Autotuning and specialization: a case study

Jaewook Shin†, Mary Hall‡, Jacqueline Chame§,
Chun Chen‡, Paul Fisher†, Paul Hovland†

† Argonne National Laboratory     ‡ University of Utah    § USC Information Sciences Institute
This work is part of TUNE

- Collaboration between Argonne, ISI, Utah
- Goal: develop compiler technology to automatically tune performance of DOE applications
- Target: Cray XT4 system at Oak Ridge
  - multi core Opteron nodes with SSE-3 SIMD extensions
Collaborative autotuning and specialization

- Programmer uses autotuning tools for parts of tuning process
  - not fully automated
- Autotuning tools used with specialization for specific classes of known input sizes
- Specialization enables compiler/tool to generate highly optimized code for specific computations, expected input sizes
Collaborative autotuning + specialization

- application
  - code triage
    - performance data
    - code outline
      - outlined kernel
        - ROSE
        - HPCToolkit, PAPI
        - CHiLL, POET
        - ROSE
        - ActiveHarmony, GCO
      - code transformations & code variant generation
        - empirical search
          - best specialized code variants
        - specialized code variants
      - user
        - transformation scripts + specialization information
  - transformation scripts + specialization information
  - library
Collaborative autotuning + specialization

- Application
- Code triage
- Performance data
- Code outline
- Outline kernel
- User
- Transformation scripts + specialization information
- Heuristics
- Empirical search
- Code transformations & code variant generation
- Specialized code variants
- Best specialized code variants
- Library

PAPI, CHiLL,
CHiLL

• Framework for Composing High-Level Loop transformations
  – based on polyhedral model
  – script interface for programmers/compilers
  – optimization strategy specified as sequence of composable transformations
  – support for specialization
### CHiLL example

L1    DO K=1,N-1
L2     DO I=K+1,N
S1          A(I,K)=A(I,K)/A(K,K)
L3     DO I=K+1,N
L4      DO J=K+1,N
S2       A(I,J)=A(I,J)-A(I,K)*A(K,J)

for(t2 = 1; t2 <= n-1; t2++) {
    for(t4 = t2+1; t4 <= n; t4++) {
        a[t4][t2]=a[t4][t2]/a[t2][t2];
        for(t6 = t2+1; t6 <= n; t6++) {
            a[t4][t6]=a[t4][t6]-a[t4][t2]*a[t2][t6];
        }
    }
}
CHiLL example

L1    DO K=1,N-1
L2     DO I=K+1,N
S1          A(I,K)=A(I,K)/A(K,K)
L3        DO I=K+1,N
L4           DO J=K+1,N
S2              A(I,J)=A(I,J)-A(I,K)*A(K,J)

for(t2 = 1; t2 <= n-1; t2++) {
    for(t4 = t2+1; t4 <= n; t4++) {
        a[t4][t2]=a[t4][t2]/a[t2][t2];
        for(t6 = t2+1; t6 <= n; t6++) {
            a[t4][t6]=a[t4][t6]-a[t4][t2]*a[t2][t6];
        }
    }
}

for(t2 = 2; t2 <= n; t2 += 64) {
    for(t4 = 1; t4 <= min(n-1,t2+62); t4++) {
        for(t6 = t4+1; t6 <= n; t6++) {
            if (t4 >= t2-1) {
                a[t6][t4] =a[t6][t4]/a[t4][t4]
            }
            if (t8 = max(t4+1,t2); t8 <= min(t2+63,n); t8++) {
                a[t6][t8]=a[t6][t8]-a[t6][t4]*a[t4][t8]
            }
        }
    }
}
CHiLL example

L1   DO K=1,N-1
L2    DO I=K+1,N
S1    A(I,K)=A(I,K)/A(K,K)
L3    DO I=K+1,N
L4    DO J=K+1,N
S2    A(I,J)=A(I,J)-A(I,K)*A(K,J)

script

permute([0,1,2])
tile(1,5,64,1)
split(1,3,[L3≤L1-2])
permute(2,[1,3,7,5])
permute(1,[1,5,7,3])
split(1,3,[L3≥L1-1])
tile(3,3,32,3)
split(3,5,[L9≤L3-1])
tile(3,9,32,5)
datacopy(3,7,2,1)
datacopy(3,7,3)
unroll(3,9,4)
tile(1,7,32,3)
tile(1,5,32,5)
datacopy(1,7,2,1)
datacopy(1,7,3)
unroll(1,9,4)
REAL*8 P1(32,32),P2(32,64),P3(32,32),P4(32,64)
OVER1=0
OVER2=0
DO T2=2,N,64
  IF (66<=T2)
    DO T4=2,T2-32,32
      DO T6=1,T4-1,32
        DO T8=T6,MIN(T4-1,T6+31)
          DO T10=T4,MIN(T2-2,T4+31)
            P1(T8-T6+1,T10-T4+1)=A(T10,T8)
          END DO
        END DO
        DO T8=T2,MIN(T2+63,N)
          DO T10=T6,MIN(T6+31,T4-1)
            P2(T10-T6+1,T8-T2+1)=A(T10,T8)
          END DO
        END DO
        OVER1=MOD(-1+N,4)
        DO T10=T2,MIN(N-OVER1,T2+60),4
          DO T12=T6,MIN(N-OVER1,T2+60),4
            A(T8,T10)=A(T8,T10)-P1(T12-T6+1,T8-T4+1)*P2(T12-T6+1,T10-T2+1)
          END DO
        END DO
      END DO
    END DO
  END IF
END DO

