Quad Core Results

John M. Levesque

May, 2008

Ten Lessons from Quad Core

- Don’t Believe the Compiler
  - This has always been the case

Nx4 Matmul

```
(    45)         DO 46020 I = 1,N
(    46)          DO 46020 J = 1,4
(    47)           A(I,J) = 0.
(    48)           DO 46020 K = 1,4
(    49)            A(I,J) = A(I,J) + B(I,K) * C(K,J)
(    50)  46020 CONTINUE
```

PGI

46, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
47, Loop unrolled 4 times (completely unrolled)
49, Loop not vectorized: loop count too small
Loop unrolled 4 times (completely unrolled)

Pathscale

(lp46020.f:46) Loop has too many loop invariants. Loop was not vectorized.

Rewrite

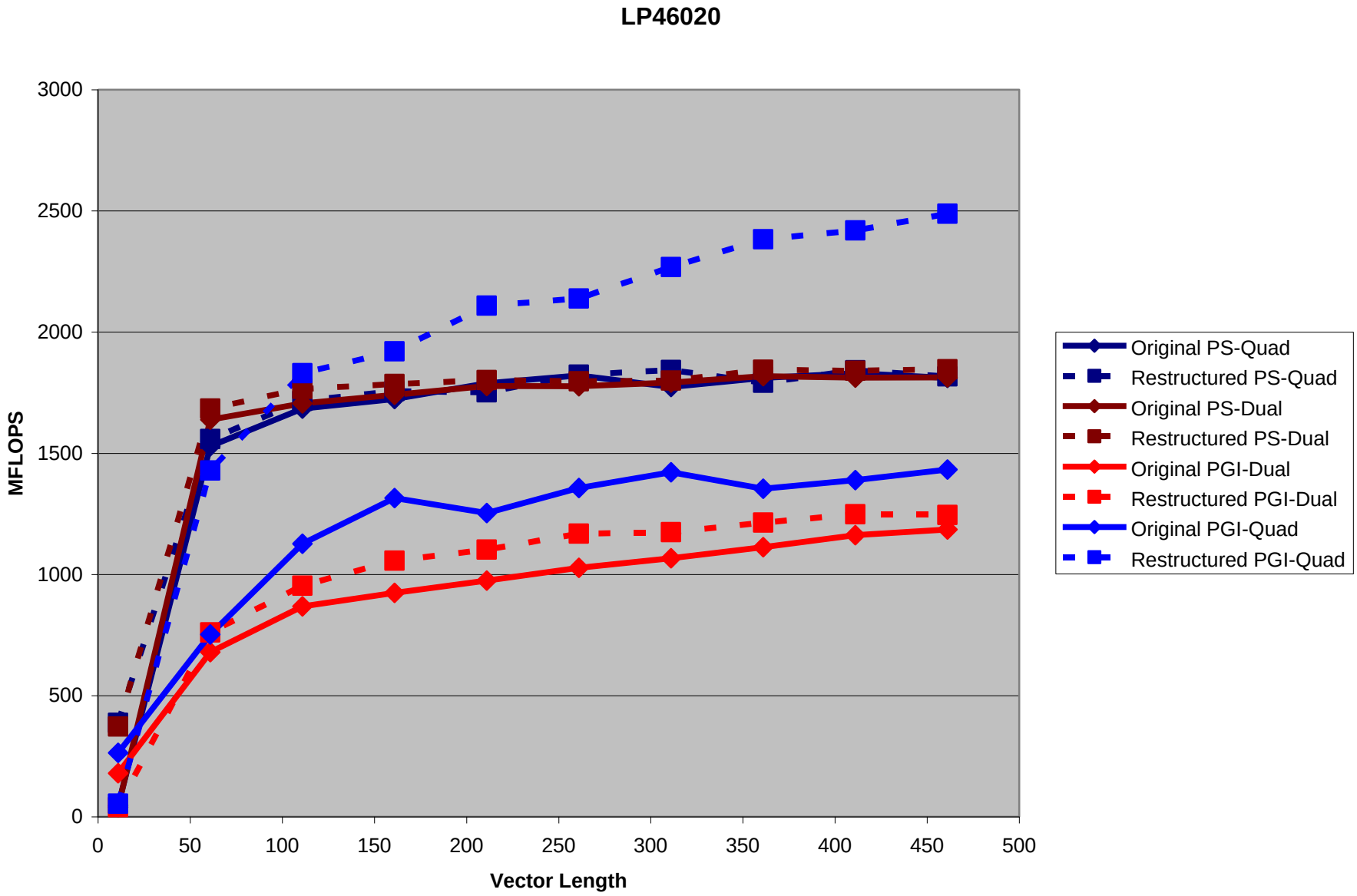
```
(   68) C        THE RESTRUCTURED
(   69)
(   70)         DO 46021 I = 1, N
(   71)           A(I,1) = B(I,1) * C(1,1) + B(I,2) * C(2,1)
(   72)       *          + B(I,3) * C(3,1) + B(I,4) * C(4,1)
(   73)           A(I,2) = B(I,1) * C(1,2) + B(I,2) * C(2,2)
(   74)       *          + B(I,3) * C(3,2) + B(I,4) * C(4,2)
(   75)           A(I,3) = B(I,1) * C(1,3) + B(I,2) * C(2,3)
(   76)       *          + B(I,3) * C(3,3) + B(I,4) * C(4,3)
(   77)           A(I,4) = B(I,1) * C(1,4) + B(I,2) * C(2,4)
(   78)       *          + B(I,3) * C(3,4) + B(I,4) * C(4,4)
(   79) 46021 CONTINUE
(   80)
```

PGI

70, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop

Pathscale

(lp46020.f:70) Loop has too many loop invariants. Loop was not vectorized.


LP46020
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Ten Lessons from Quad Core

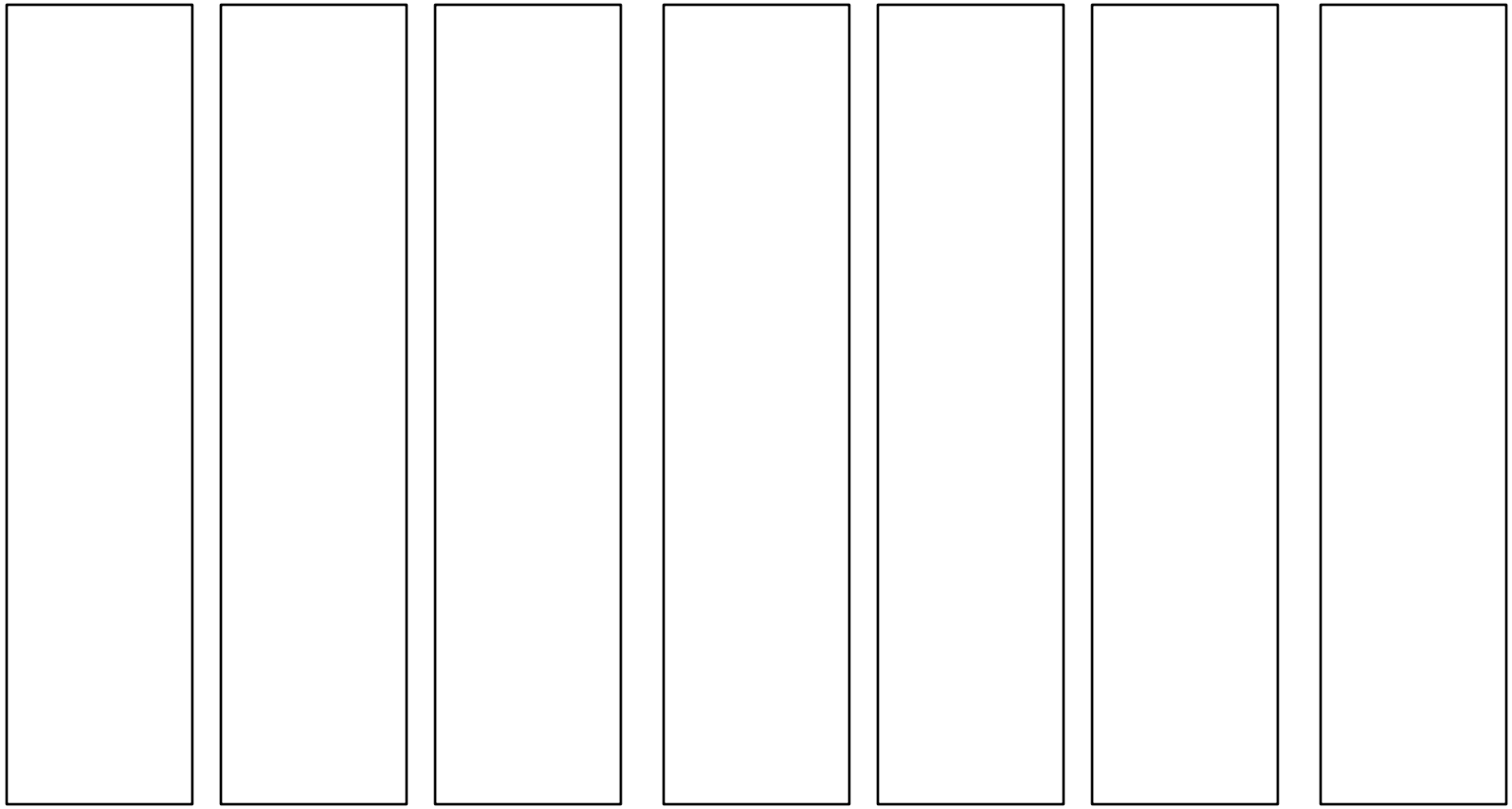
- Don’t Believe the Compiler
- Align Arrays correctly

