Cray XT Optimization Basics

On-node Optimizations

Jeff Larkin
Cray Supercomputing Center of Excellence
larkin@cray.com

CRAY
THE SUPERCOMPUTER COMPANY

Before you make it run fast on the machine, make it run fast on the processor.

XT OPTIMIZATION RULE #1

Basic Optimizations

- Accessing Memory Efficiently
- Cache Blocking
- Loop Reordering
- Loop Unrolling
- Vectorization
- Logic Reordering
- Using Libraries

ACCESS MEMORY EFFICIENTLY

Indirect Addressing

```
(  300) C     FIVE OPERATIONS - TWO OPERANDS     RATIO = 5/2
(  301)
(  302)       DO 41012 I = 1, N
(  303)        Y(IY(I)) =  c0 + X(IX(I)) * (C1 + X(IX(I))
(  304)      *          * (C2 + X(IX(I))                 ))
(  305) 41012 CONTINUE
```

302, Loop unrolled 2 times

Contiguous Addressing

```
(  799)       DO 41033 I = 1, N
(  800)        Y(I) = c0 + X(I) * (C1 + X(I) * (C2 + X(I)
(  801)      *                  * (C3 + X(I)              )))
(  802) 41033 CONTINUE
```

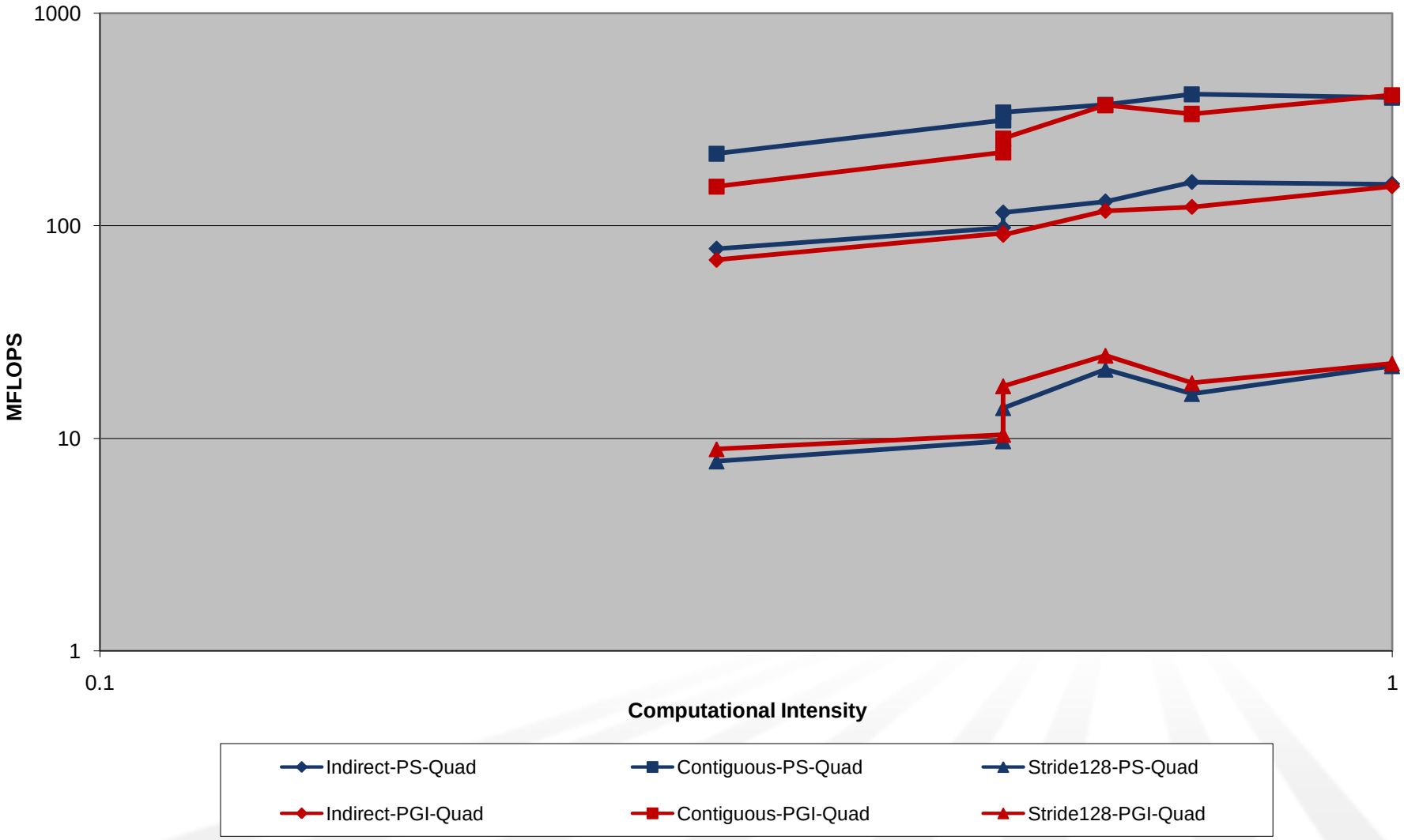
799, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop

