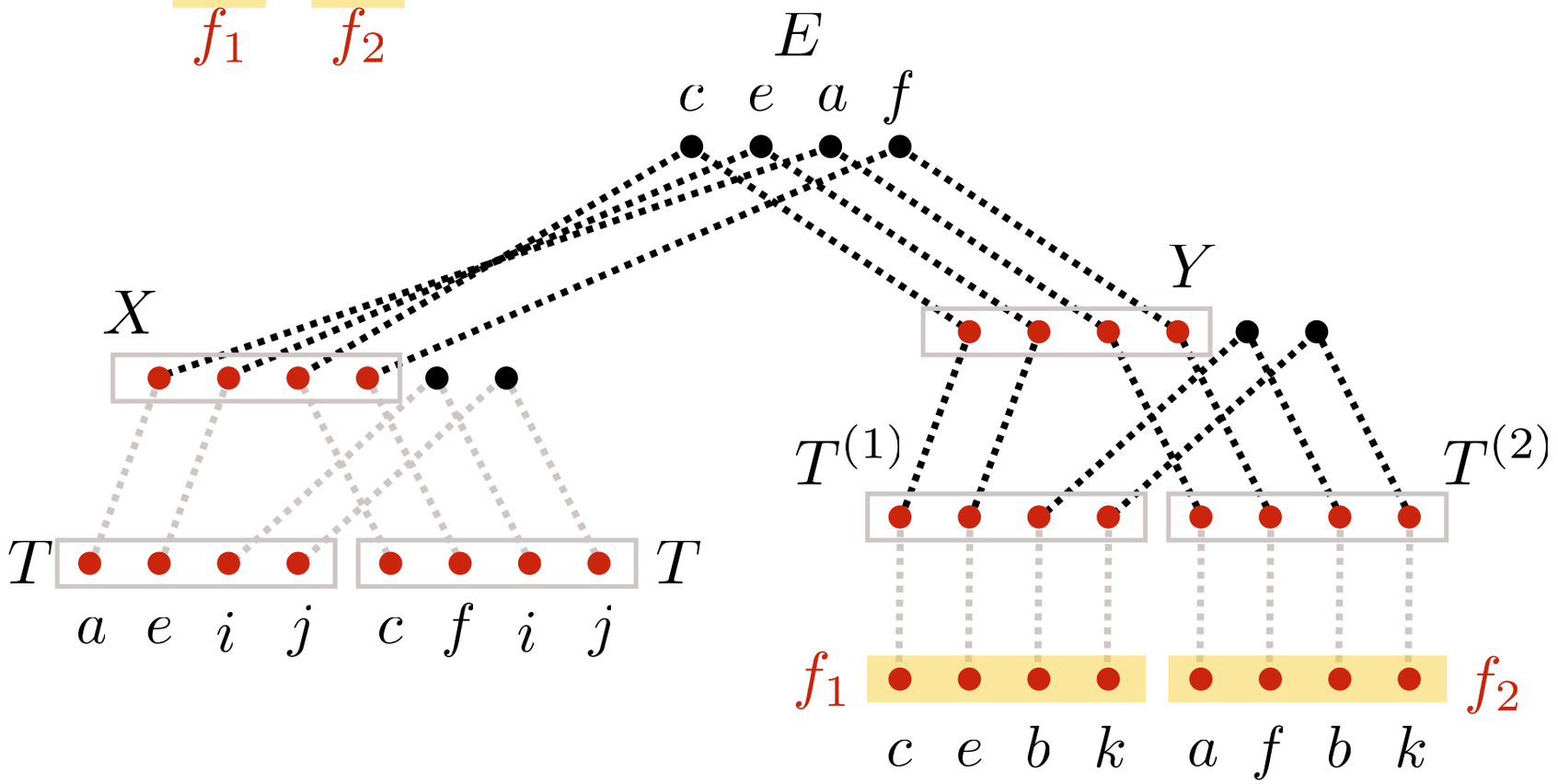
for c, e, a, f **do**

  $E \mathrel{+}= X_{aecf} \cdot Y_{ceaf}$

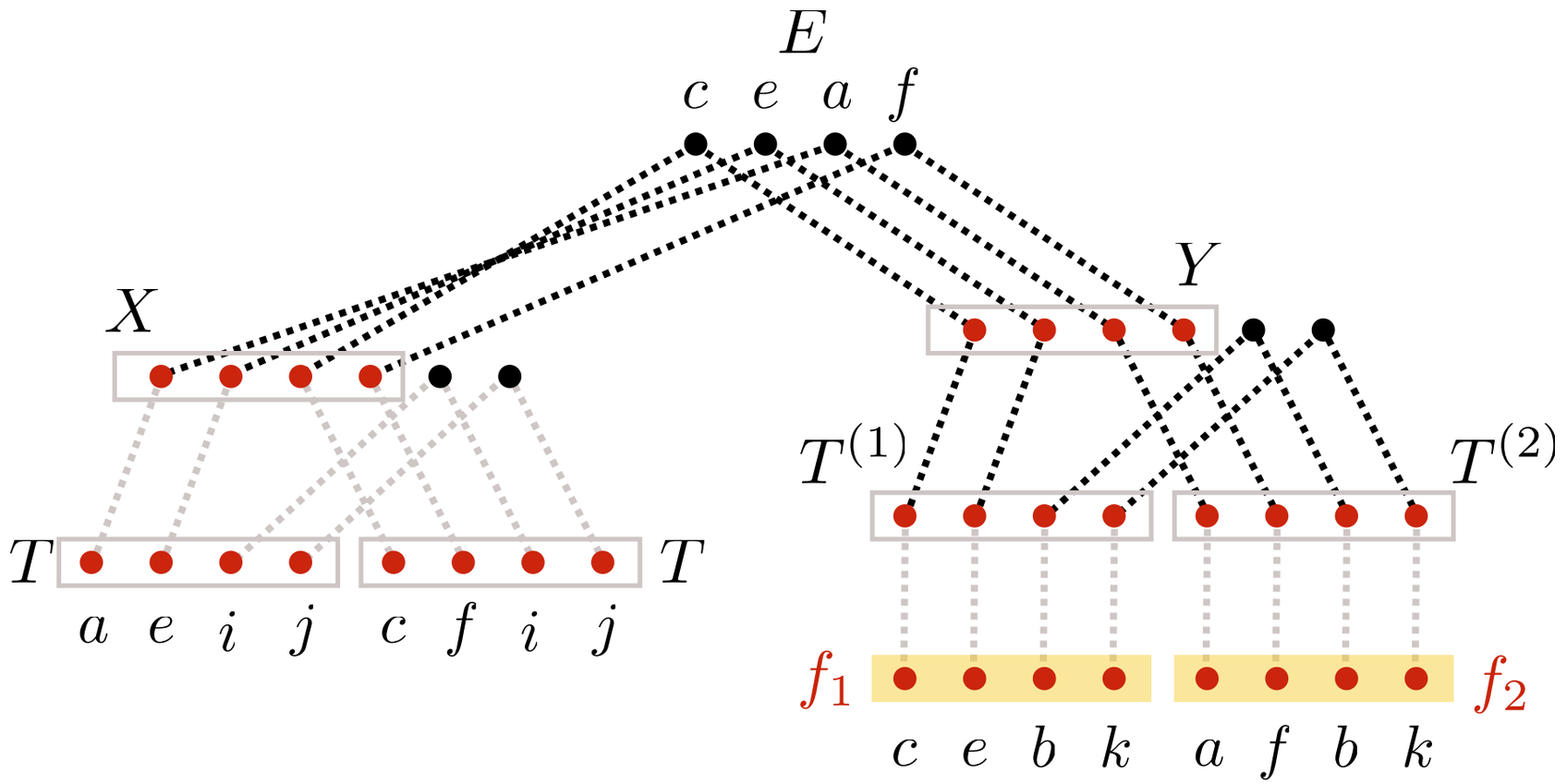


Expression tree ⇒ fusion graph

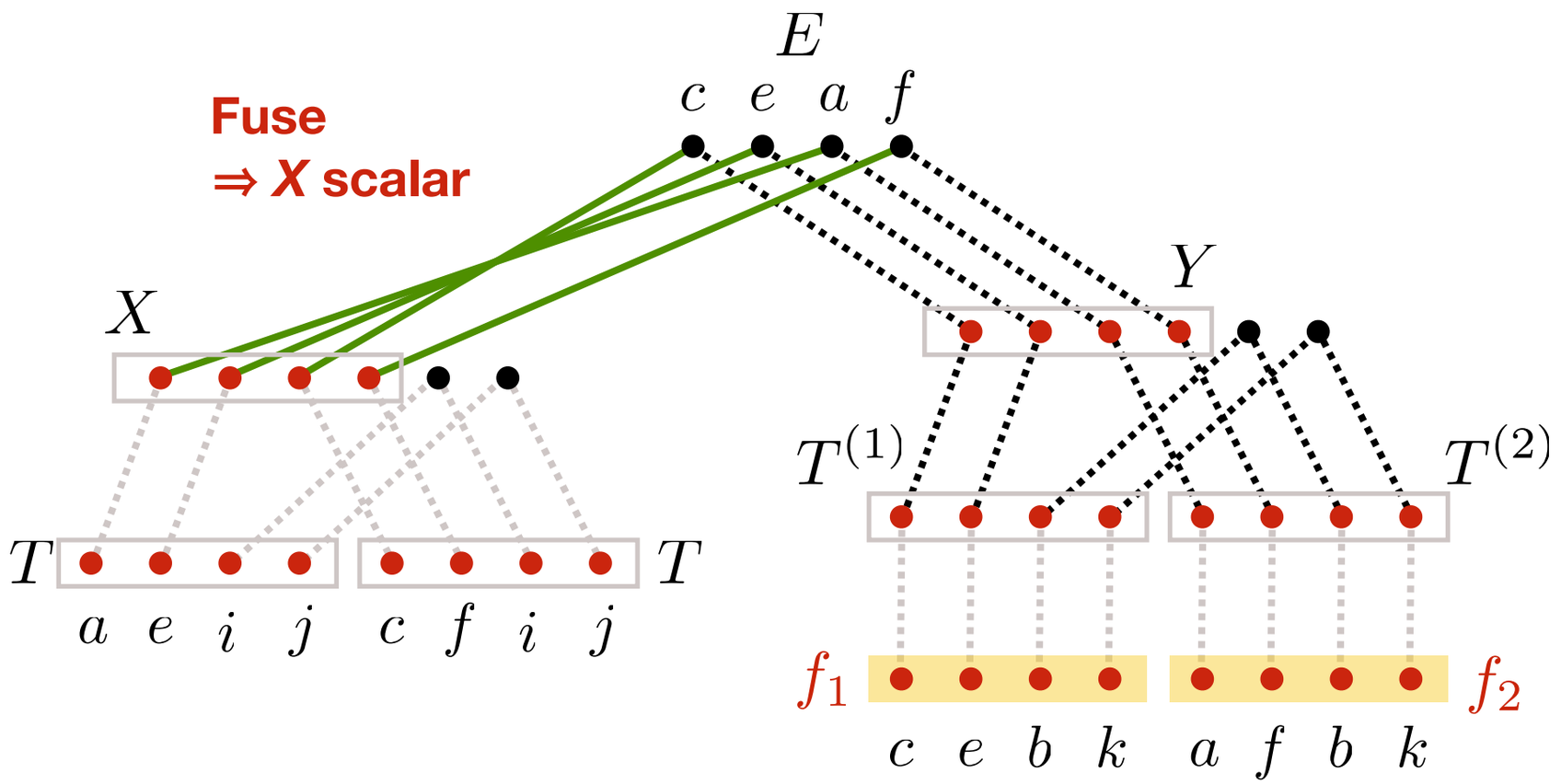




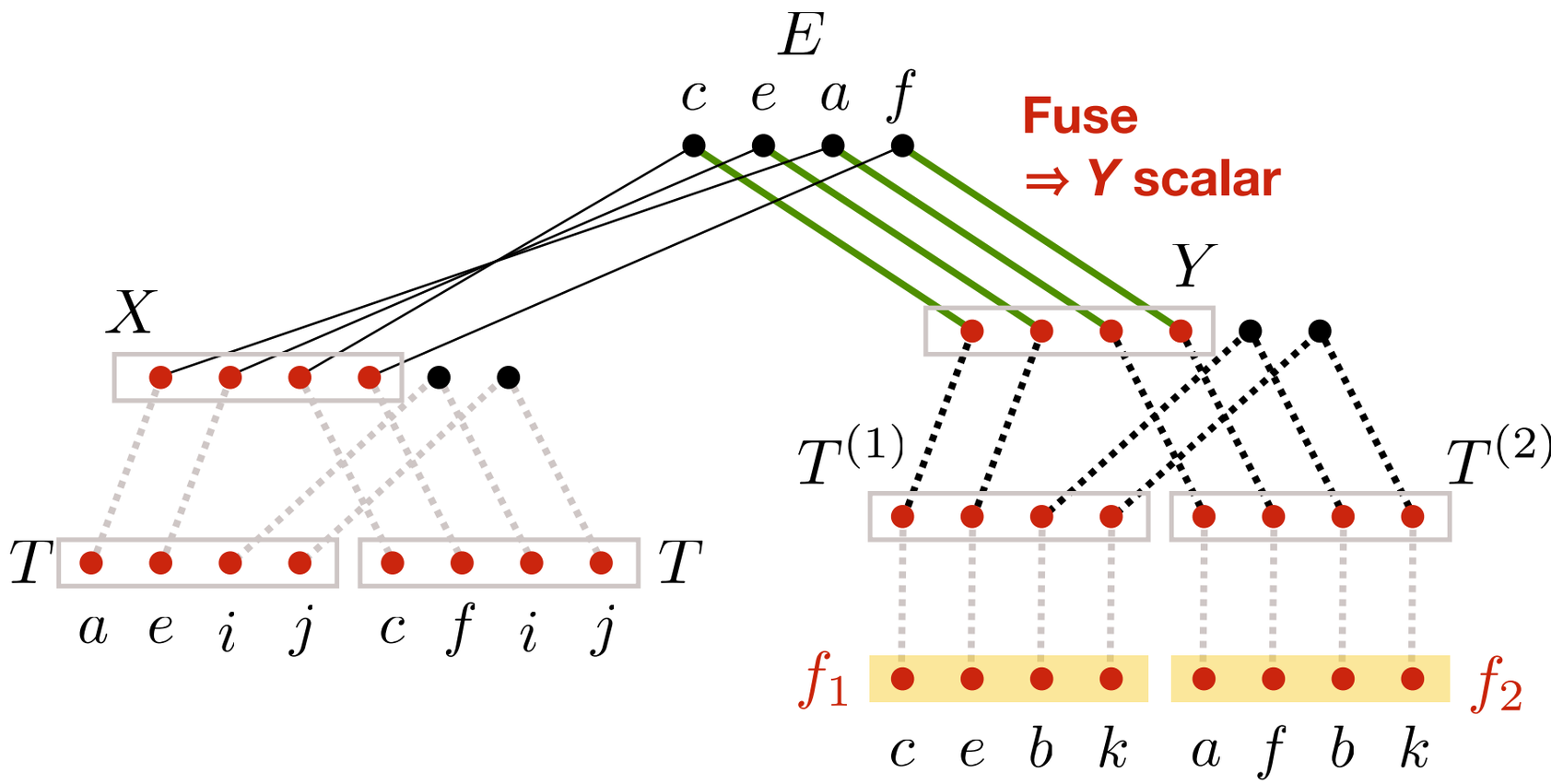
Fusion graph



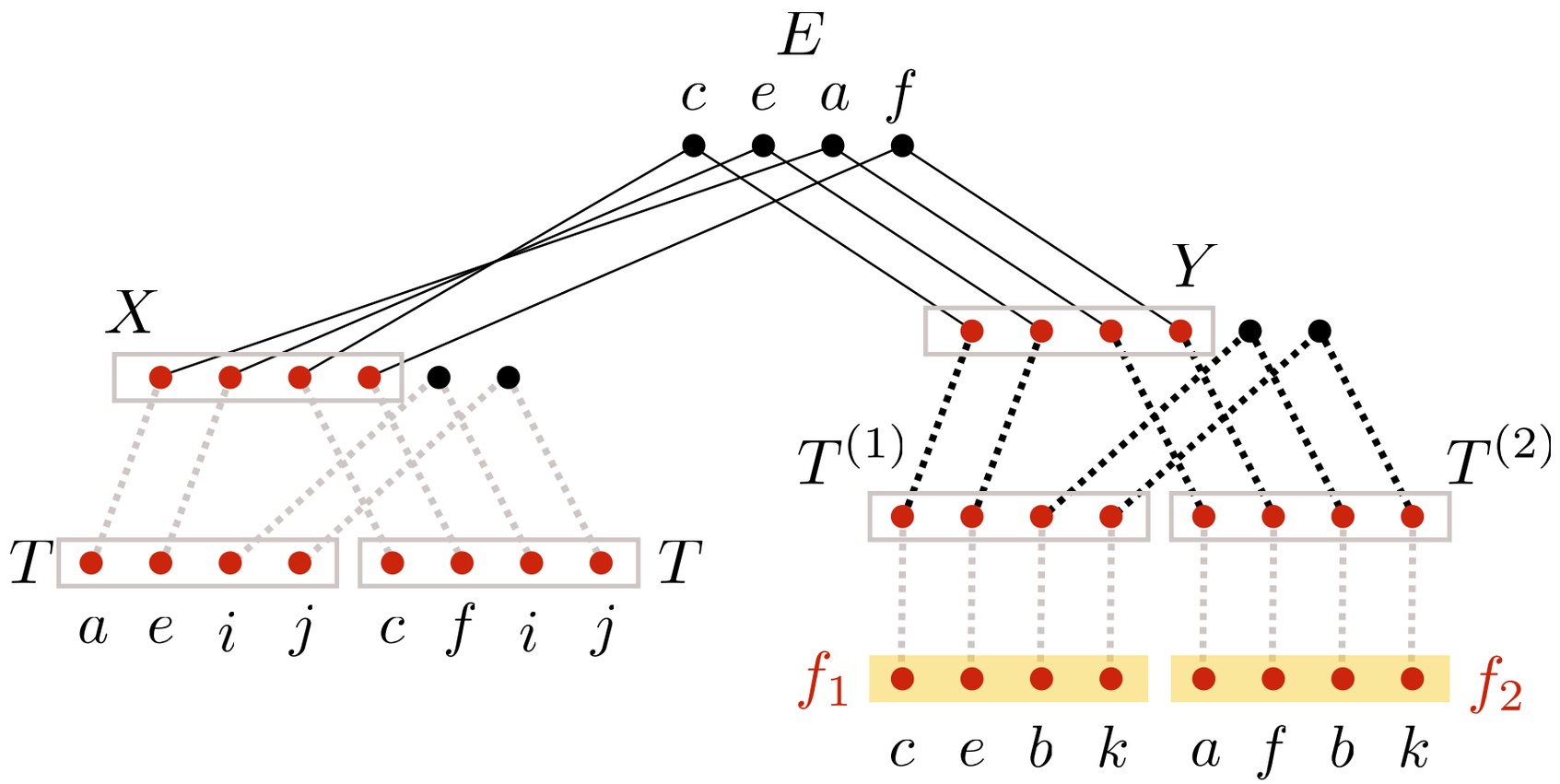