Memory Banks

Fetch of A ==========➔

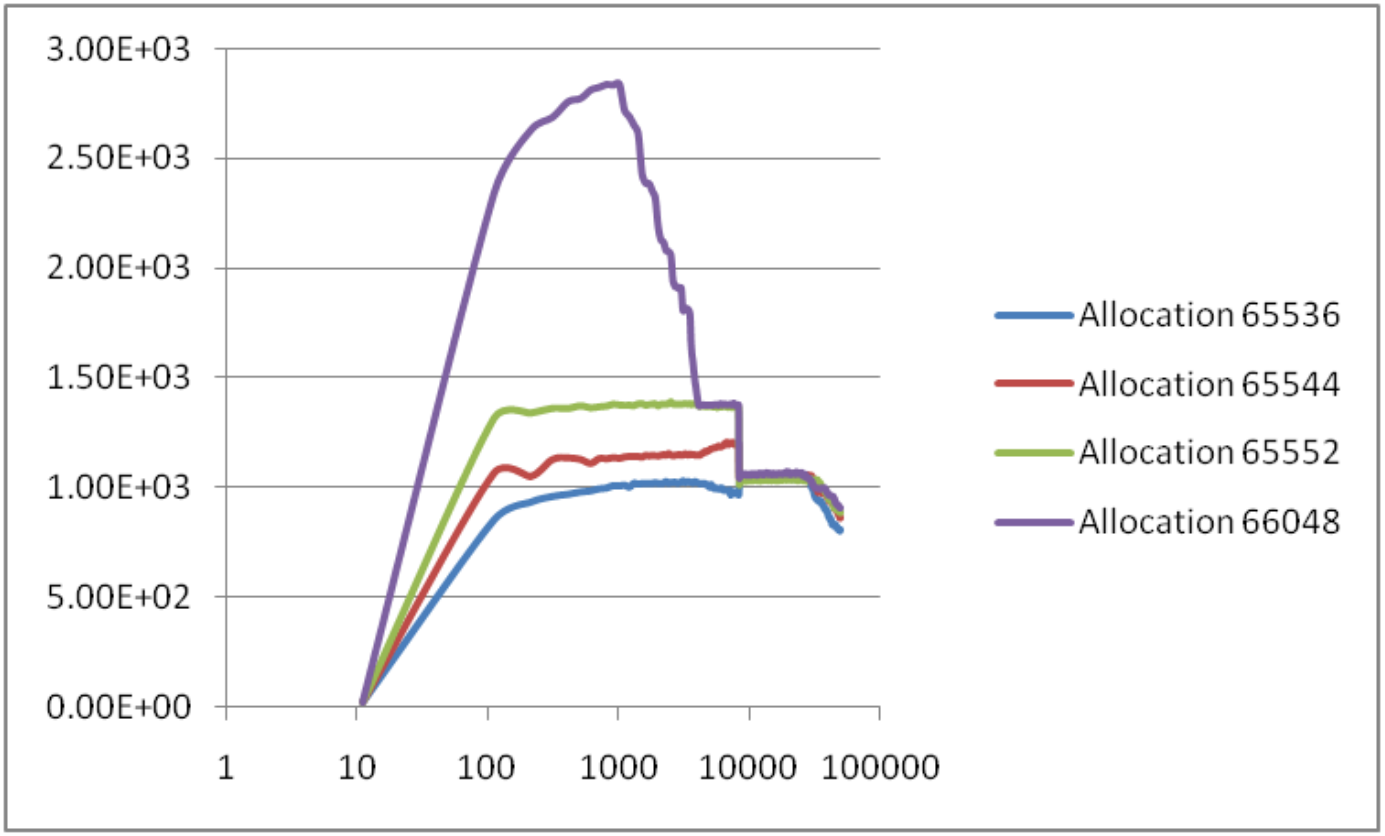
Fetch of B ==========➔

Fetch of C==========➔



Performance = F(Array Alignment)

Stream Triad (MFLOPS)
A(allocation),B(allocation),C(allocation)



Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize

Big Loop

```
(   52) C      THE ORIGINAL
(   53)
(   54)       DO 47020   J = 1, JMAX
(   55)        DO 47020  K = 1, KMAX
(   56)         DO 47020 I = 1, IMAX
(   57)          JP         = J + 1
(   58)          JR         = J - 1
(   59)          KP         = K + 1
(   60)          KR         = K - 1
(   61)          IP         = I + 1
(   62)          IR         = I - 1
(   63)          IF (J .EQ. 1)     GO TO 50
(   64)           IF( J .EQ. JMAX) GO TO 51
(   65)            XJ = ( A(I,JP,K) - A(I,JR,K) ) * DA2
(   66)            YJ = ( B(I,JP,K) - B(I,JR,K) ) * DA2
(   67)            ZJ = ( C(I,JP,K) - C(I,JR,K) ) * DA2
(   68)            GO TO 70
(   69)    50     J1 = J + 1
(   70)           J2 = J + 2
(   71)           XJ = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
(   72)           YJ = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
(   73)           ZJ = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
(   74)           GO TO 70
(   75)    51     J1 = J - 1
(   76)           J2 = J - 2
(   77)           XJ = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
(   78)           YJ = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
(   79)           ZJ = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
(   80)    70     CONTINUE
(   81)           IF (K .EQ. 1)     GO TO 52
(   82)            IF (K .EQ. KMAX) GO TO 53
(   83)             XK = ( A(I,J,KP) - A(I,J,KR) ) * DB2
(   84)             YK = ( B(I,J,KP) - B(I,J,KR) ) * DB2
(   85)             ZK = ( C(I,J,KP) - C(I,J,KR) ) * DB2
(   86)             GO TO 71
```

Big Loop

```
(   87)     52    K1 = K + 1
(   88)           K2 = K + 2
(   89)           XK = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
(   90)           YK = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
(   91)           ZK = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
(   92)           GO TO 71
(   93)     53    K1 = K - 1
(   94)           K2 = K - 2
(   95)           XK = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
(   96)           YK = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
(   97)           ZK = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
(   98)     71    CONTINUE
(   99)           IF (I .EQ. 1)     GO TO 54
(  100)            IF (I .EQ. IMAX) GO TO 55
(  101)              XI = ( A(IP,J,K) - A(IR,J,K) ) * DC2
(  102)              YI = ( B(IP,J,K) - B(IR,J,K) ) * DC2
(  103)              ZI = ( C(IP,J,K) - C(IR,J,K) ) * DC2
(  104)              GO TO 60
(  105)     54    I1 = I + 1
(  106)           I2 = I + 2
(  107)           XI = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
(  108)           YI = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
(  109)           ZI = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
(  110)           GO TO 60
(  111)     55    I1 = I - 1
(  112)           I2 = I - 2
(  113)           XI = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
(  114)           YI = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
(  115)           ZI = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
(  116)     60    CONTINUE
(  117)           DINV     = XJ * YK * ZI  +  YJ * ZK * XI  +  ZJ * XK * YI
(  118)       *              - XJ * ZK * YI  -  YJ * XK * ZI  -  ZJ * YK * XI
(  119)           D(I,J,K) = 1. / (DINV + 1.E-20)
(  120) 47020 CONTINUE
(  121)
```

PGI

55, Invariant if transformation

Loop not vectorized: loop count too small

56, Invariant if transformation

Pathscale

Nothing

Re-Write

```
(  141) C      THE RESTRUCTURED
(  142)
(  143)       DO 47029 J = 1, JMAX
(  144)        DO 47029 K = 1, KMAX
(  145)
(  146)         IF(J.EQ.1)THEN
(  147)
(  148)         J1         = 2
(  149)         J2         = 3
(  150)         DO 47021 I = 1, IMAX
(  151)          VAJ(I) = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
(  152)          VBJ(I) = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
(  153)          VCJ(I) = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
(  154) 47021   CONTINUE
(  155)
(  156)         ELSE IF(J.NE.JMAX) THEN
(  157)
(  158)         JP         = J+1
(  159)         JR         = J-1
(  160)         DO 47022 I = 1, IMAX
(  161)          VAJ(I) = ( A(I,JP,K) - A(I,JR,K) ) * DA2
(  162)          VBJ(I) = ( B(I,JP,K) - B(I,JR,K) ) * DA2
(  163)          VCJ(I) = ( C(I,JP,K) - C(I,JR,K) ) * DA2
(  164) 47022   CONTINUE
(  165)
(  166)         ELSE
(  167)
(  168)         J1         = JMAX-1
(  169)         J2         = JMAX-2
(  170)         DO 47023 I = 1, IMAX
(  171)          VAJ(I) = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
(  172)          VBJ(I) = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
(  173)          VCJ(I) = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
(  174) 47023   CONTINUE
(  175)
(  176)         ENDIF
```

Re-Write

```
(  178)          IF(K.EQ.1) THEN
(  179)
(  180)          K1          = 2
(  181)          K2          = 3
(  182)          DO 47024 I = 1, IMAX
(  183)           VAK(I) = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
(  184)           VBK(I) = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
(  185)           VCK(I) = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
(  186) 47024    CONTINUE
(  187)
(  188)          ELSE IF(K.NE.KMAX)THEN
(  189)
(  190)          KP          = K + 1
(  191)          KR          = K - 1
(  192)          DO 47025 I = 1, IMAX
(  193)           VAK(I) = ( A(I,J,KP) - A(I,J,KR) ) * DB2
(  194)           VBK(I) = ( B(I,J,KP) - B(I,J,KR) ) * DB2
(  195)           VCK(I) = ( C(I,J,KP) - C(I,J,KR) ) * DB2
(  196) 47025    CONTINUE
(  197)
(  198)          ELSE
(  199)
(  200)          K1          = KMAX - 1
(  201)          K2          = KMAX - 2
(  202)          DO 47026 I = 1, IMAX
(  203)           VAK(I) = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
(  204)           VBK(I) = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
(  205)           VCK(I) = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
(  206) 47026    CONTINUE
(  207)          ENDIF
(  208)
```

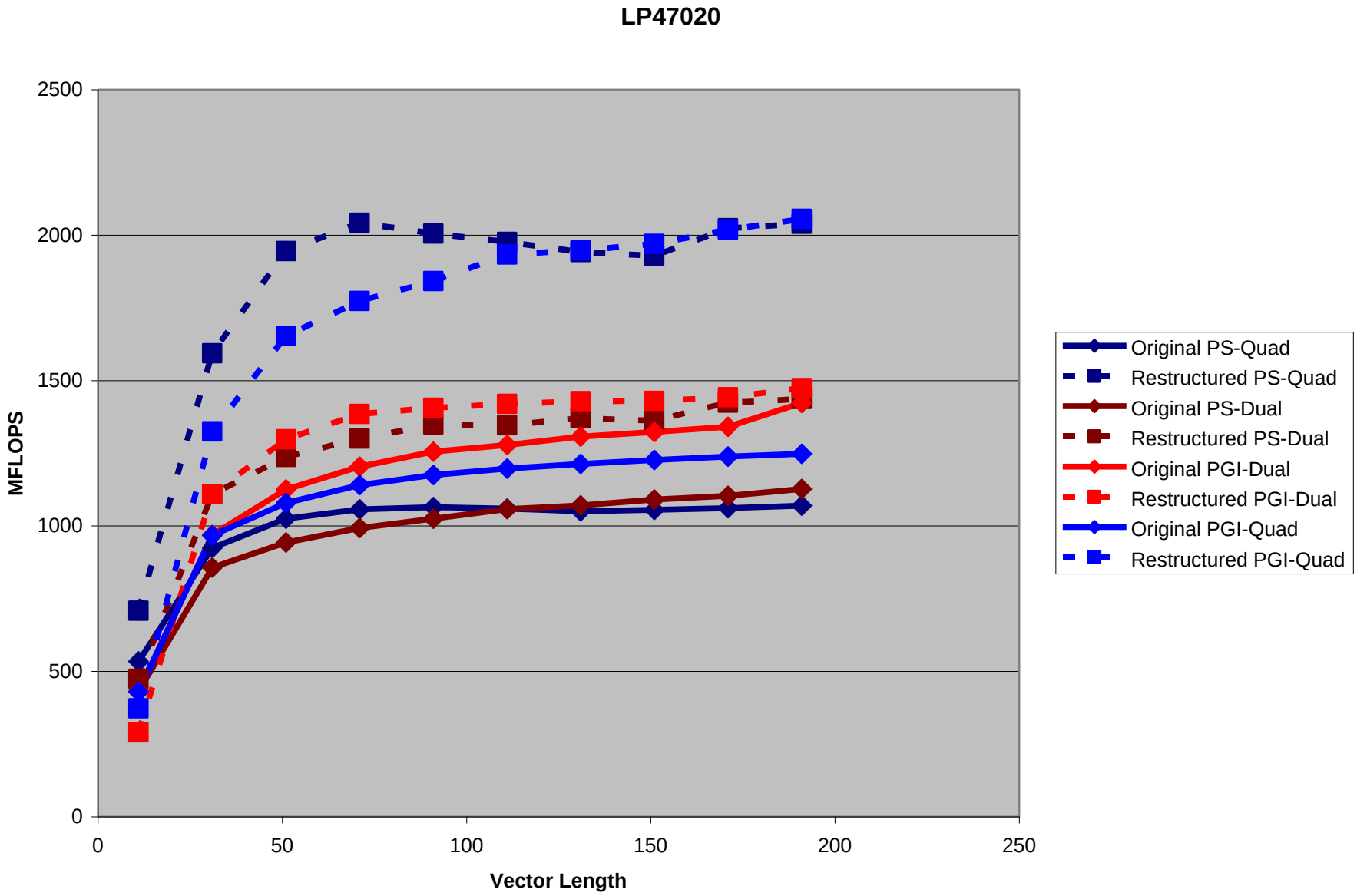
Re-Write

```
(  209)         I = 1
(  210)         I1         = 2
(  211)         I2         = 3
(  212)         VAI(I) = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
(  213)         VBI(I) = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
(  214)         VCI(I) = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
(  215)
(  216)         DO 47027 I = 2, IMAX-1
(  217)          IP         = I + 1
(  218)          IR         = I - 1
(  219)             VAI(I) = ( A(IP,J,K) - A(IR,J,K) ) * DC2
(  220)             VBI(I) = ( B(IP,J,K) - B(IR,J,K) ) * DC2
(  221)             VCI(I) = ( C(IP,J,K) - C(IR,J,K) ) * DC2
(  222) 47027    CONTINUE
(  223)
(  224)          I = IMAX
(  225)          I1          = IMAX - 1
(  226)          I2          = IMAX - 2
(  227)          VAI(I) = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
(  228)          VBI(I) = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
(  229)          VCI(I) = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
(  230)
(  231)          DO 47028 I = 1, IMAX
(  232)           DINV = VAJ(I) * VBK(I) * VCI(I) + VBJ(I) * VCK(I) * VAI(I)
(  233)       1         + VCJ(I) * VAK(I) * VBI(I) - VAJ(I) * VCK(I) * VBI(I)
(  234)       2         - VBJ(I) * VAK(I) * VCI(I) - VCJ(I) * VBK(I) * VAI(I)
(  235)             D(I,J,K) = 1. / (DINV + 1.E-20)
(  236) 47028    CONTINUE
(  237) 47029 CONTINUE

(  238)
```