Bad Stride Addressing

```
( 1239)        II=1
( 1240)
( 1241)        DO 41072 I = 1, N
( 1242)         Y(II) = c0 + X(II) * (C1 + X(II) * (C2 + X(II) ))
( 1243)         II = II + ISTRIDE
( 1244) 41072 CONTINUE
```

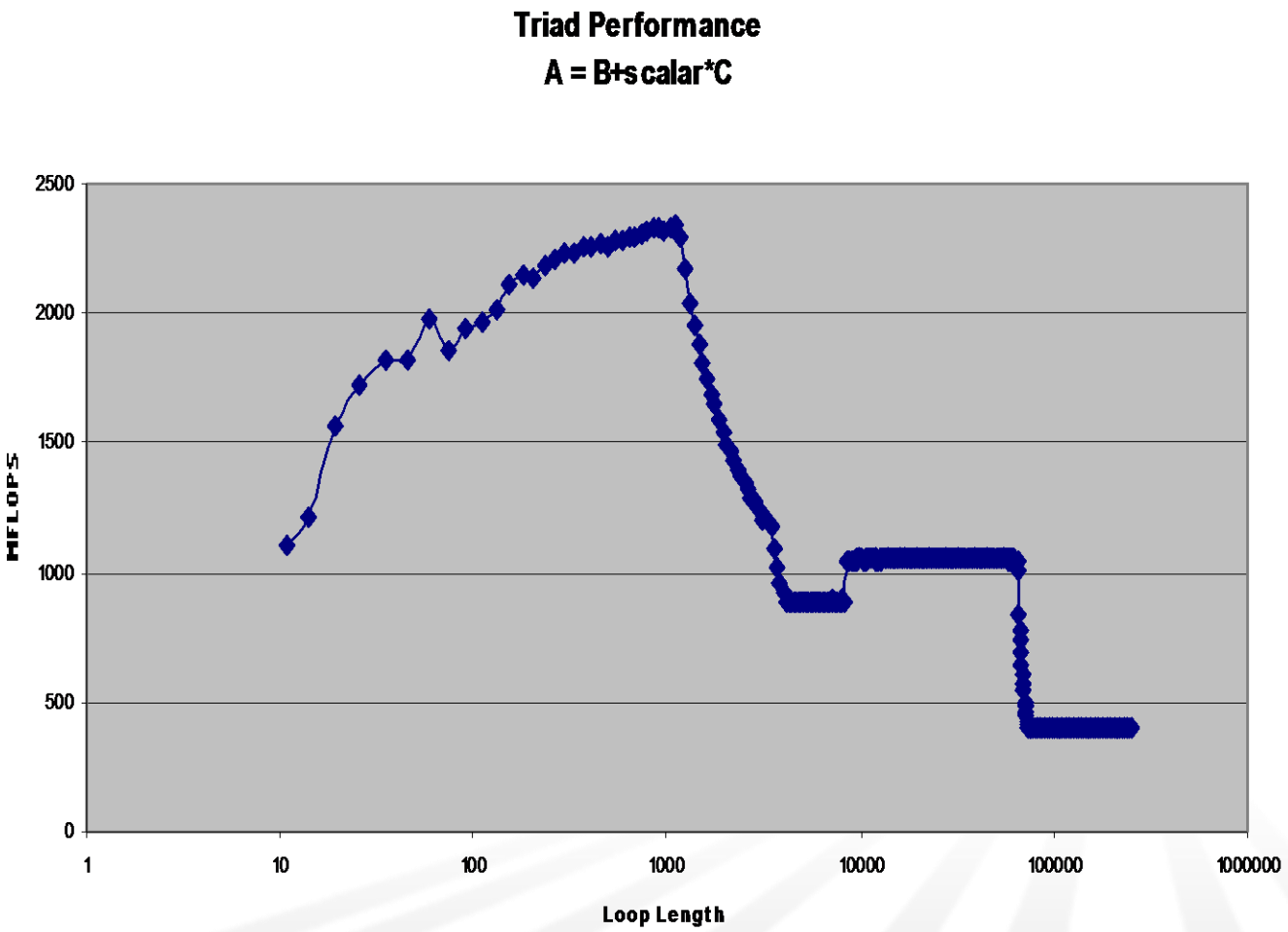
1241, Loop unrolled 1 times

Memory Access Patterns



CACHE BLOCKING

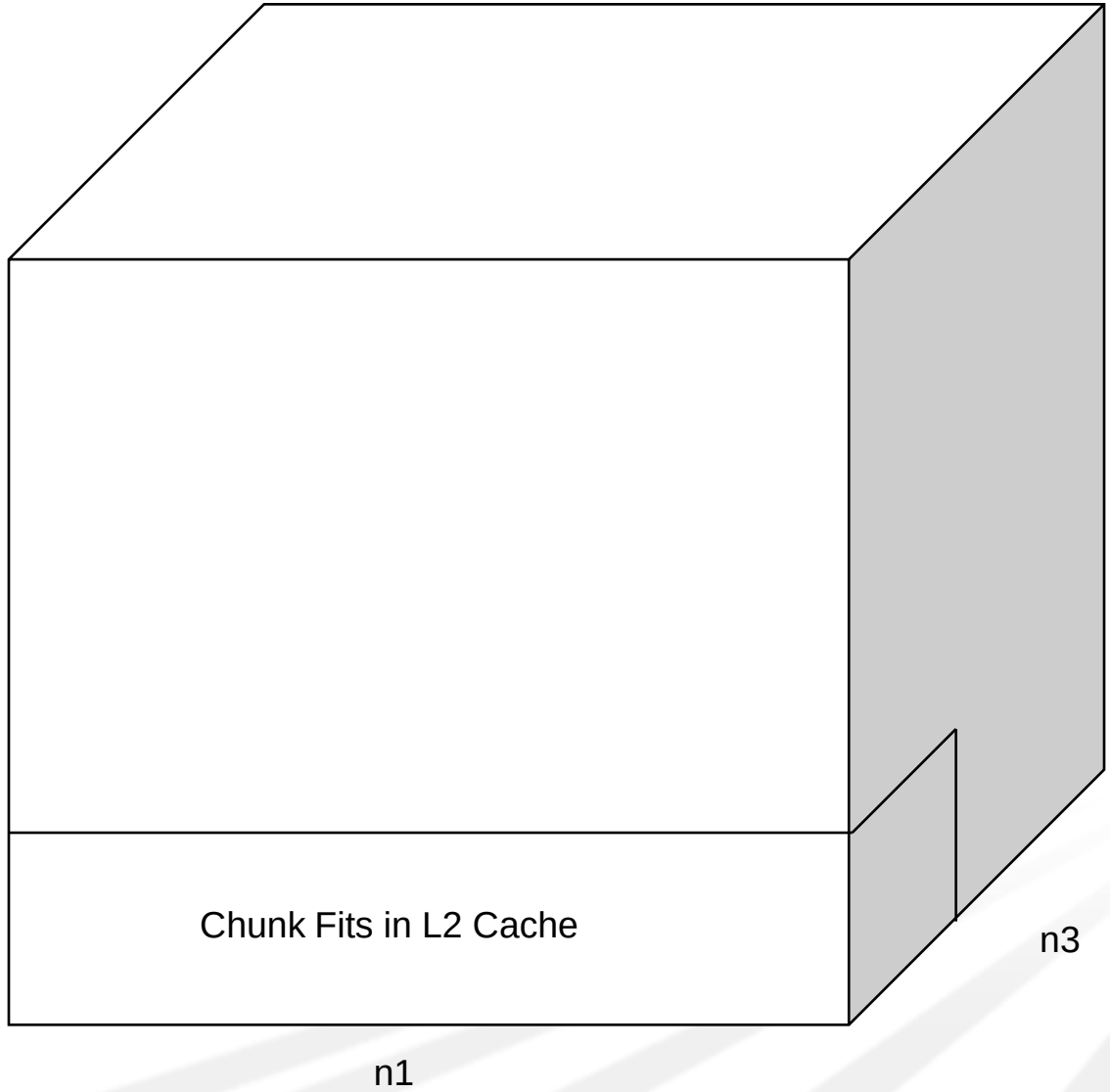
Why Block for Cache?