TRSM

data copy
unroll by 4
unroll cleanup
Automatically generated LU (2)

IF (66<=T2)
DO T4=1,T2-33,32
DO T6=T2-1,N,32
DO T8=T4,T4+31
    DO T10=T6,MIN(N,T6+31)
    P3(T8-T4+1,T10-T6+1)=A(T10,T8)
    DO T8=T2,MIN(T2+63,N)
    DO T10=T4,T4+31
    OVER2=MOD(-1+N,4)
    DO T10=T2,MIN(N-OVER2,T2+60),4
    DO T12=T4,T4+31
        A(T8,T10)=A(T8,T10)-P3(T12-T4+1,T8-T6+1)*P4(T12-T4+1,T10-T2+1)
        A(T8,T10+1)=A(T8,T10+1)-P3(T12-T4+1,T8-T6+1)*P4(T12-T4+1,T10+1-T2+1)
        A(T8,T10+2)=A(T8,T10+2)-P3(T12-T4+1,T8-T6+1)*P4(T12-T4+1,T10+2-T2+1)
        A(T8,T10+3)=A(T8,T10+3)-P3(T12-T4+1,T8-T6+1)*P4(T12-T4+1,T10+3-T2+1)
    DO T10=MAX(T2,N-OVER2+1),MIN(T2+63,N)
    DO T12=T4,T4+31
        A(T8,T10)=A(T8,T10)-P3(T12-T4+1,T8-T6+1)*P4(T12-T4+1,T10-T2+1)
DO T4=T2-1,MIN(N-1,T2+62)
DO T8=T4+1,N
    A(T8,T4)=A(T8,T4)/A(T4,T4)
DO T6=T4+1,MIN(T2+63,N)
DO T8=T4+1,N
    A(T8,T6)=A(T8,T6)-A(T8,T4)*A(T4,T6)
Case study: Nek5000

- Spectral element code
- Applications: nuclear energy, astrophysics, ocean modeling, combustion, bio fluids, ....
- Scales to $P > 10,000$
- ~60% of time spent on mxm44_0
  - matrix multiply of very small, rectangular matrices
  - matrix sizes remain the same for different problem sizes
Case study: Nek5000

- Spectral element code
- Applications: nuclear energy, astrophysics, ocean modeling, combustion, bio fluids
- Scales to $P > 10,000$
- $\sim 60\%$ of time spent on $\text{mxm44}_0$
  - matrix multiply of very small, rectangular matrices
  - matrix sizes remain the same for different problem sizes

from $\sim 112,000$ lines of code to matrix multiply
Profiling mxm44_0

- 8 input sizes comprise 74% of time

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

```fortran
    do i=1,M
        do j=1,N
            c(i,j)=0.0d0
            do k=1,K
                c(i,j)=c(i,j)+a(i,k)*b(k,j)
            end do
        end do
    end do
```
mXM of small matrices: optimization strategy

- Small sizes fit in L1 caches
- Optimization opportunities
  - Exploit reuse in registers
  - Exploit SIMD (in the Opteron SSE-3)
  - Reduce loop overheads
mxf of small matrices: code transformations

• Loop permutation:
  – bring loops with unit stride to innermost position
  – enable native compiler to generate efficient SIMD instructions (no packing)

• Loop unrolling:
  – expose opportunities for scalar replacement and instruction scheduling
  – matrix sizes too small for one-size-fits-all unrolling
  – aggressive unrolling -- full unrolling when profitable
Search space of code variants

• Loop permutation
  – 3 loops $\rightarrow$ 6 loop orders

• Loop unrolling (or unroll-and-jam)
  – $M \times N \times K$ unroll size combinations

• Empirical search
  – size $10 \times 10 \times 10$ $\rightarrow$ 6000 code variants (hours)
  – $100 \times 10 \times 10$ $\rightarrow$ 60,000 code variants (days)
  – Use heuristics to prune search space
Code variants vs. %peak