PGI

144, Invariant if transformation
Loop not vectorized: loop count too small
150, Generated 3 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
160, Generated 4 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 6 prefetch instructions for this loop
Generated vector sse code for inner loop
o o o

Pathscale
(lp47020.f:132) LOOP WAS VECTORIZED.
(lp47020.f:150) LOOP WAS VECTORIZED.
(lp47020.f:160) LOOP WAS VECTORIZED.
(lp47020.f:170) LOOP WAS VECTORIZED.
(lp47020.f:182) LOOP WAS VECTORIZED.
(lp47020.f:192) LOOP WAS VECTORIZED.
(lp47020.f:202) LOOP WAS VECTORIZED.
(lp47020.f:216) LOOP WAS VECTORIZED.
(lp47020.f:231) LOOP WAS VECTORIZED.
(lp47020.f:248) LOOP WAS VECTORIZED.


LP47020
MFLOPS
2500
2000
1500
1000
500
0
0
50
100
150
200
250
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll

Traditional MATMUL

```
(    41) C      THE ORIGINAL
(    42)
(    43)       DO 46030 J  = 1, N
(    44)        DO 46030 I = 1, N
(    45)         A(I,J) = 0.
(    46) 46030 CONTINUE
(    47)
(    48)       DO 46031   K = 1, N
(    49)        DO 46031  J = 1, N
(    50)         DO 46031 I = 1, N
(    51)          A(I,J) = A(I,J) + B(I,K) * C(K,J)
(    52) 46031 CONTINUE
(    53)
```

PGI

```
   43, Loop not vectorized: contains call
    44, Memory zero idiom, loop replaced by memzero call
    48, Interchange produces reordered loop nest: 49, 48, 50
    50, Generated 3 alternate loops for the inner loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
```

Pathscale

```
(lp46030.f:44) LOOP WAS VECTORIZED.
(lp46030.f:44) LOOP WAS VECTORIZED.
(lp46030.f:50) Loop has too many loop invariants. Loop was not
vectorized.
(lp46030.f:50) LOOP WAS VECTORIZED.
(lp46030.f:50) LOOP WAS VECTORIZED.
(lp46030.f:50) LOOP WAS VECTORIZED.
```

Rewrite

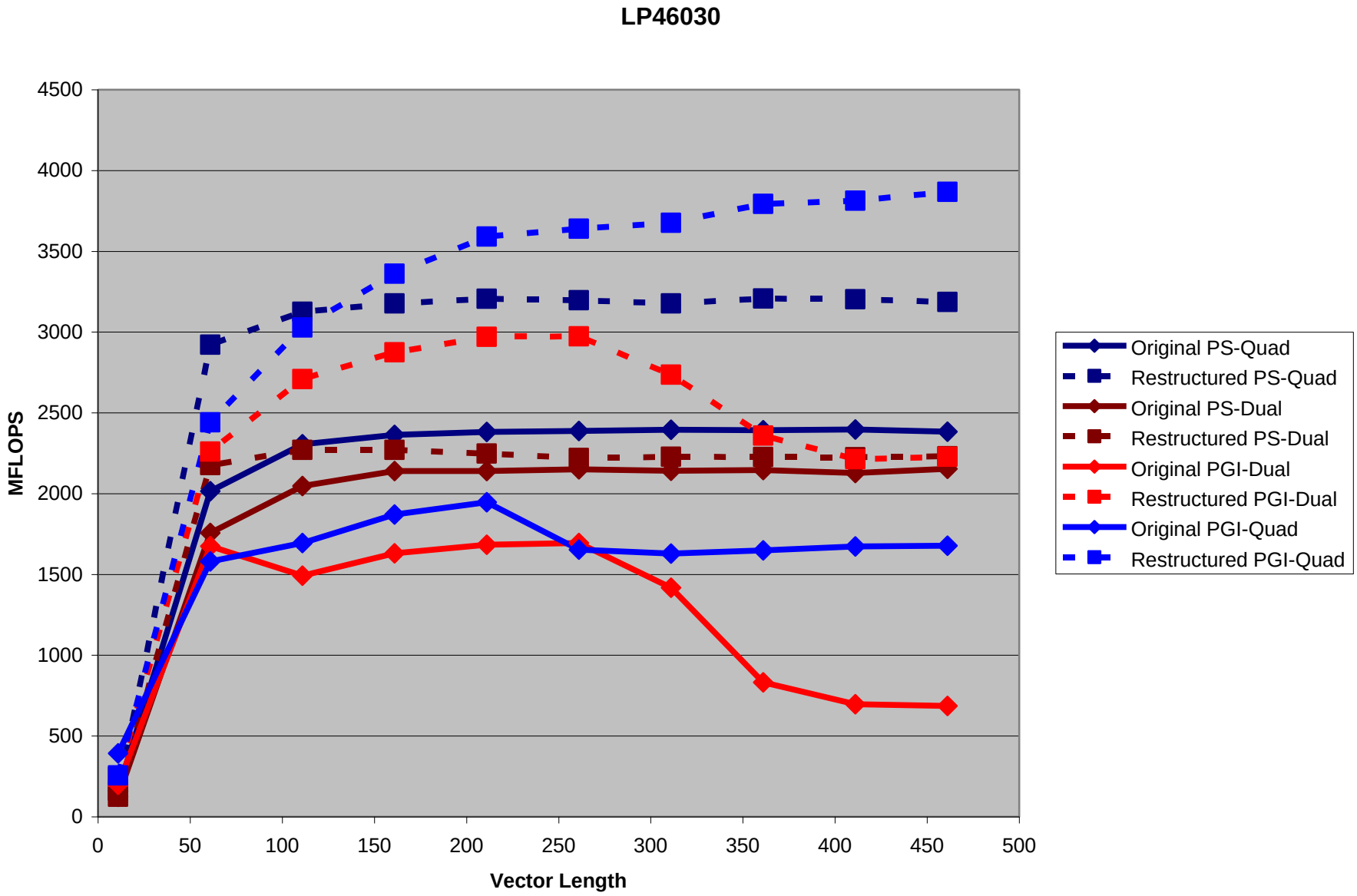
```
(   69) C       THE RESTRUCTURED
(   70)
(   71)       DO 46032  J = 1, N
(   72)        DO 46032 I = 1, N
(   73)         A(I,J)=0.
(   74) 46032 CONTINUE
(   75) C
(   76)       DO 46033   K = 1, N-5, 6
(   77)        DO 46033  J = 1, N
(   78)         DO 46033 I = 1, N
(   79)          A(I,J) = A(I,J) + B(I,K  ) * C(K  ,J)
(   80)      *                   + B(I,K+1) * C(K+1,J)
(   81)      *                   + B(I,K+2) * C(K+2,J)
(   82)      *                   + B(I,K+3) * C(K+3,J)
(   83)      *                   + B(I,K+4) * C(K+4,J)
(   84)      *                   + B(I,K+5) * C(K+5,J)
(   85) 46033 CONTINUE
(   86) C
(   87)       DO 46034  KK = K, N
(   88)        DO 46034  J = 1, N
(   89)         DO 46034 I = 1, N
(   90)          A(I,J) = A(I,J) + B(I,KK) * C(KK ,J)
(   91) 46034 CONTINUE
(   92)
```

Rewrite

```
PGI
    71, Loop not vectorized: contains call
    72, Memory zero idiom, loop replaced by memzero call
    78, Generated 3 alternate loops for the inner loop
        Generated vector sse code for inner loop
        Generated 7 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 7 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 7 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 7 prefetch instructions for this loop
    87, Interchange produces reordered loop nest: 88, 87, 89
    89, Generated 3 alternate loops for the inner loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
```

Rewrite

Pathscale
(lp46030.f:72) LOOP WAS VECTORIZED.
(lp46030.f:72) LOOP WAS VECTORIZED.
(lp46030.f:78) LOOP WAS VECTORIZED.
(lp46030.f:78) LOOP WAS VECTORIZED.
(lp46030.f:89) Loop has too many loop invariants. Loop was not vectorized.
(lp46030.f:89) LOOP WAS VECTORIZED.
(lp46030.f:89) LOOP WAS VECTORIZED.
(lp46030.f:89) LOOP WAS VECTORIZED.