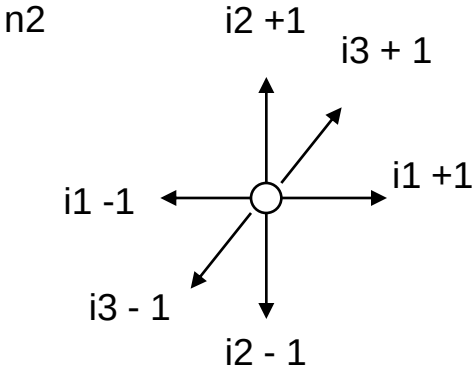
NPB MG routine RESID

```
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*16*3 arrays
= 512 KBytes

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 (12.397761)
  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 (669.97)
  D2 cache hit ratio                             98.3% (96.9%)
  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
```

LOOP REORDERING

Loop Reordering: Original

```
(   44) C       THE ORIGINAL
(   45)
(   46)        DO 44050 I = 1, N
(   47)         DO 44050 J = 1, N
(   48)          A(I,J) = 0.0
(   49)          DO 44050 K = 1, N
(   50)           A(I,J) = A(I,J) + B(I,K) * C(K,J)
(   51) 44050 CONTINUE
(   52)
```

PGI

49, Generated 2 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop

Pathscale

(lp44050.f:46) Loop has too many loop invariants. Loop was not vectorized.
(lp44050.f:46) LOOP WAS VECTORIZED.
(lp44050.f:46) LOOP WAS VECTORIZED.
(lp44050.f:46) LOOP WAS VECTORIZED.

Loop Reordering: Rewritten

```
(   77) C      THE RESTRUCTURED
(   78)
(   79)       DO 44051 J = 1, N
(   80)        DO 44051 I = 1, N
(   81)         A(I,J) = 0.0
(   82) 44051 CONTINUE
(   83)
(   84)       DO 44052 K = 1, N
(   85)        DO 44052 J = 1, N
(   86)         DO 44052 I = 1, N
(   87)          A(I,J) = A(I,J) + B(I,K) * C(K,J)
(   88) 44052 CONTINUE
(   89) C
```