Fusion graph



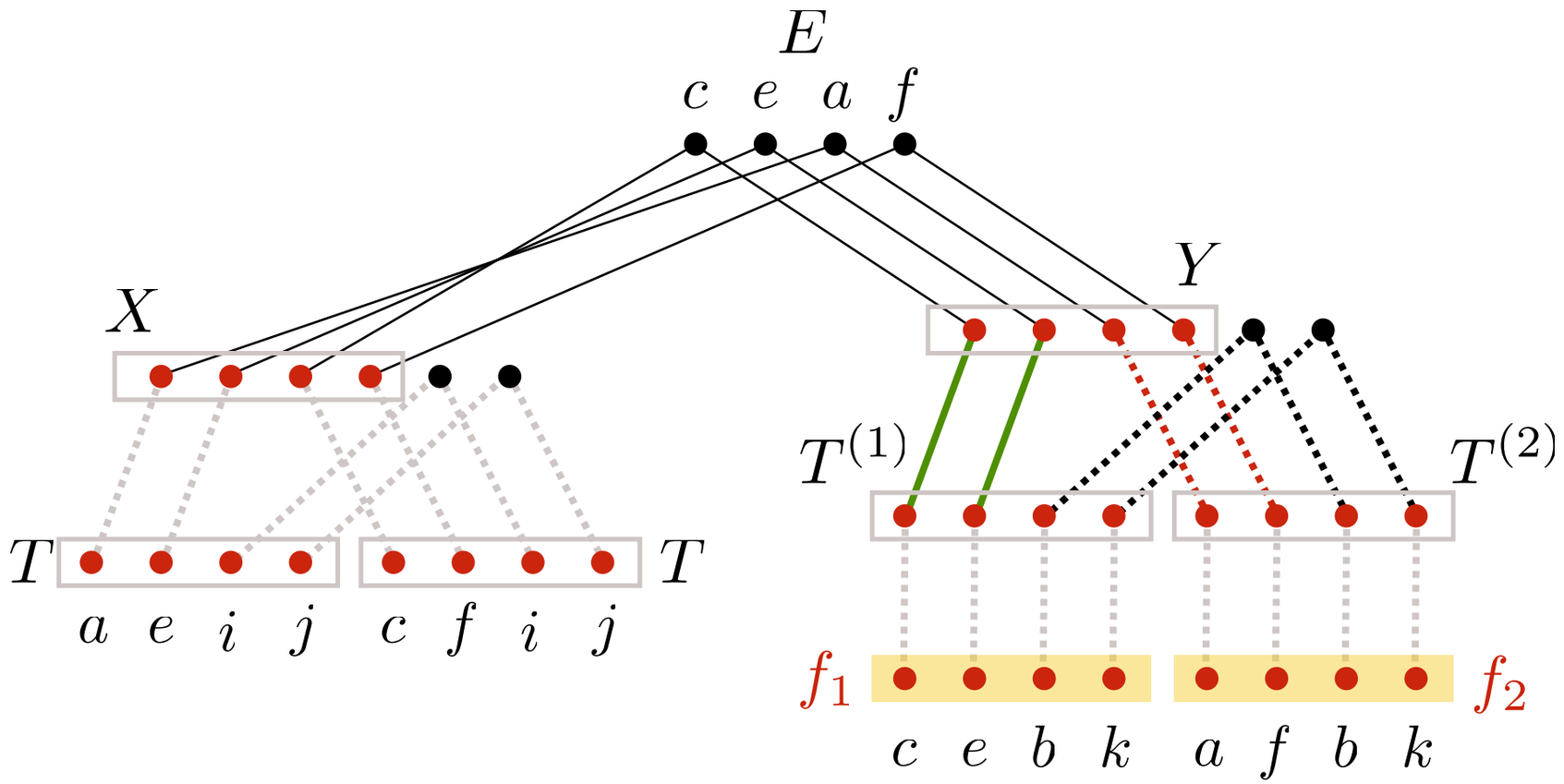
Fusion graph



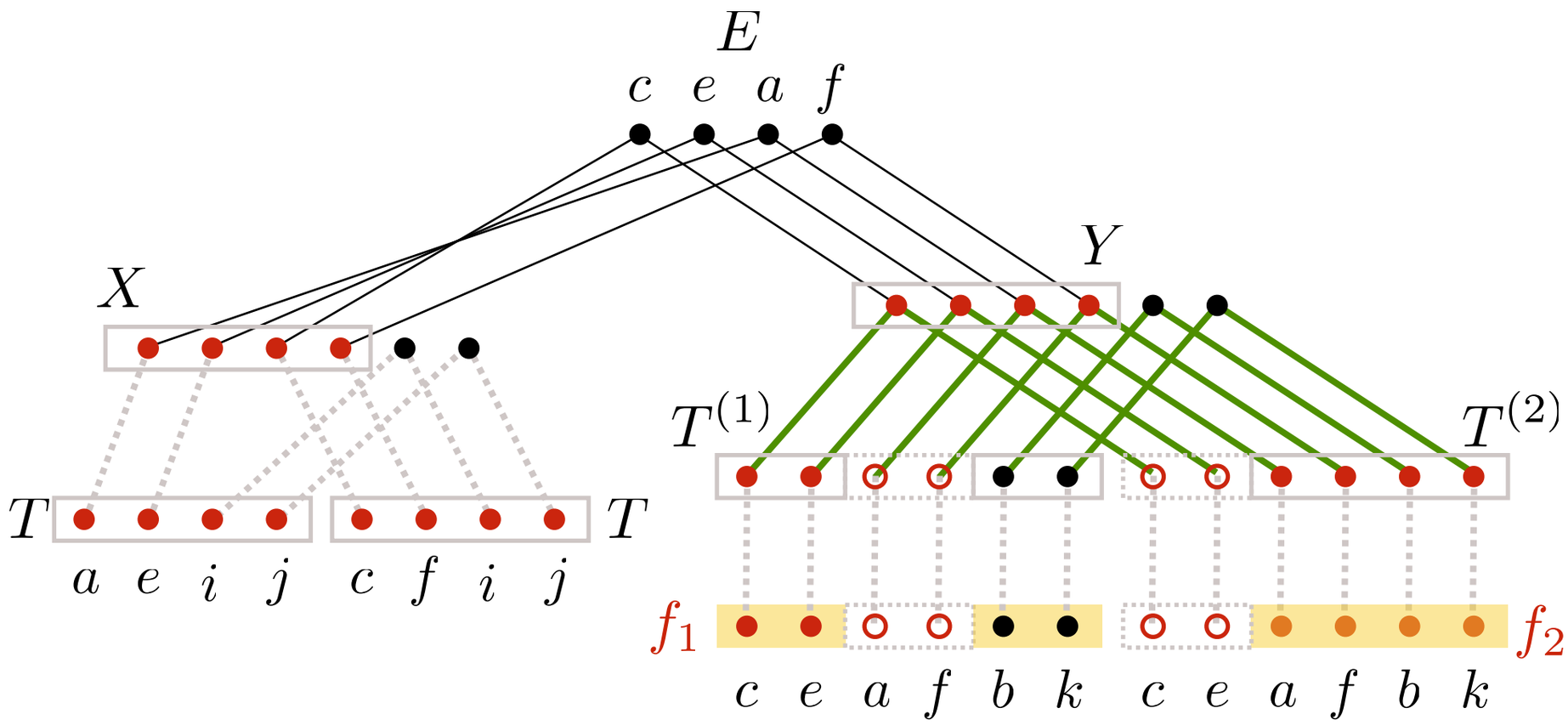
Fusion graph

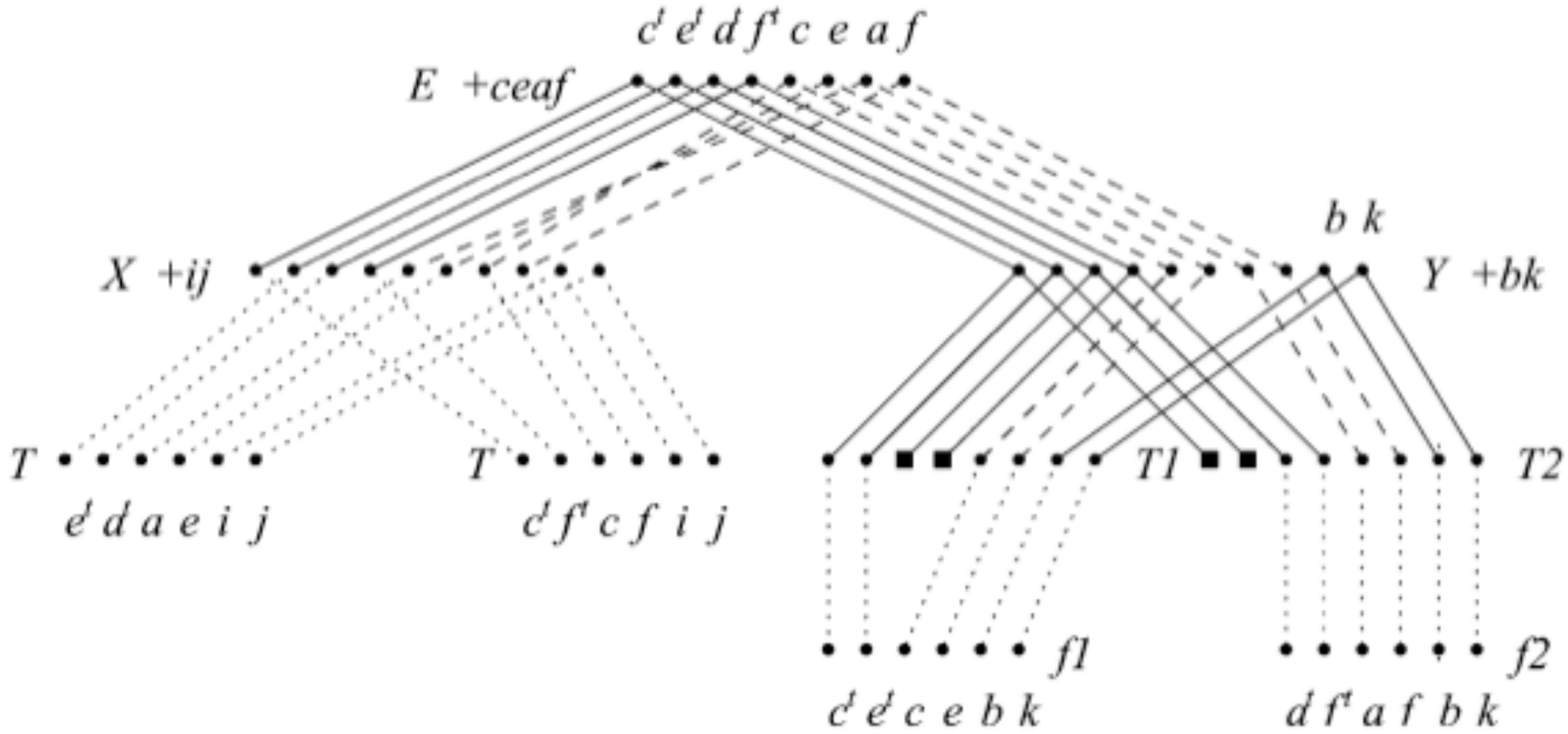


Fusion graph



Fusion graph





(b) Partially fused computation from Fig. 5.

Empirical compilers and tools

Code generation tools for autotuning

- Code generation tools