LP46030
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
3000
3500
4000
4500
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

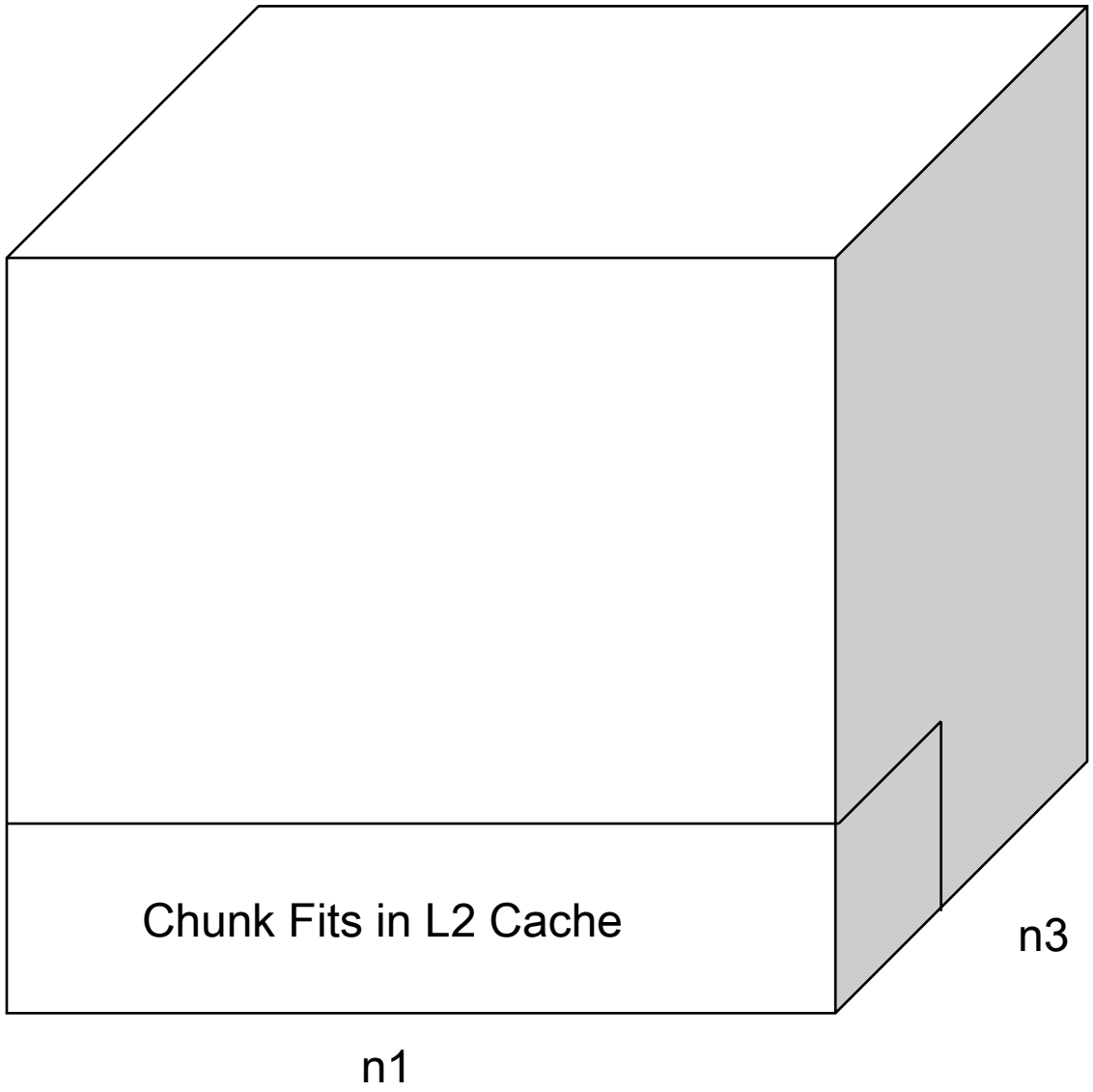
Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block

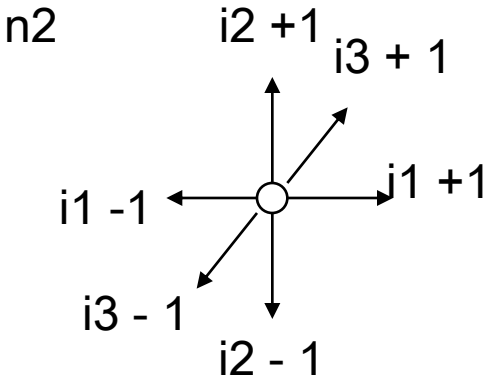
NPB MG routine RESID

```
      do i3=2,n3-1
           do i2=2,n2-1
              do i1=1,n1
                 u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
     >                   + u(i1,i2,i3-1) + u(i1,i2,i3+1)
                 u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
     >                   + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
              enddo
              do i1=2,n1-1
                 r(i1,i2,i3) = v(i1,i2,i3)
     >                         - a(0) * u(i1,i2,i3)
     >                         - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
     >                         - a(3) * ( u2(i1-1) + u2(i1+1) )
              enddo
           enddo
      enddo
```

```
==========================================================================
USER / resid_
--------------------------------------------------------------------------
  Time%                                              42.4%
  Time                                           12.397761
  Imb.Time                                        0.000370
  Imb.Time%                                           0.0%
  Calls                                                340
  PAPI_L1_DCA                 2719.188M/sec     33711498004 ops
  DC_L2_REFILL_MOESI            79.644M/sec       987402929 ops
  DC_SYS_REFILL_MOESI            4.059M/sec        50318116 ops
  BU_L2_REQ_DC                 129.172M/sec      1601429574 req
  User time                     12.398 secs      32233848320 cycles
  Utilization rate                                    100.0%
  L1 Data cache misses          83.703M/sec      1037721045 misses
  LD & ST per D1 miss                                 32.49 ops/miss
  D1 cache hit ratio                                  96.9%
  LD & ST per D2 miss                                669.97 ops/miss
  D2 cache hit ratio                                  96.9%
  L2 cache hit ratio                                  95.2%
  Memory to D1 refill            4.059M/sec        50318116 lines
  Memory to D1 bandwidth       247.723MB/sec      3220359424 bytes
  L2 to Dcache bandwidth      4861.112MB/sec     63193787456 bytes
==========================================================================
```



Entire Cube does not fit in L2 Cache
256*256*256*3 arrays
= 402 MBytes



Take data in chunks that Fit in L2 Cache
256*16*32*3 arrays
= 1 MBytes

Tiling for better Cache utilization

```
do i3block=2,n3-1,BLOCK3
      do i2block=2,n2-1,BLOCK2
      do i3=i3block,min(n3-1,i3block+BLOCK3-1)
         do i2=i2block,min(n2-1,i2block+BLOCK2-1)
            do i1=1, n1
               u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
     >                + u(i1,i2,i3-1) + u(i1,i2,i3+1)
               u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
     >                + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
            enddo
            do i1=1, n1
               r(i1,i2,i3) = v(i1,i2,i3)
     >                     - a(0) * u(i1,i2,i3)
     >                     - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
     >                     - a(3) * ( u2(i1-1) + u2(i1+1) )
            enddo
         enddo
       enddo
       enddo
       enddo
```

```
==========================================================================
USER / resid_
--------------------------------------------------------------------------
  Time%                                              36.3%
  Time                                           8.753226
  Imb.Time                                       0.000596
  Imb.Time%                                          0.0%
  Calls                                               340
  PAPI_L1_DCA                 3861.533M/sec   33800955933 ops
  DC_L2_REFILL_MOESI           116.399M/sec    1018867620 ops
  DC_SYS_REFILL_MOESI            2.755M/sec      24114222 ops
  BU_L2_REQ_DC                 161.490M/sec    1413560527 req
  User time                      8.753 secs   22758444048 cycles
  Utilization rate                                 100.0%
  L1 Data cache misses         119.154M/sec    1042981842 misses
  LD & ST per D1 miss                               32.41 ops/miss
  D1 cache hit ratio                                96.9%
  LD & ST per D2 miss                             1401.70 ops/miss
  D2 cache hit ratio                                98.3%
  L2 cache hit ratio                                97.7%
  Memory to D1 refill            2.755M/sec      24114222 lines
  Memory to D1 bandwidth       168.145MB/sec    1543310208 bytes
  L2 to Dcache bandwidth      7104.420MB/sec   65207527680 bytes
```

```
 do i3block=2,n3-1,BLOCK3
 do i2block=2,n2-1,BLOCK2
 do i3=i3block,min(n3-1,i3block+BLOCK3-1)
    do i2=i2block,min(n2-1,i2block+BLOCK2-1)
       do i1=1,n1
          u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
>                + u(i1,i2,i3-1) + u(i1,i2,i3+1)
          u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
>                + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
       enddo
       do i1=2,n1-1
          r(i1,i2,i3) = v(i1,i2,i3)
>                     - a(0) * u(i1,i2,i3)
>                     - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
>                     - a(3) * ( u2(i1-1) + u2(i1+1) )
       enddo
    enddo
 enddo
 enddo
 enddo
```

```
do i3block=2,n3-1,BLOCK3
 do i2block=2,n2-1,BLOCK2
 do i3=i3block,min(n3-1,i3block+BLOCK3-1)
    do i2=i2block,min(n2-1,i2block+BLOCK2-1)
       do i1=2,n1-1
          u21 = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
>               + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
          u21p1 = u(i1+1,i2-1,i3-1) + u(i1+1,i2+1,i3-1)
>               + u(i1+1,i2-1,i3+1) + u(i1+1,i2+1,i3+1)
          u21m1 = u(i1-1,i2-1,i3-1) + u(i1-1,i2+1,i3-1)
>               + u(i1-1,i2-1,i3+1) + u(i1-1,i2+1,i3+1)
          u11p1 = u(i1+1,i2-1,i3) + u(i1+1,i2+1,i3)
>               + u(i1+1,i2,i3-1) + u(i1+1,i2,i3+1)
          u11m1 = u(i1-1,i2-1,i3) + u(i1-1,i2+1,i3)
>               + u(i1-1,i2,i3-1) + u(i1-1,i2,i3+1)
          r(i1,i2,i3) = v(i1,i2,i3)
>                     - a(0) * u(i1,i2,i3)
>                     - a(2) * ( u21 + u11m1 + u11p1 )
>                     - a(3) * ( u21m1 + u21p1 )
       enddo
    enddo
 enddo
 enddo
 enddo
```

Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block
- Don’t stride through memory

Bad Striding

```
(    5)        COMMON A(8,8,IIDIM,8),B(8,8,iidim,8)


(   59)        DO 41090 K = KA, KE, -1
(   60)          DO 41090 J = JA, JE
(   61)            DO 41090 I = IA, IE
(   62)              A(K,L,I,J) = A(K,L,I,J) - B(J,1,i,k)*A(K+1,L,I,1)
(   63)       *    - B(J,2,i,k)*A(K+1,L,I,2) - B(J,3,i,k)*A(K+1,L,I,3)
(   64)       *    - B(J,4,i,k)*A(K+1,L,I,4) - B(J,5,i,k)*A(K+1,L,I,5)
(   65) 41090 CONTINUE
(   66)
```

PGI

59, Loop not vectorized: loop count too small
60, Interchange produces reordered loop nest: 61, 60
Loop unrolled 5 times (completely unrolled)
61, Generated vector sse code for inner loop