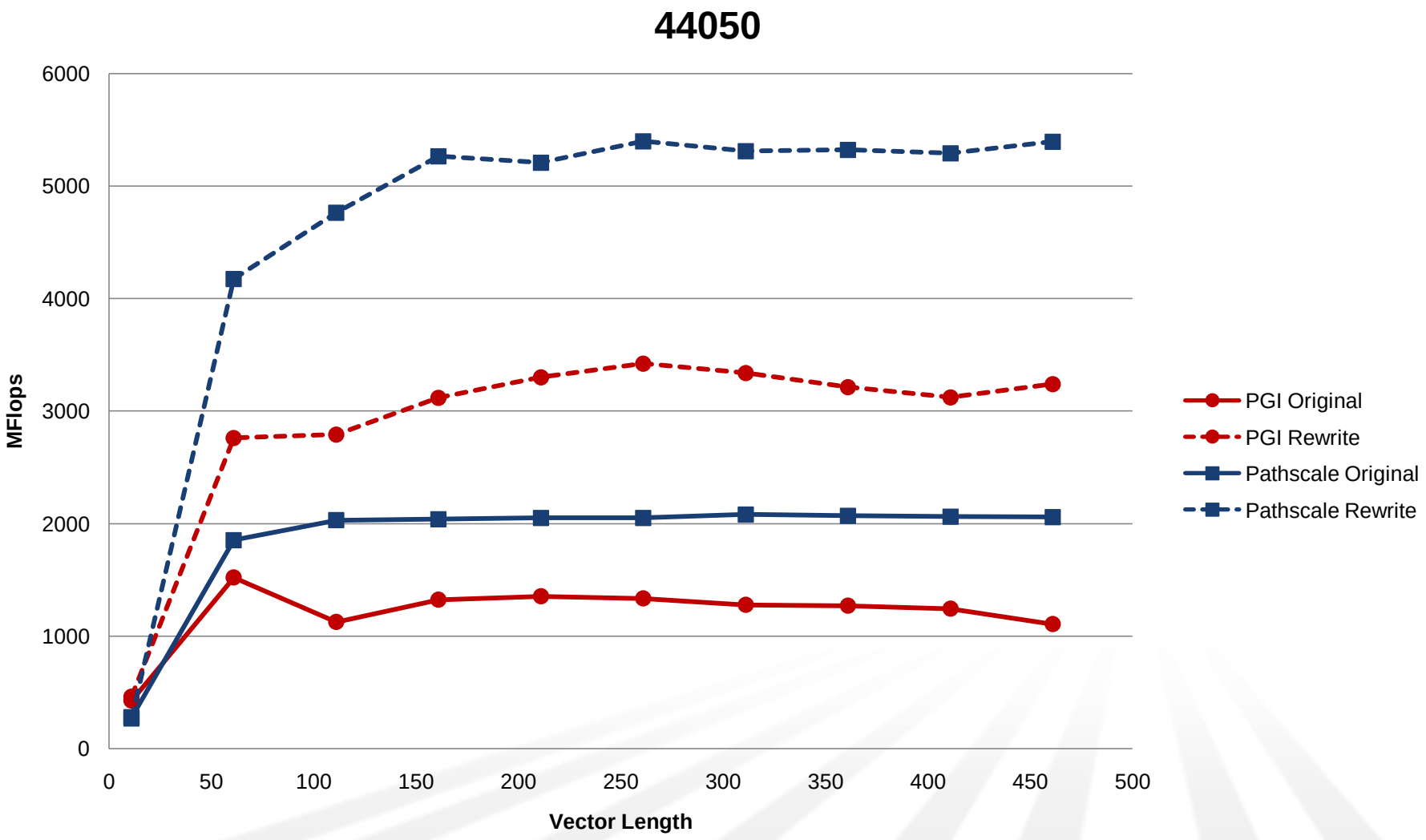
PGI

```
79, Loop not vectorized: contains call
80, Memory zero idiom, loop replaced by memzero call
84, Interchange produces reordered loop nest: 85, 84, 86
86, 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