  - GNU Superoptimizer -- Exhaustive search over schedules of straight-line code
  - Denali -- Theorem proving based scheduling
  - iFKO (Whaley @ UTSA) -- Iterative floating-point kernel optimizer
  - POET (Yi @ UTSA) -- Parameterized Optimizations for Empirical Tuning

Iterative/empirical compilers

- Compile-time
  - "Iterative compilation" -- Kisuki, Knijnenberg, O'Boyle, *et al*.
  - Hybrid model/search-based compiler -- Hall, et al. (USC)
  - Eigenmann @ Purdue (Polaris)
  - Quinlan, et al. (LLNL / PERI)
  - Qasem (TSU), Kennedy, Mellor-Crummey (Rice) -- Whole program tuning
  - Compilers that learn -- Cavazos (UDel); Stephenson/Amarsinghe (MIT)
- Run-time: Voss, et al.: ADAPT

Administrivia

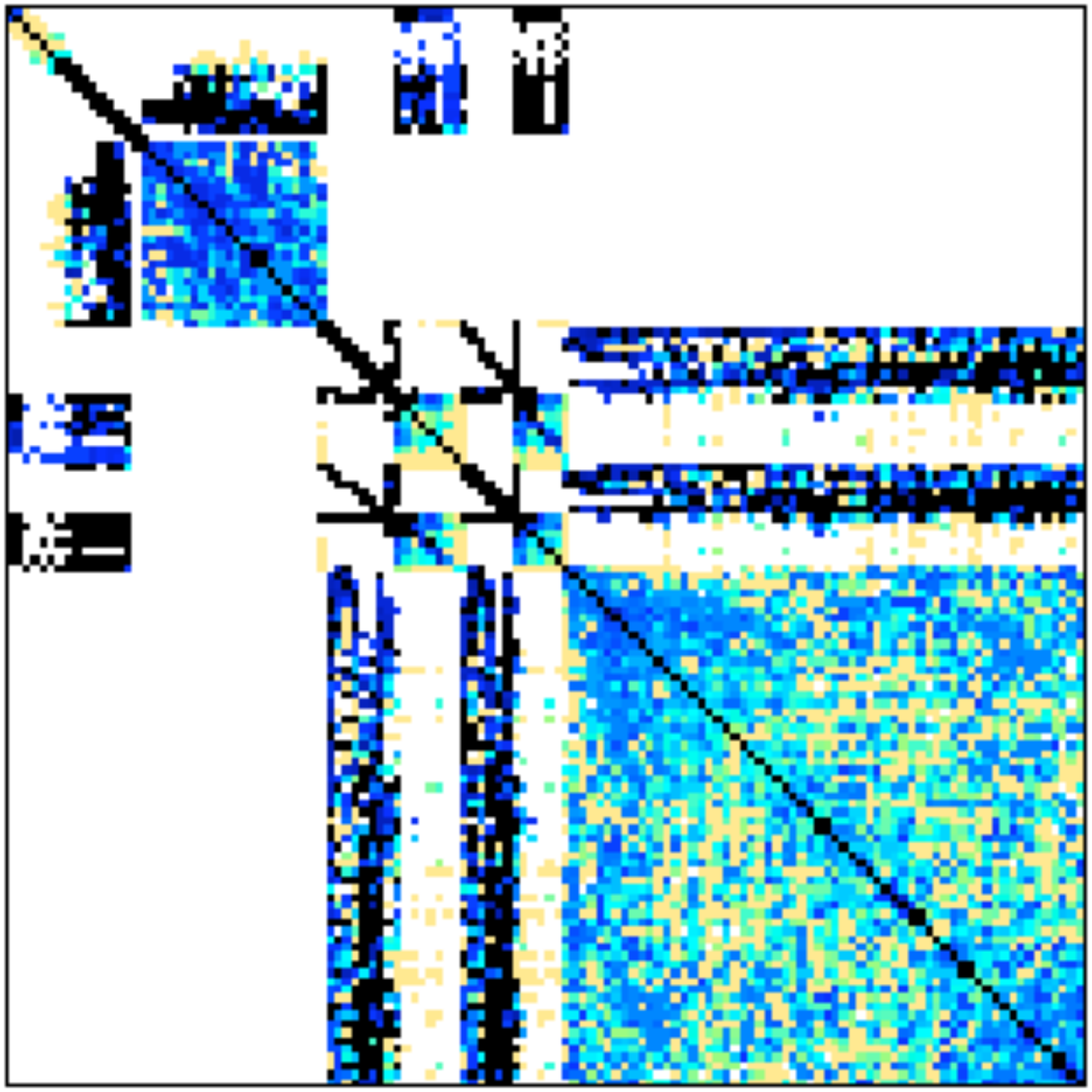
Upcoming schedule changes

- Some adjustment of topics (TBD)
- Today — Project proposals due
- Th 3/13 — SIAM Parallel Processing (attendance encouraged)
- Tu 4/1 — No class
- Th 4/3 — Attend talk by Doug Post from DoD HPC Modernization Program