Pathscale

(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

Rewrite

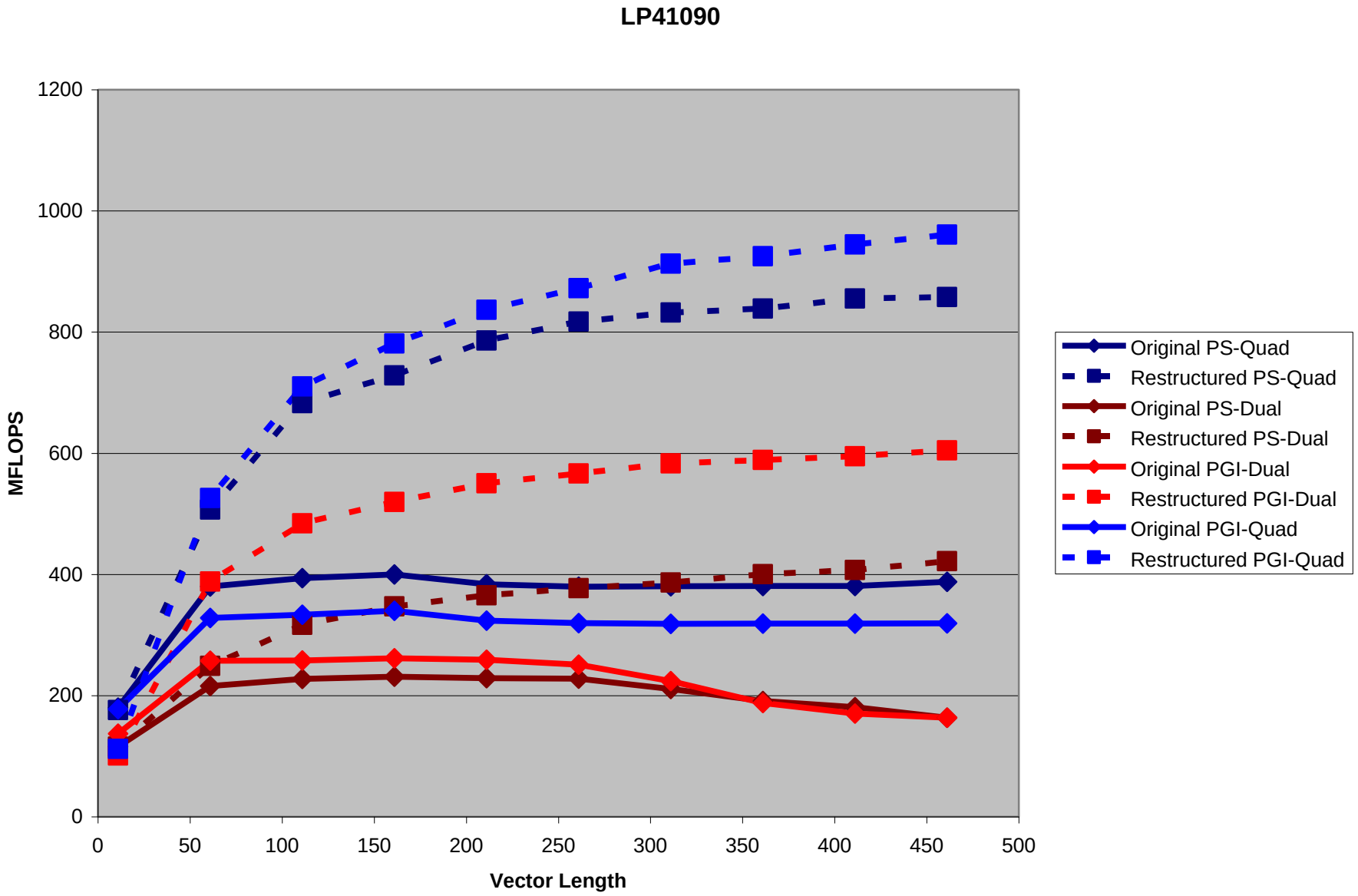
```
(     6)        COMMON AA(IIDIM,8,8,8),BB(IIDIM,8,8,8)


(    95)        DO 41091 K = KA, KE, -1
(    96)          DO 41091 J = JA, JE
(    97)            DO 41091 I = IA, IE
(    98)              AA(I,K,L,J) = AA(I,K,L,J) - BB(I,J,1,K)*AA(I,K+1,L,1)
(    99)       *      - BB(I,J,2,K)*AA(I,K+1,L,2) - BB(I,J,3,K)*AA(I,K+1,L,3)
(   100)       *      - BB(I,J,4,K)*AA(I,K+1,L,4) - BB(I,J,5,K)*AA(I,K+1,L,5)
(   101) 41091 CONTINUE
```

PGI

95, Loop not vectorized: loop count too small
96, Outer loop unrolled 5 times (completely unrolled)
97, Generated 3 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop

Pathscale

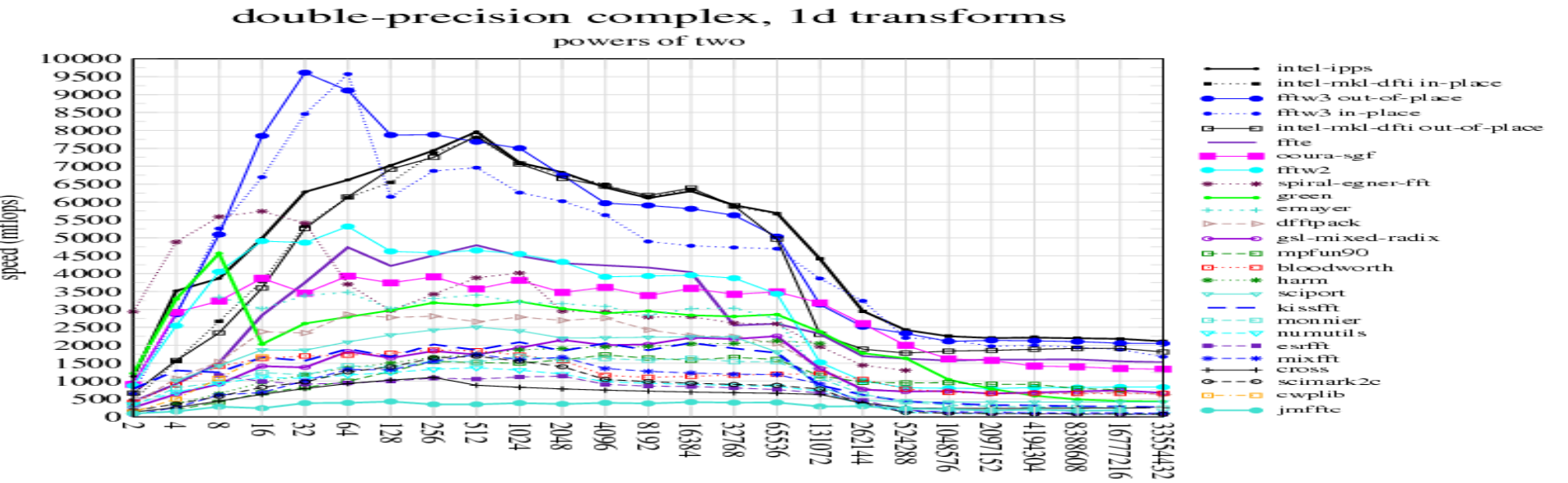
```
(lp41090.f:99) LOOP WAS VECTORIZED.
```

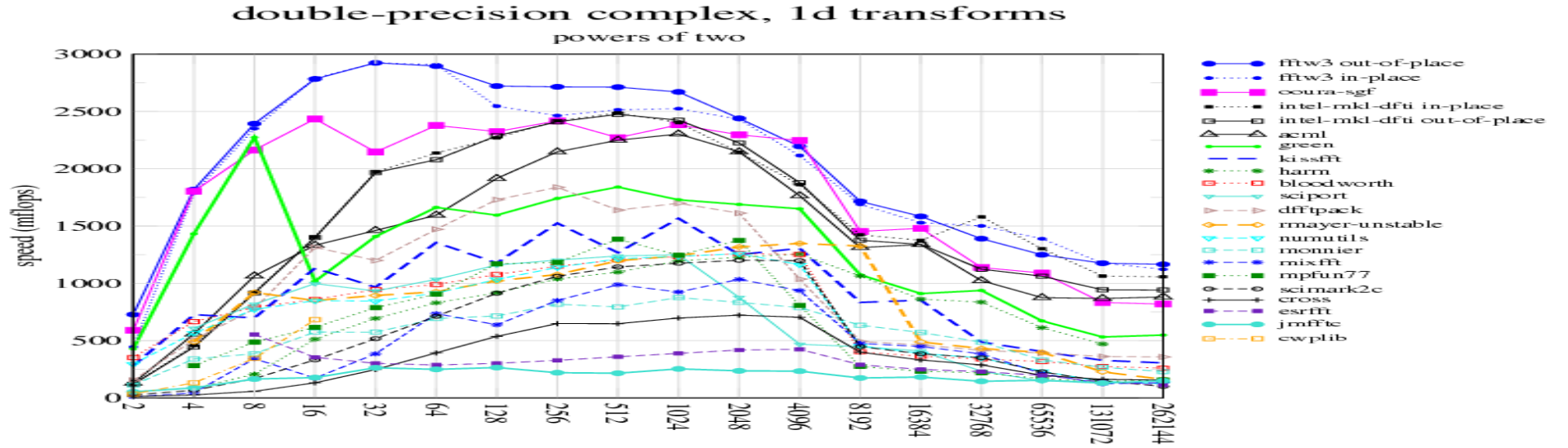

LP41090
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Ten Lessons from Quad Core

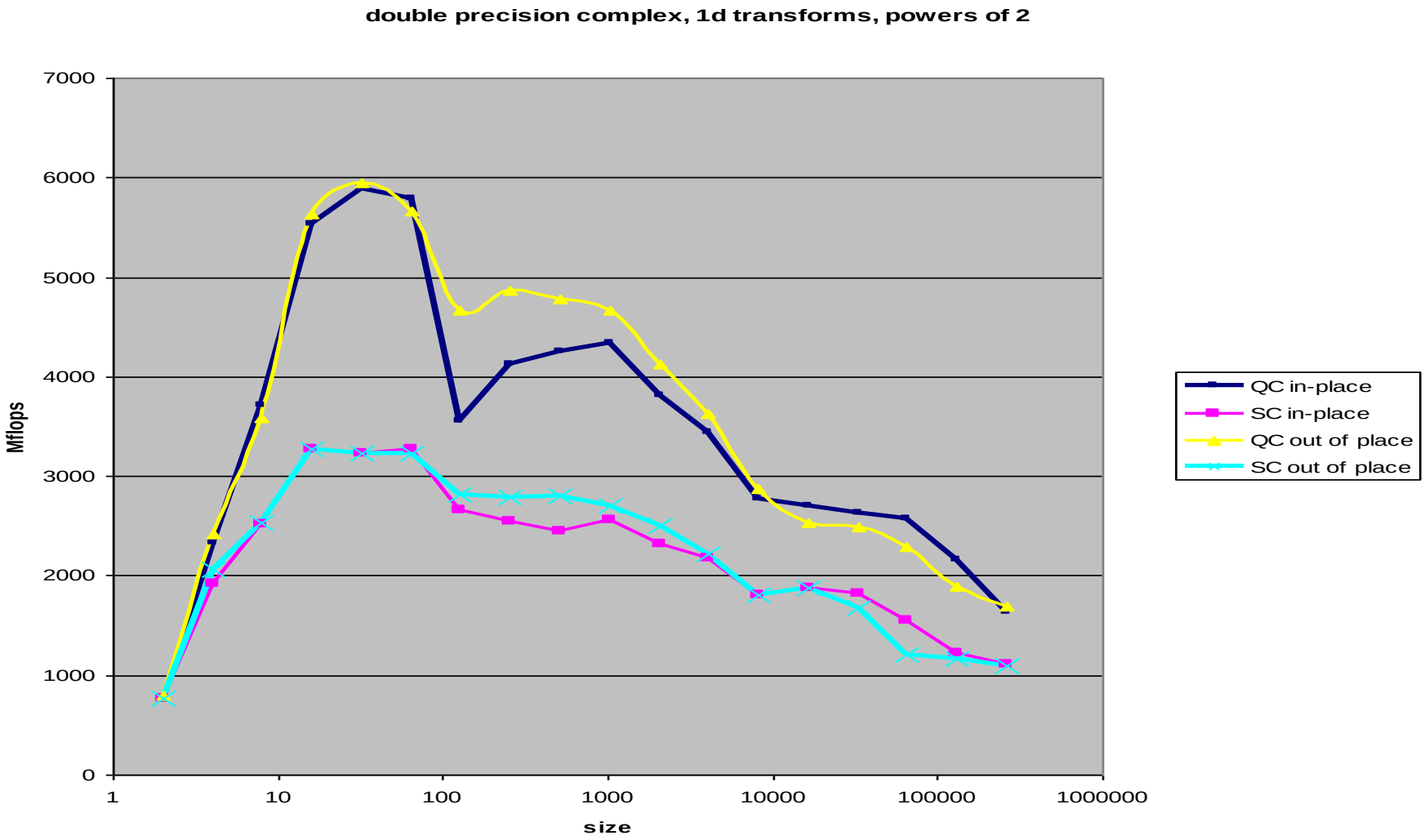
- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block
- Don’t stride through memory
- Use Quad Core enabled Libraries

FFTW benchmarks (fftw.org)





Problem solved on Barcelona?



Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block
- Don’t stride through memory
- Use Quad Core enabled Libraries
- Sustained Peak will only go down on Quad Core

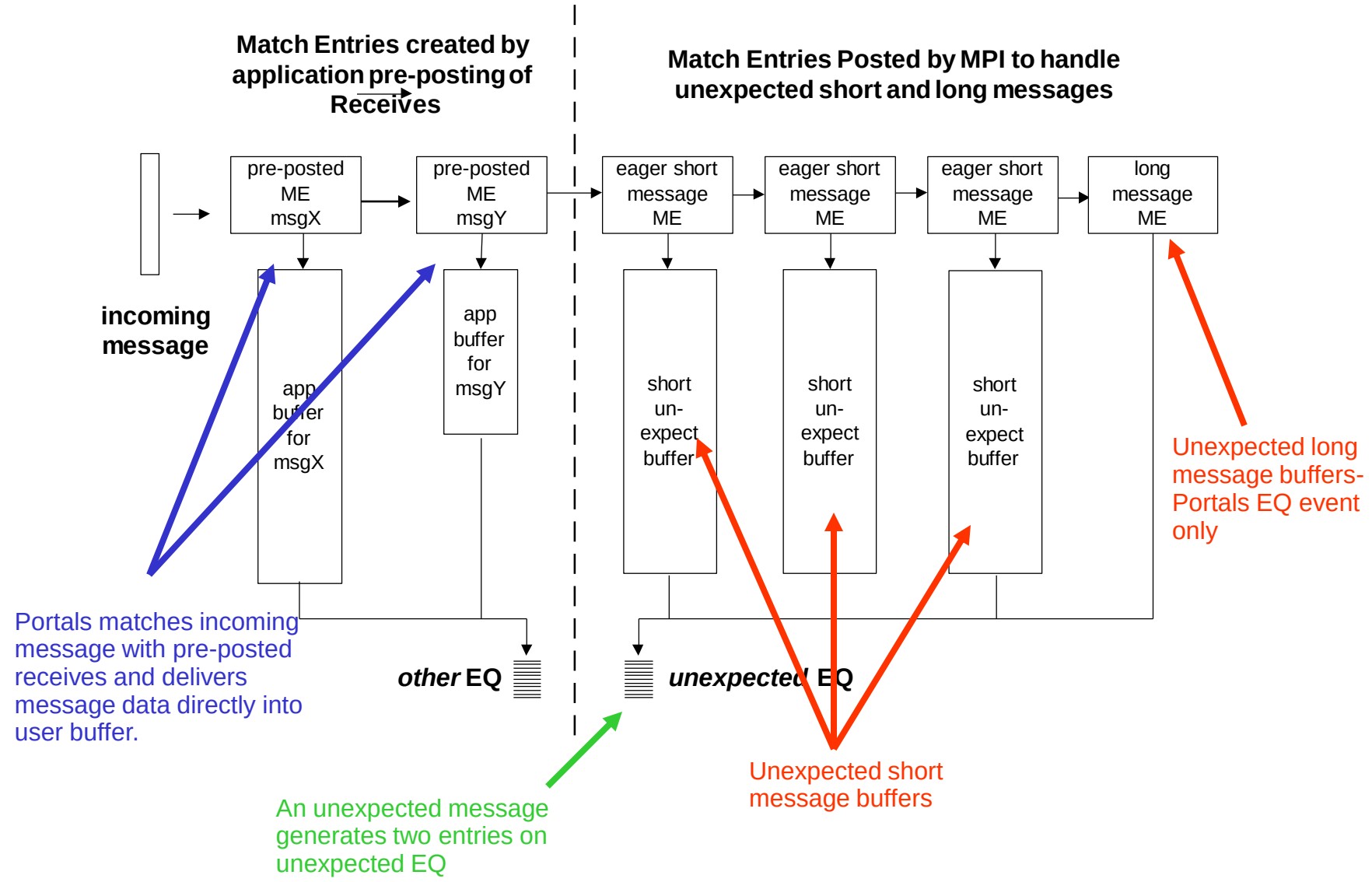
Leslie3d – SPEC benchmark - 1 Core

	Dual Core	Quad Core
Time	1045 Seconds	657 Seconds
GFLOPS	.667 GFLOPS	.869 GFLOPS
% of Peak	12.8 %	9.9%

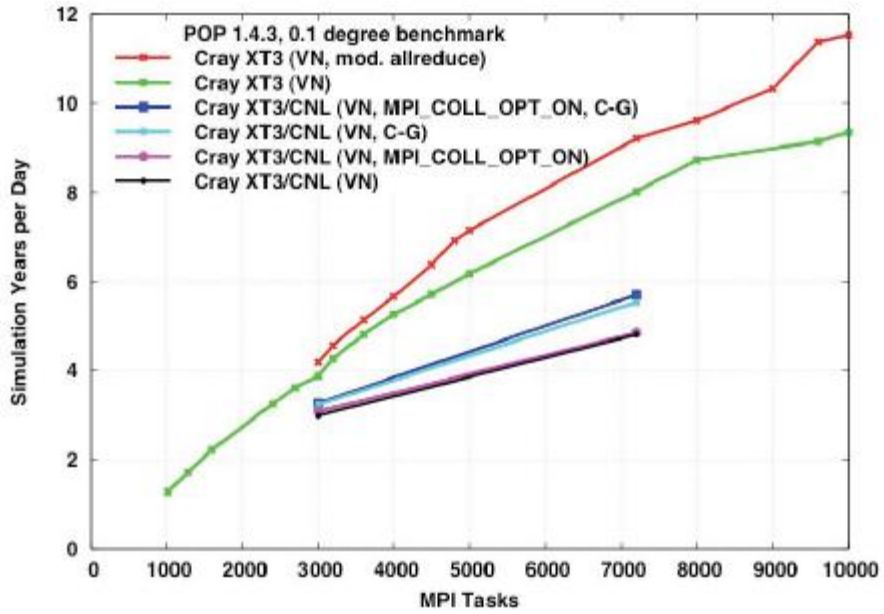
Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block
- Don’t stride through memory
- Use Quad Core enabled Libraries
- Sustained Peak will only go down on Quad Core
- Pre-post Receives

XT MPI – Receive Side



Suddenly last November ...



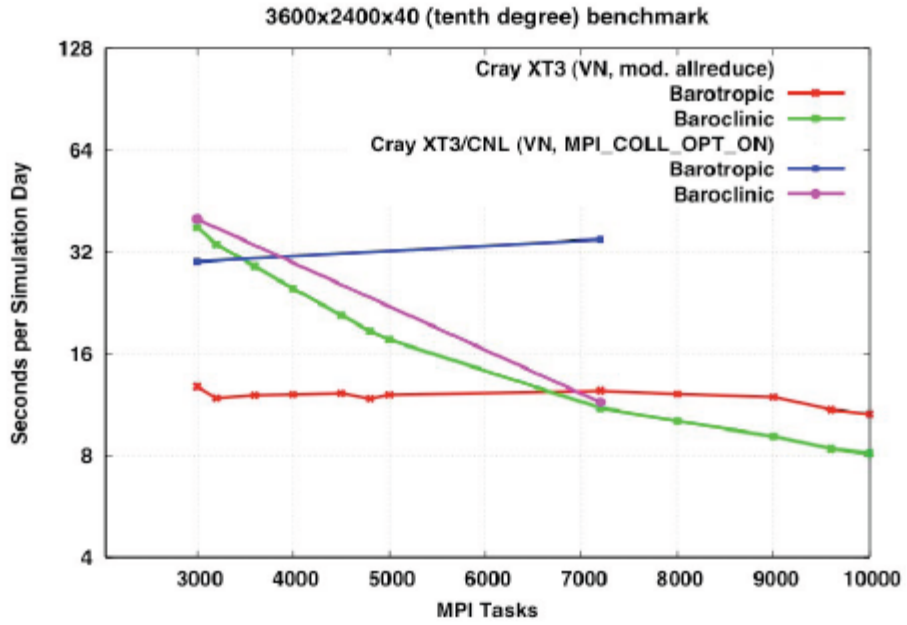
POP performance was not scaling well to large processor counts on Cray XT3 (using CNL). Even comparing Catamount results without C-G to CNL performance with C-G, CNL performance was much worse.

OAK RIDGE National Laboratory

10

Office of Science U.S. DEPARTMENT OF ENERGY

Catamount vs. CNL Phase Analysis



Performance difference is clearly in the Barotropic phase.