```
(lp44050.f:80) LOOP WAS VECTORIZED.
(lp44050.f:80) LOOP WAS VECTORIZED.
(lp44050.f:86) Loop has too many loop invariants. Loop was not vectorized.
(lp44050.f:86) LOOP WAS VECTORIZED.
(lp44050.f:86) LOOP WAS VECTORIZED.
(lp44050.f:86) LOOP WAS VECTORIZED.
```

Loop Reordering: Results



Nested Loops

```
(   47)        DO 45020 I = 1, N
(   48)         F(I) = A(I) + .5
(   49)         DO 45020 J = 1, 10
(   50)          D(I,J) = B(J) * F(I)
(   51)          DO 45020 K = 1, 5
(   52)           C(K,I,J) = D(I,J) * E(K)
(   53) 45020 CONTINUE
```

PGI
49, Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop
Loop unrolled 2 times (completely unrolled)
Pathscale
(lp45020.f:48) LOOP WAS VECTORIZED.
(lp45020.f:48) Non-contiguous array "C(_BLNK__.0.0)" reference exists. Loop was not vectorized.

Rewrite

```
(   71)      DO 45021 I = 1,N
(   72)       F(I) = A(I) + .5
(   73) 45021 CONTINUE
(   74)
(   75)      DO 45022 J = 1, 10
(   76)       DO 45022 I = 1, N
(   77)        D(I,J) = B(J) * F(I)
(   78) 45022 CONTINUE
(   79)
(   80)      DO 45023 K = 1, 5
(   81)       DO 45023 J = 1, 10
(   82)        DO 45023 I = 1, N
(   83)         C(K,I,J) = D(I,J) * E(K)
(   84) 45023 CONTINUE
```

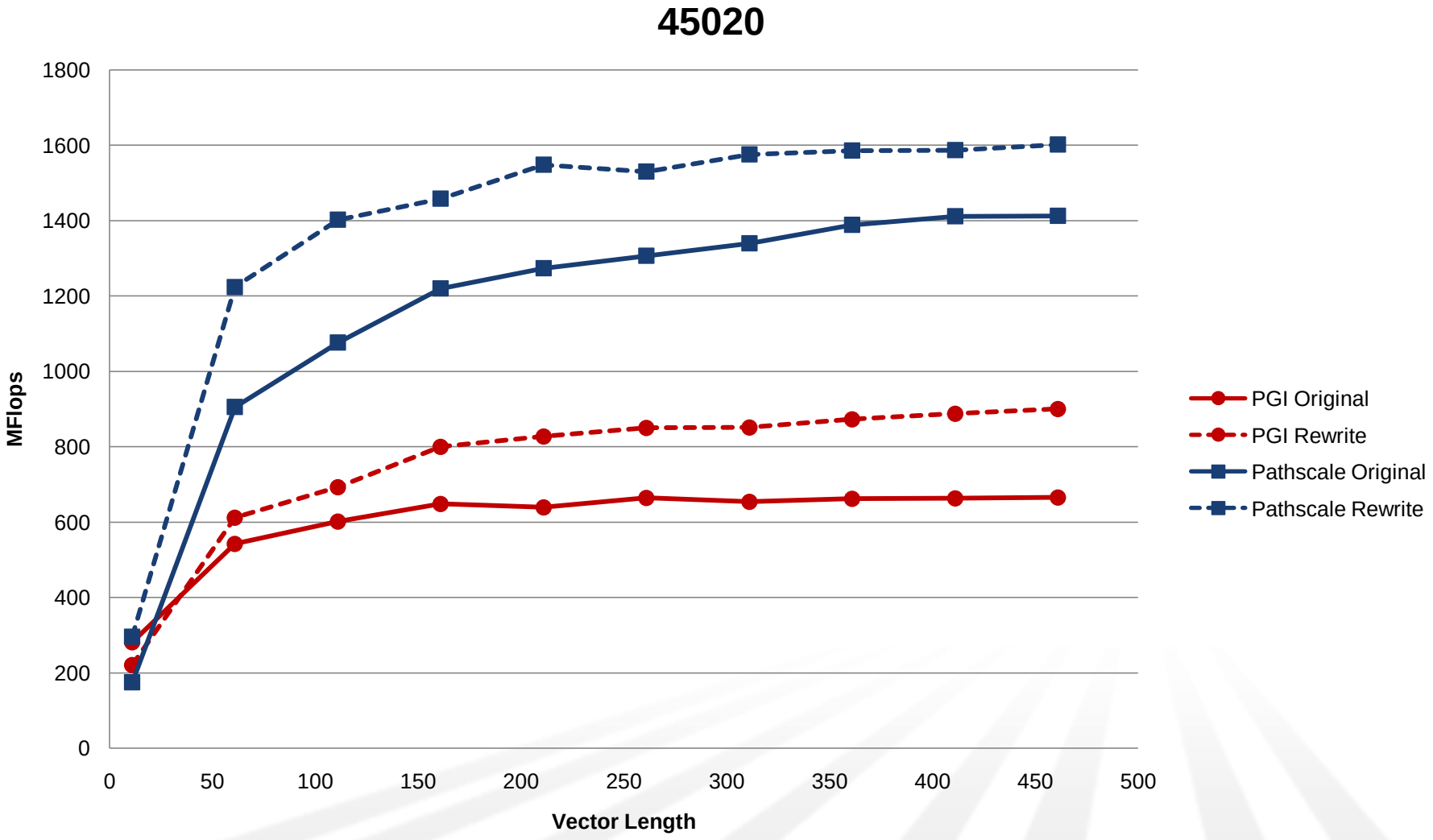
PGI