Opteron Phenom, input size 10x10x10
Reduction of search space using heuristics
Benefits of specialization for Nek5000

• ATLAS with architectural default for AMD64K10h64SSE3 in version 3.8.2
• ACML 4.1.0
• GOTO BLAS goto_barcelona-r1.26
• CHiLL 0.1.5, ifort v.10.1
• 2.5 GHz AMD Opteron
  – separate instruction & data 64KB L1, 512KB L2, 2MB L3
  – Linux Ubuntu_8.04-x86_64
Benefits of specialization for Nek5000

1.34 speedup for Nek5000
Steps to autotuning applications

✓ Performance tools to identify kernels to outline + interface to ROSE
✓ Code outlining automated in ROSE
✓ Common compiler interface to transformation frameworks (POET, CHiLL)
  • Optimization strategy (transformation recipes) still derived by programmers
    – need compiler algorithms for optimization strategy, models
✓ Search engines: Active Harmony (Maryland), GCO (UTK)
  • Pruning heuristics still derived by programmers/compiler developers
Questions?
Example: loop order ijk, unroll 8-4-1

• FUNCTION VAN (A, B, C)

  INTEGER VAN, T4, T6
  DOUBLE PRECISION A, B, C

  DIMENSION A(8, 10)
  DIMENSION B(10, 100)
  DIMENSION C(8, 100)

  DO 2, T4 = 1, 97, 4
    C(1, T4) = 0.0000000000000000000D+00
    C(1 + 1, T4) = 0.0000000000000000000D+00
    C(1 + 2, T4) = 0.0000000000000000000D+00
    C(1 + 3, T4) = 0.0000000000000000000D+00
    C(1 + 4, T4) = 0.0000000000000000000D+00
    C(1 + 5, T4) = 0.0000000000000000000D+00
    C(1 + 6, T4) = 0.0000000000000000000D+00
    C(1 + 7, T4) = 0.0000000000000000000D+00
    C(1, T4 + 1) = 0.0000000000000000000D+00
    C(1 + 1, T4 + 1) = 0.0000000000000000000D+00
    C(1 + 2, T4 + 1) = 0.0000000000000000000D+00
    C(1 + 3, T4 + 1) = 0.0000000000000000000D+00
    C(1 + 4, T4 + 1) = 0.0000000000000000000D+00
    C(1 + 5, T4 + 1) = 0.0000000000000000000D+00
    C(1 + 6, T4 + 1) = 0.0000000000000000000D+00
    C(1 + 7, T4 + 1) = 0.0000000000000000000D+00
    C(1, T4 + 2) = 0.0000000000000000000D+00
    C(1 + 1, T4 + 2) = 0.0000000000000000000D+00
    C(1 + 2, T4 + 2) = 0.0000000000000000000D+00
    C(1 + 3, T4 + 2) = 0.0000000000000000000D+00
    C(1 + 4, T4 + 2) = 0.0000000000000000000D+00
    C(1 + 5, T4 + 2) = 0.0000000000000000000D+00
    C(1 + 6, T4 + 2) = 0.0000000000000000000D+00
    C(1 + 7, T4 + 2) = 0.0000000000000000000D+00
    C(1, T4 + 3) = 0.0000000000000000000D+00
    C(1 + 1, T4 + 3) = 0.0000000000000000000D+00
    C(1 + 2, T4 + 3) = 0.0000000000000000000D+00
    C(1 + 3, T4 + 3) = 0.0000000000000000000D+00
    C(1 + 4, T4 + 3) = 0.0000000000000000000D+00
    C(1 + 5, T4 + 3) = 0.0000000000000000000D+00
    C(1 + 6, T4 + 3) = 0.0000000000000000000D+00
    C(1 + 7, T4 + 3) = 0.0000000000000000000D+00

  2 CONTINUE

  DO 4, T6 = 1, 10, 1
    C(1, T4) = C(1, T4) + A(1, T6) * B(T6, T4)
    C(1 + 1, T4) = C(1 + 1, T4) + A(1 + 1, T6) * B(T6, T4)
    C(1 + 2, T4) = C(1 + 2, T4) + A(1 + 2, T6) * B(T6, T4)
    C(1 + 3, T4) = C(1 + 3, T4) + A(1 + 3, T6) * B(T6, T4)
    C(1 + 4, T4) = C(1 + 4, T4) + A(1 + 4, T6) * B(T6, T4)
    C(1 + 5, T4) = C(1 + 5, T4) + A(1 + 5, T6) * B(T6, T4)
    C(1 + 6, T4) = C(1 + 6, T4) + A(1 + 6, T6) * B(T6, T4)
    C(1 + 7, T4) = C(1 + 7, T4) + A(1 + 7, T6) * B(T6, T4)

  4 CONTINUE

  VAN = 0
  RETURN
END