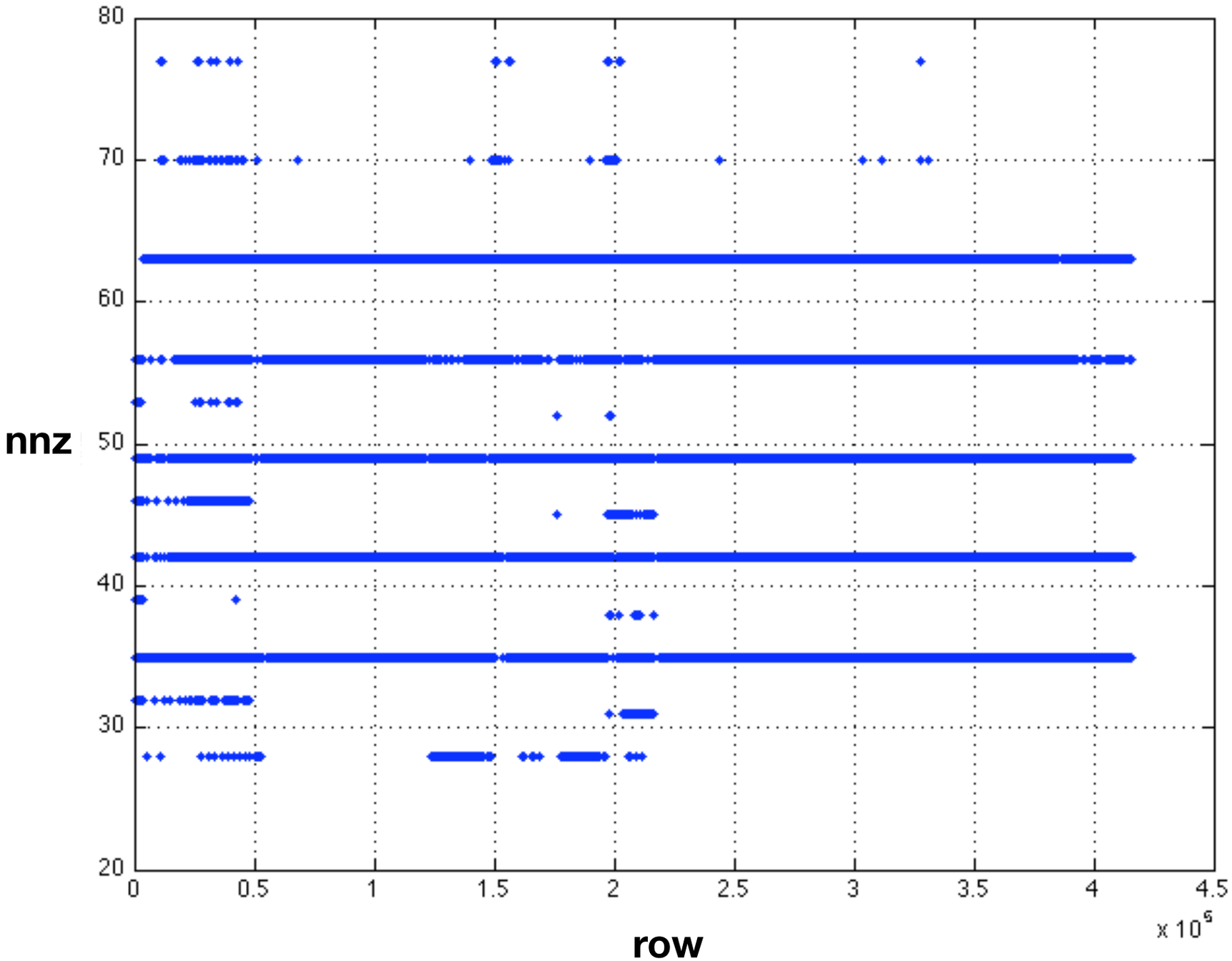
Homework 1: Parallel conjugate gradients

- Put name on write-up!
- Grading: **100** pts max
  - Correct implementation — **50** pts
  - **Evaluation — 45** pts
    - Tested on two samples matrices — 5
    - Implemented and tested on stencil — 10
    - “Explained” performance (e.g., per proc, load balance, comp. vs. comm) — 15
    - Performance model — 15 pts
  - Write-up “quality” — 5 pts

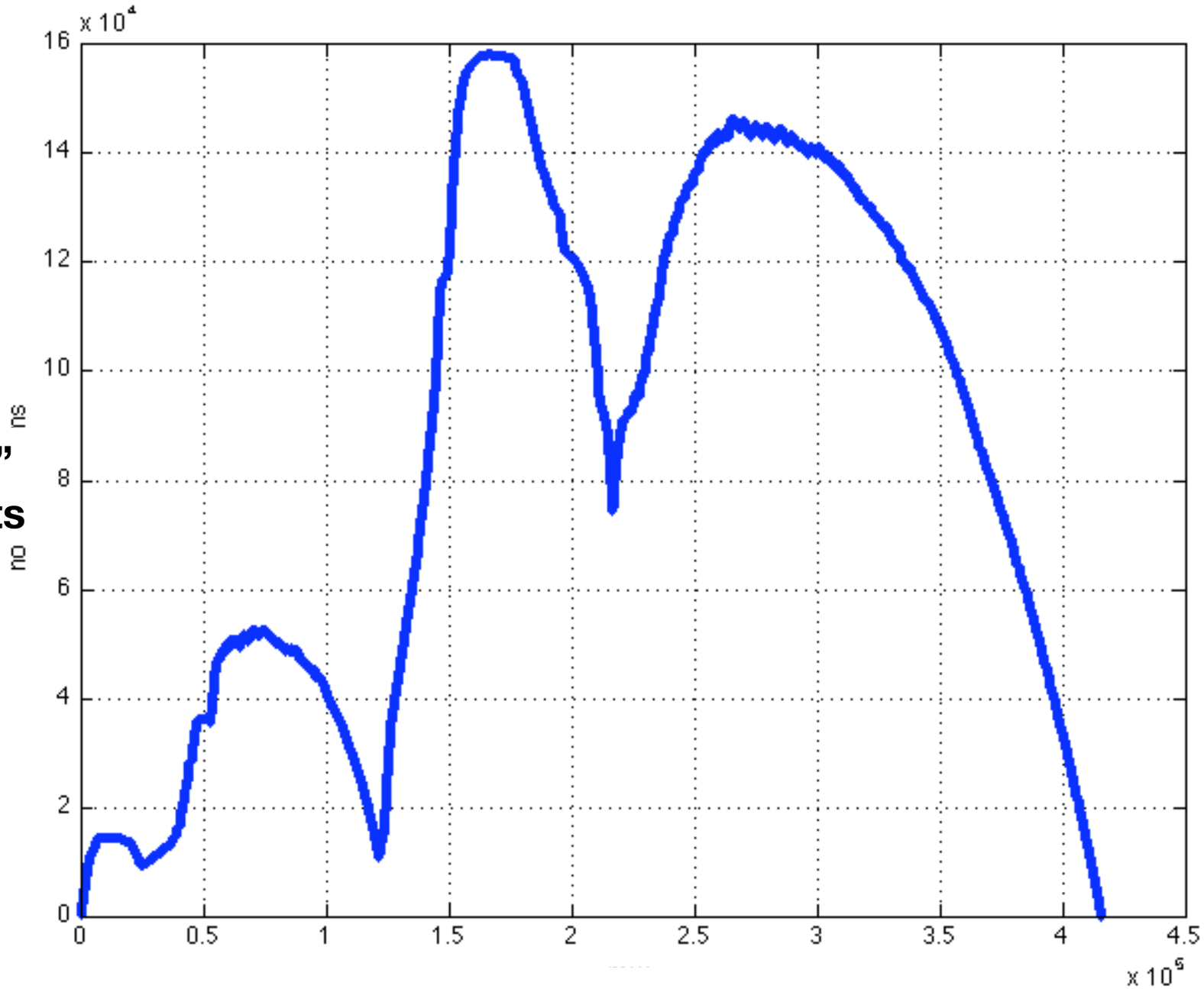
Spy plot of matrix ‘msdoor--UF1644’