```
 73, Generated an alternate loop for the inner loop
     Generated vector sse code for inner loop
     Generated 1 prefetch instructions for this loop
     Generated vector sse code for inner loop
     Generated 1 prefetch instructions for this loop
 78, Generated 2 alternate loops for the inner loop
     Generated vector sse code for inner loop
     Generated 1 prefetch instructions for this loop
     Generated vector sse code for inner loop
     Generated 1 prefetch instructions for this loop
     Generated vector sse code for inner loop
     Generated 1 prefetch instructions for this loop
 82, Interchange produces reordered loop nest: 83, 84, 82
     Loop unrolled 5 times (completely unrolled)
 84, Generated vector sse code for inner loop
     Generated 1 prefetch instructions for this loop
```

Pathscale

```
(lp45020.f:73) LOOP WAS VECTORIZED.
(lp45020.f:78) LOOP WAS VECTORIZED.
(lp45020.f:78) LOOP WAS VECTORIZED.
(lp45020.f:84) Non-contiguous array "C(_BLNK__.0.0)" reference
exists. Loop was not vectorized.
(lp45020.f:84) Non-contiguous array "C(_BLNK__.0.0)" reference
exists. Loop was not vectorized.
```

Nested Loops: Results



LOOP UNROLLING

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-7, 8
(   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)      *                   + B(I,K+6) * C(K+6,J)
(   86)      *                   + B(I,K+7) * C(K+7,J)
(   87) 46033 CONTINUE
(   88) C
(   89)       DO 46034  KK = K, N
(   90)        DO 46034  J = 1, N
(   91)         DO 46034 I = 1, N
(   92)          A(I,J) = A(I,J) + B(I,KK) * C(KK ,J)
(   93) 46034 CONTINUE
(   94)
```