Hypotheses

1. First thought was that the MPI_COLL_OPT_ON version of MPI_ALLREDUCE was broken under CNL . Experiments with the "dual-core-aware" point-to-point implementation of allreduce, which was still in the code, did not improve performance. Further investigation showed that performance degradation was occurring when running in SN mode as well.
2. Second thought was that this was the dreaded OS jitter problem (for which allreduce is an excellent diagnostic tool). Added additional barriers and timers and found …

Barotropic PCG Solver Logic

```
! calculate (PC)r
WORK1 = R*A0R   ! use diag. precond.

! update conjugate direction vector s
WORK0 = R*WORK1
   call t_startf("pcg_global_sum_c0")
eta1 = global_sum(WORK0,RCALCT)
   call t_stopf("pcg_global_sum_c0")

S =  WORK1 + S*(eta1/eta0)

! compute As
   call t_startf("pcg_ninept_4_c")
call ninept_4(Q,A0,AN,AE,ANE,S)
   call t_stopf("pcg_ninept_4_c")
```

```
!  compute next solution and residual
eta0 = eta1
WORK0 = Q*S
   call t_barrierf("sync_pcg_glb_sum_c1")
   call t_startf("pcg_global_sum_c1")
eta1 = eta0/global_sum(WORK0,RCALCT)
   call t_stopf("pcg_global_sum_c1")

X = X + eta1*S
R = R - eta1*Q

!  test for convergence
...
```

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

Barotropic PCG Solver Performance

7200 MPI tasks, without timing barriers (process 0)

	Called	Wallclock	max	min
BAROTROPIC	1133	167.164764	1.315988	0.070891
pcg_global_sum_c0	139720	30.399334	1.170748	0.000111
pcg_ninept_4_c	139720	9.045540	0.002591	0.000033
pcg_global_sum_c1	139720	112.140366	0.006126	0.000062

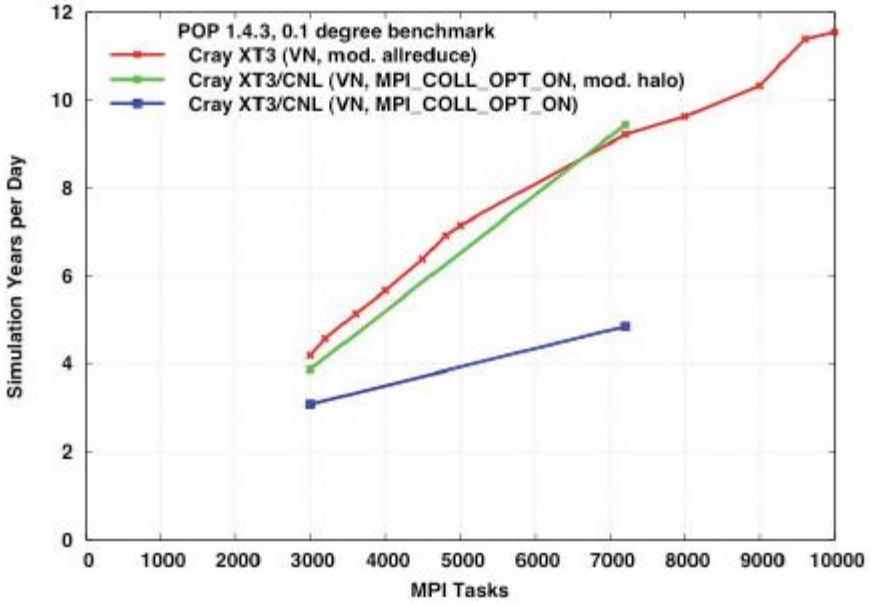
7200 MPI tasks, with timing barriers (process 0)

	Called	Wallclock	max	min
BAROTROPIC	1133	188.724838	0.342617	0.089528
pcg_global_sum_c0	139720	26.993826	0.002908	0.000098
pcg_ninept_4_c	139720	8.519553	0.001952	0.000037
sync_pcg_glb_sum_c1	139720	111.815437	0.003291	0.000075
pcg_global_sum_c1	139720	26.179775	0.002964	0.000063

Hypotheses II

- The timing barrier is capturing all of the performance "degradation", after which the allreduce behaves normally. If the problem were "OS jitter" occurring within the allreduce, then both the barrier and the allreduce would be impacted equally.
- The routine `ninept_4` is primarily a halo update, and the new hypothesis is that "ragged release" from the halo update is the source of the problems attributed to the allreduce. Next I tried alternate implementations of the halo update.

Modified Halo Update on XT3



Changing implementation of halo update, preposting receive requests, appears to eliminate performance degradation.

OAK RIDGE National Laboratory

18

Office of Science U.S. DEPARTMENT OF ENERGY

ninept_4 original

```
do j=jphys_b,jphys_e
    do i=iphys_b,iphys_e
       XOUT(i,j) = (9 pt. weighted sum)
    end do
end do

! fill buffers and send east-west
! boundary info
do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      buffer_east_snd(i)= ...
      buffer_west_snd(i)= ...
   end do
end do

call MPI_ISEND(buffer_east_snd, ...
call MPI_ISEND(buffer_west_snd, ...

! receive east-west boundary info and
! copy buffers into ghost cells
call MPI_RECV(buffer_west_rcv, ...
call MPI_RECV(buffer_east_rcv, ...

call MPI_WAITALL(2, …

do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      XOUT(n,j)          = ...
      XOUT(iphys_e+n,j) = ...
   end do
end do

! send north-south boundary info
call MPI_ISEND(XOUT(...
call MPI_ISEND(XOUT(...

! receive north-south boundary info
call MPI_RECV(XOUT(...
call MPI_RECV(XOUT(...
call MPI_WAITALL(2, …
```

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

ninept_4 modified

```
! Prepost receive requests
call MPI_IRECV(buffer_west_rcv, ...
call MPI_IRECV(buffer_east_rcv, ...
call MPI_IRECV(XOUT(...
call MPI_IRECV(XOUT(...

do j=jphys_b,jphys_e
    do i=iphys_b,iphys_e
       XOUT(i,j) = (9 pt. weighted sum)
    end do
end do

! fill buffers and send east-west
! boundary info
do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      buffer_east_snd(i)= ...
      buffer_west_snd(i)= ...
   end do
end do
```

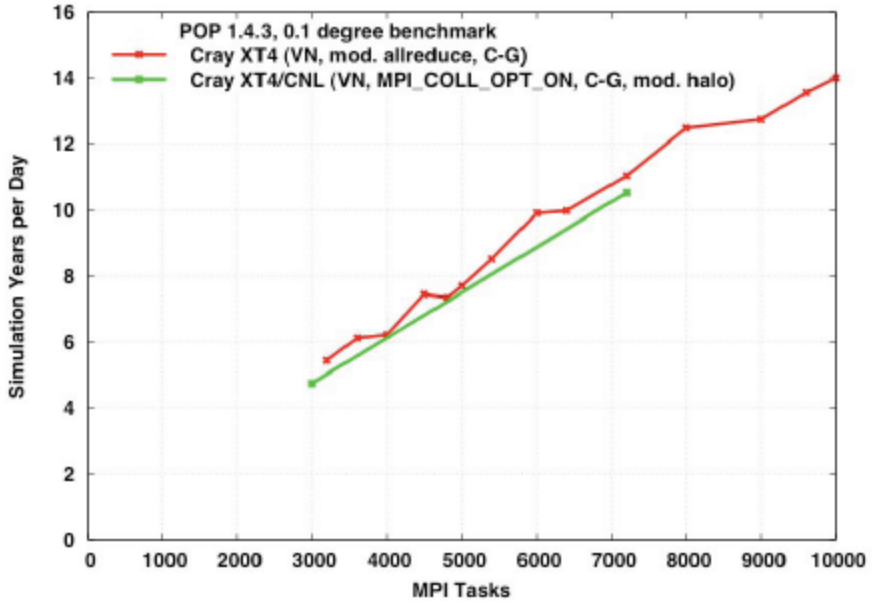
```
call MPI_ISEND(buffer_east_snd, ...
call MPI_ISEND(buffer_west_snd, ...

! receive east-west boundary info and
! copy buffers into ghost cells
call MPI_WAITALL(2, …

do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      XOUT(n,j)         = ...
      XOUT(iphys_e+n,j) = ...
   end do
end do

! send north-south boundary info
call MPI_ISEND(XOUT(...
call MPI_ISEND(XOUT(...

! receive north-south bddy info
call MPI_WAITALL(6, …
```

OAK RIDGE National Laboratory

17

Office of Science U.S. DEPARTMENT OF ENERGY

Modified Halo Update on XT4



Similar results hold with C-G algorithm and on Cray XT4.

OAK RIDGE National Laboratory

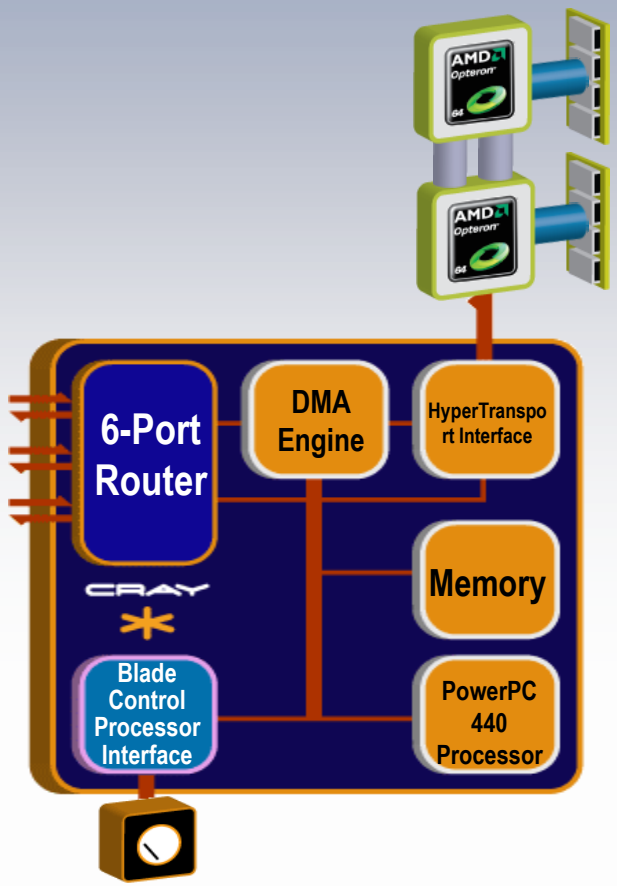
Office of Science U.S. DEPARTMENT OF ENERGY

Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block
- Don’t stride through memory
- Use Quad Core enabled Libraries
- Sustained Peak will only go down on Quad Core
- Pre-post Receives
- MPT 3.0 and Seastar + will finally help MPI_ALLTOALL

Cray SeaStar2+ Interconnect

- **New firmware will be released in early 2008 that will improve SeaStar performance**
- **Improvements:**
  - **Improved packet arbitration and aging algorithm lowers global latency**
  - **Using 4 virtual channels improves sustained global bandwidth**



Packet arbitration and aging Improvement	
PTRANS	4.60%
MPIFFT	12.4%
AllReduce	12.4%
AllToAll	36.3%

Multiple virtual channels Improvement	
PTRANS	10–25%
MPIFFT	25%
RandomRing bandwidth	>40%

Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block
- Don’t stride through memory
- Use Quad Core enabled Libraries
- Sustained Peak will only go down on Quad Core
- Pre-post Receives
- MPT 3.0 and Seastar + will finally help MPI_ALLTOALL
- Investigate Pre-fetching

Sparse CSR MV

Unroll q loop x times

```
do q = 1, n_rhs

    next_row_begin = row_start (1)

    do i = 1, n_rows

        row_begin      = next_row_begin
        next_row_begin = row_start (i +1)
        ip             = 0.0_wp

        do k = row_begin, next_row_begin - 1
            ip   = ip  +  values (k) * x (col_index (k), q)
        end do

        y (i, q) = ip

    end do
end do
```