Non-zeros per row ‘msdoor--UF1644’

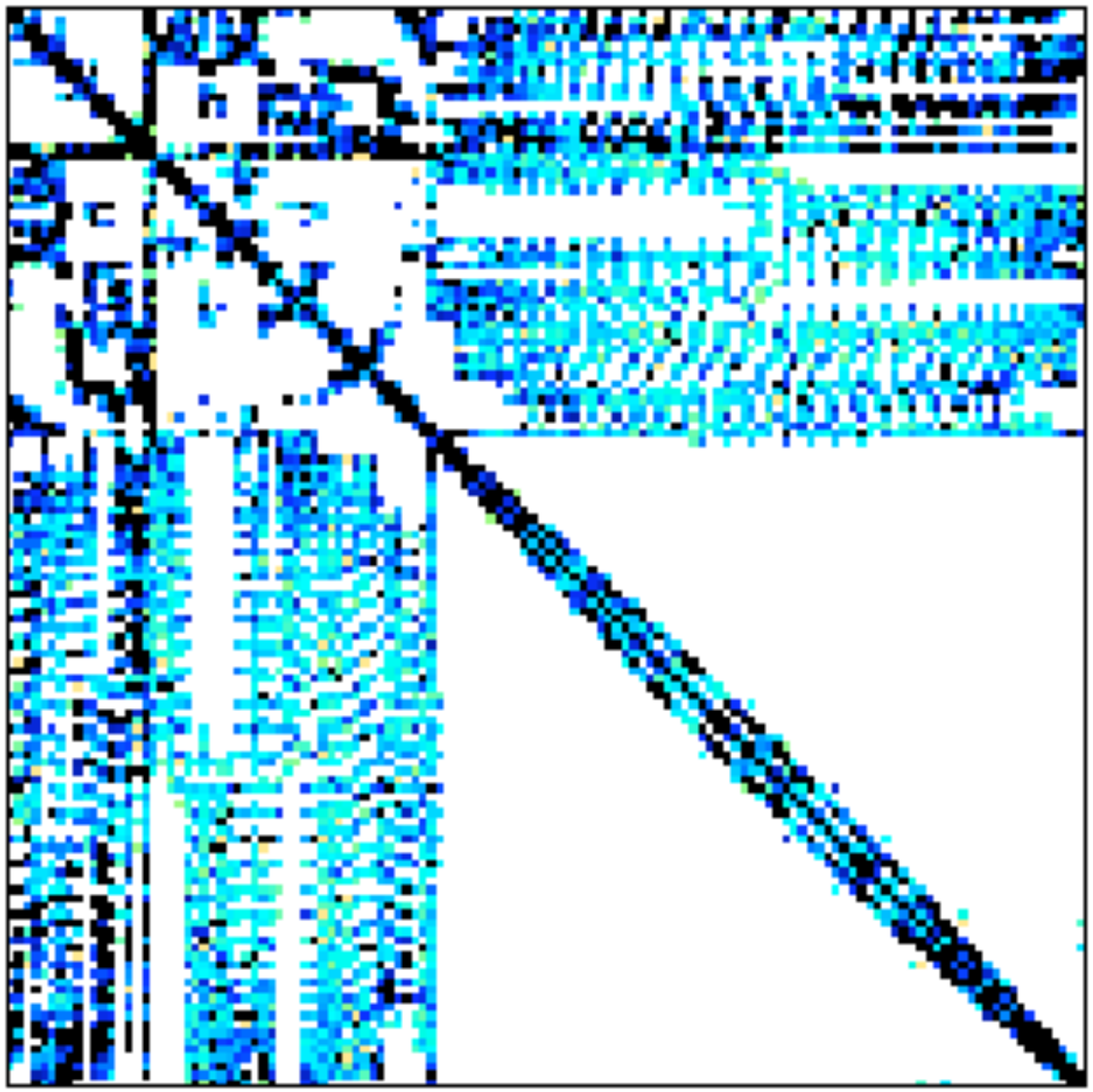


Non-zeros per row ‘msdoor--UF1644’