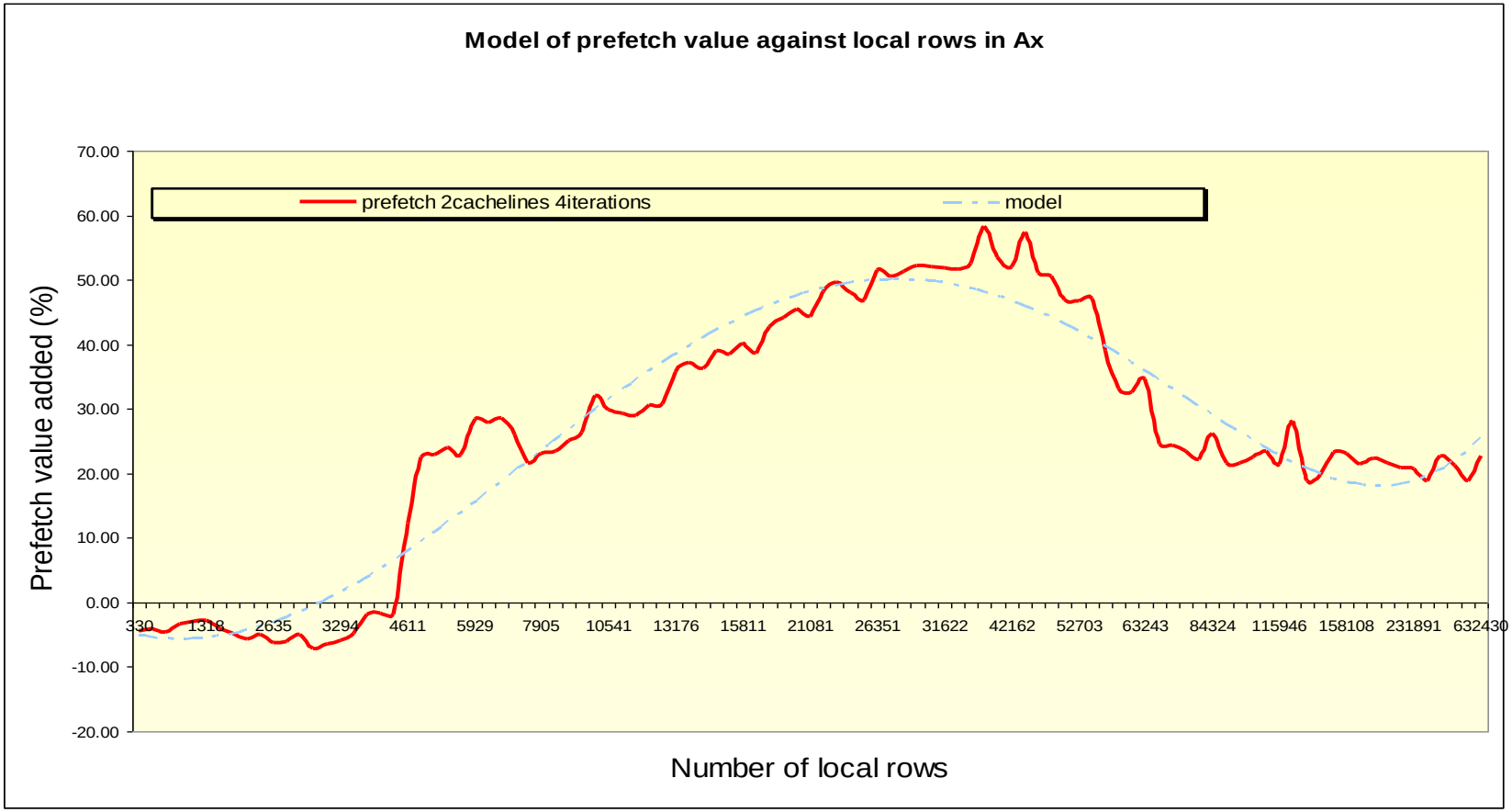
Should Scream on Granite!

Prefetch x cachelines of values and y cachelines of col_index, z iterations ahead

Unroll k loop x times

+ 3 choices of compilers
+ implicit unroll options
+ zero / one based indexing

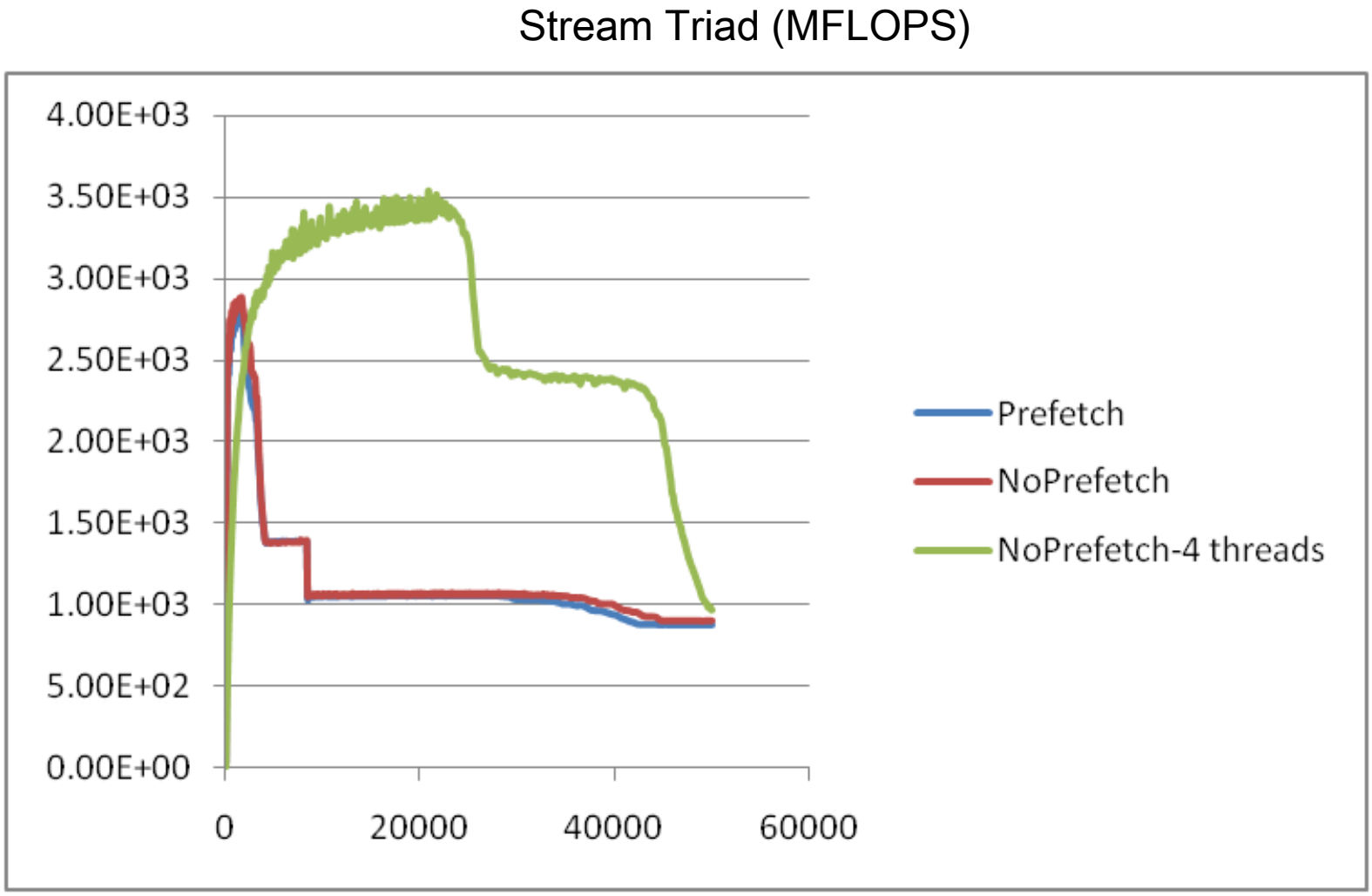
e.g. prefetch value



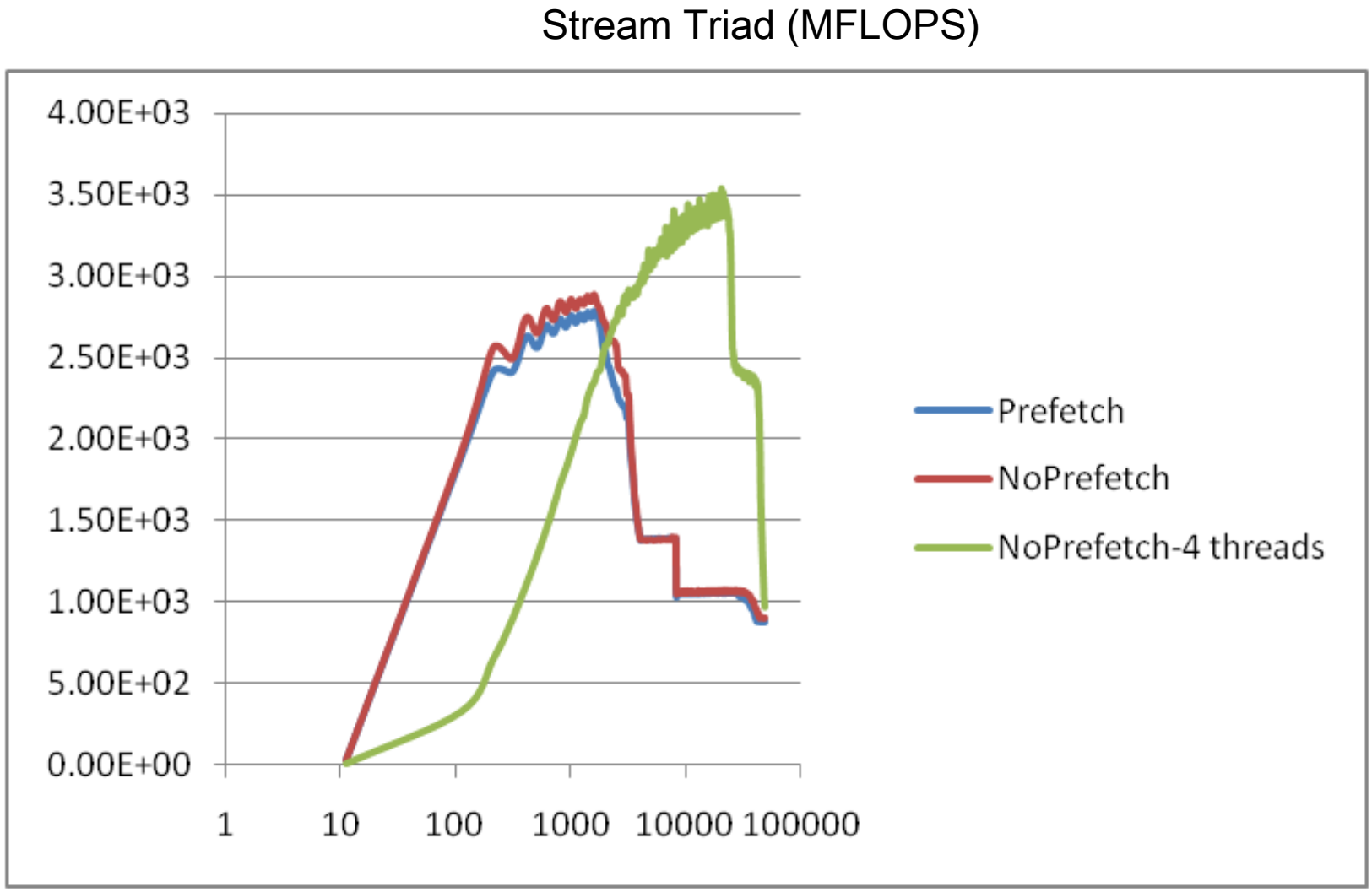
Ten Lessons from Quad Core

- Don’t Believe the Compiler
- Align Arrays correctly
- Vectorize
- Unroll
- Cache Block
- Don’t stride through memory
- Use Quad Core enabled Libraries
- Sustained Peak will only go down on Quad Core
- Pre-post Receives
- MPT 3.0 and Seastar + will finally help MPI_ALLTOALL
- Investigate Pre-fetching
- Investigate OpenMP

Performance = F(Cache Utilization)



Performance = F(Cache Utilization)



Who is counting?

- There were 12
- QUESTIONS????