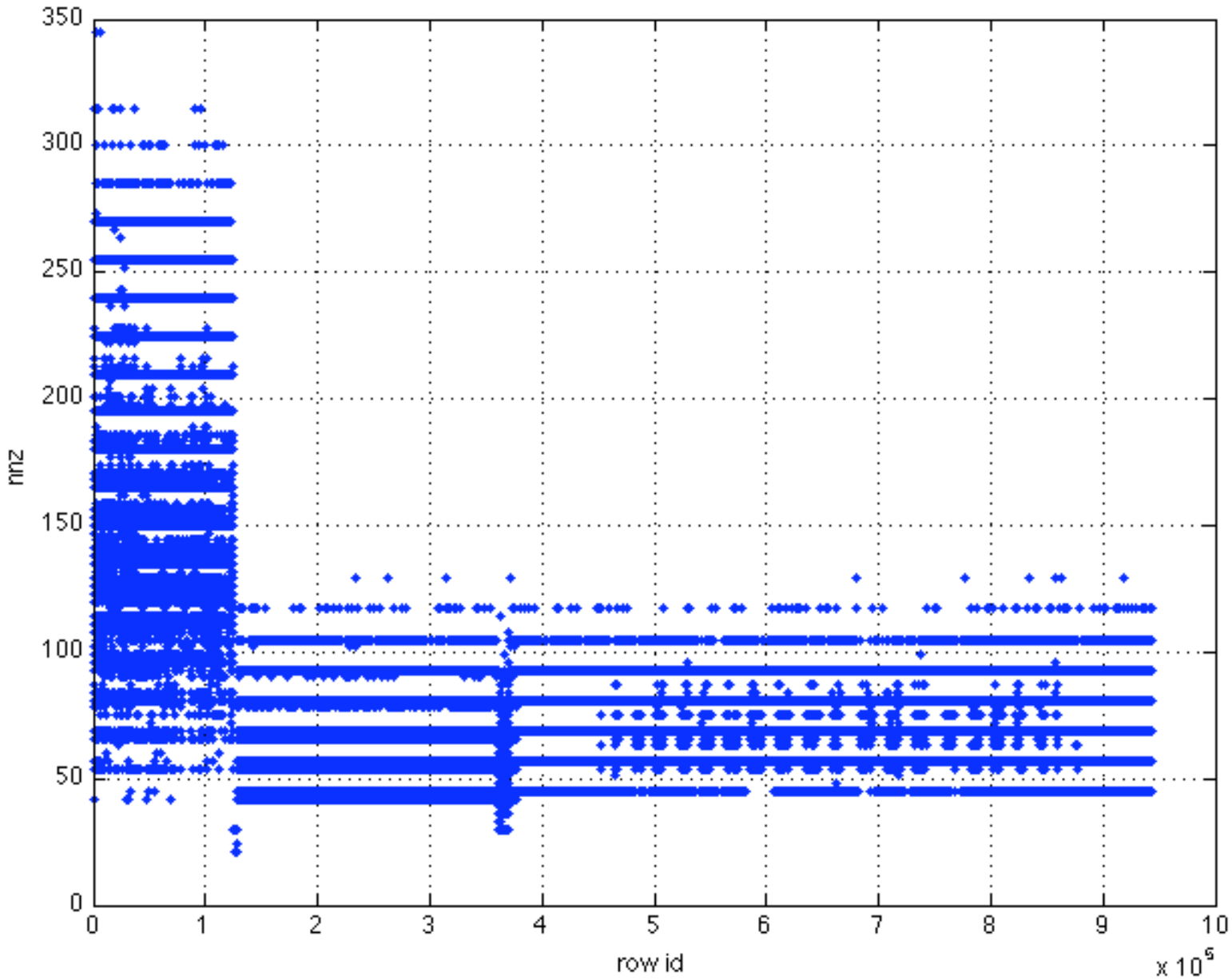


“active” elements

Spy plot of matrix ‘audikw_1--UF1252’



Non-zeros per row ‘audikw_1--UF1252’

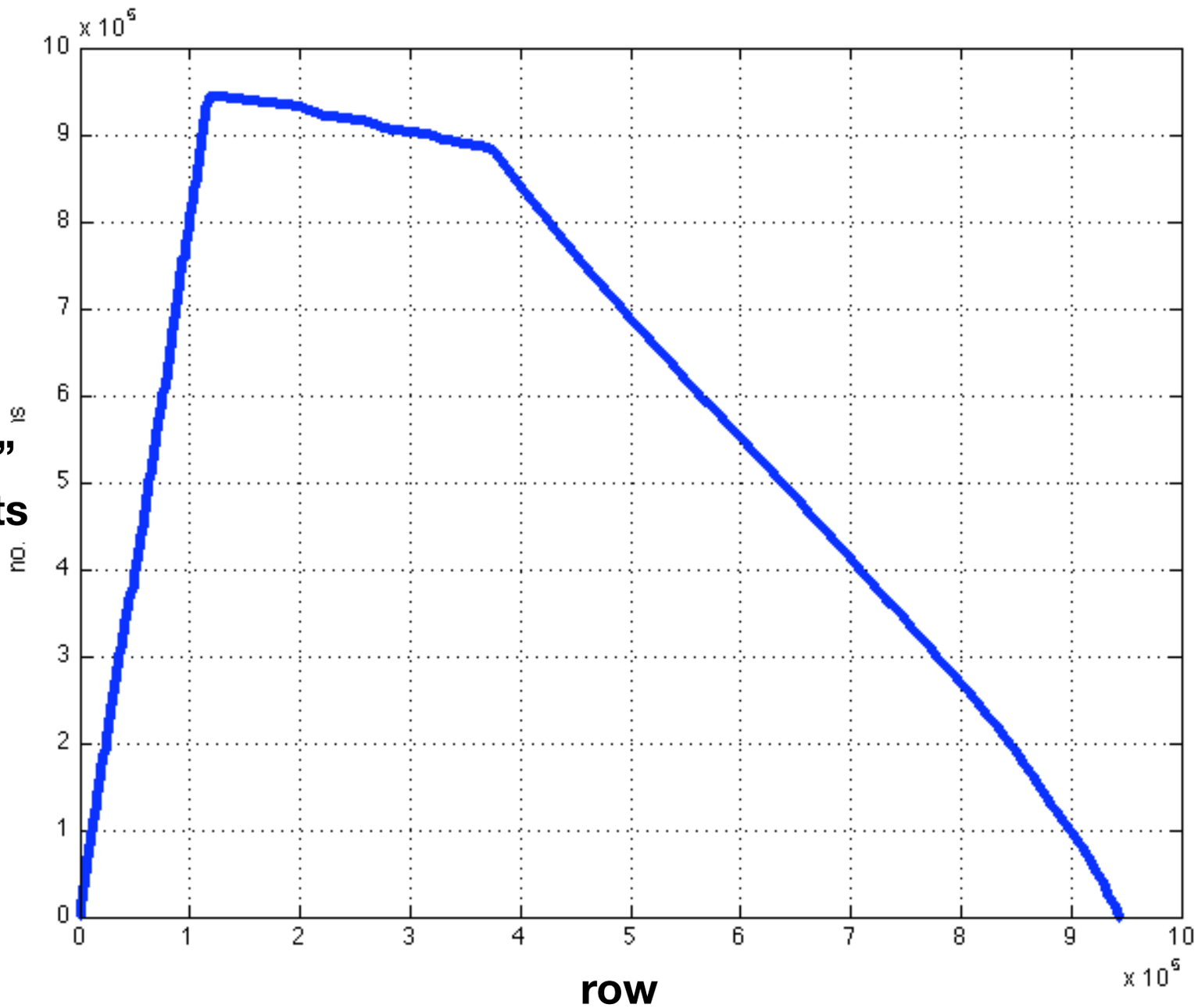


nnz

row

Active elements for ‘audikw_1--UF1252’

“active” elements



Homework 2: Parallel n-body using the particle-mesh method

- Acceleration of particle *i*, due to forces from all other particles:

$$\ddot{\mathbf{r}}_i = \sum_{j \neq i} \mathbf{F}_{ij} = -\sum_{j \neq i} \frac{G m_j (\mathbf{r}_i - \mathbf{r}_j)}{||\mathbf{r}_i - \mathbf{r}_j||^3}$$

- Not yet decided what exactly I will ask to do (implementation? pencil-and-paper? thoughts?)

Projects

- Your goal should be to do something useful, interesting, and/or publishable!
  - Something you're already working on, suitably adapted for this course
  - Faculty-sponsored/mentored
  - Collaborations encouraged

“In conclusion…”

Backup slides

My criteria for “approving” your project

- “Relevant to this course:” Many themes, so think (and “do”) broadly
  - Parallelism and architectures
  - Numerical algorithms
  - Programming models
  - Performance modeling/analysis

General styles of projects

- Theoretical: Prove something hard (high risk)
- Experimental:
  - Parallelize something
  - Take existing parallel program, and improve it using models & experiments
  - Evaluate algorithm, architecture, or programming model

Examples

- *Anything of interest to a faculty member/project outside CoC*
- Parallel sparse triple product ($R*A*R^T$, used in multigrid)
- Future FFT
- Out-of-core or I/O-intensive data analysis and algorithms
- Block iterative solvers (convergence & performance trade-offs)
- Sparse LU
- Data structures and algorithms (trees, graphs)
- Look at mixed-precision
- Discrete-event approaches to continuous systems simulation
- Automated performance analysis and modeling, tuning
- “Unconventional,” but related
  - Distributed deadlock detection for MPI
  - UPC language extensions (dynamic block sizes)
  - Exact linear algebra