Rewrite

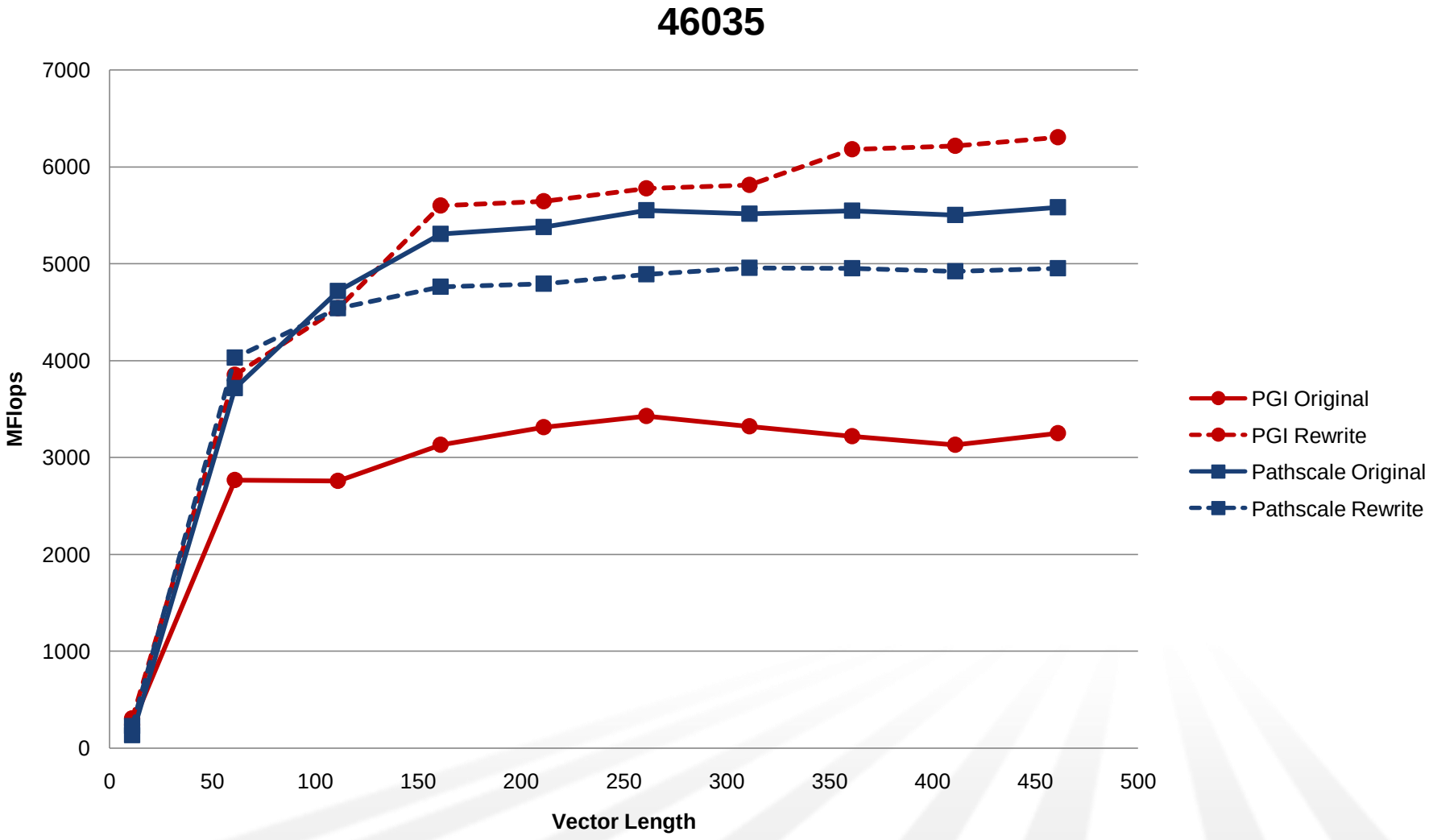
```
PGI
  71, Loop not vectorized/parallelized: contains call
  72, Memory zero idiom, loop replaced by call to __c_mzero4
  78, Generated 3 alternate loops for the inner loop
      Generated vector sse code for inner loop
      Generated 7 prefetch instructions for this loop
      Generated vector sse 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
  89, Loop interchange produces reordered loop nest: 90,89,91
  91, 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

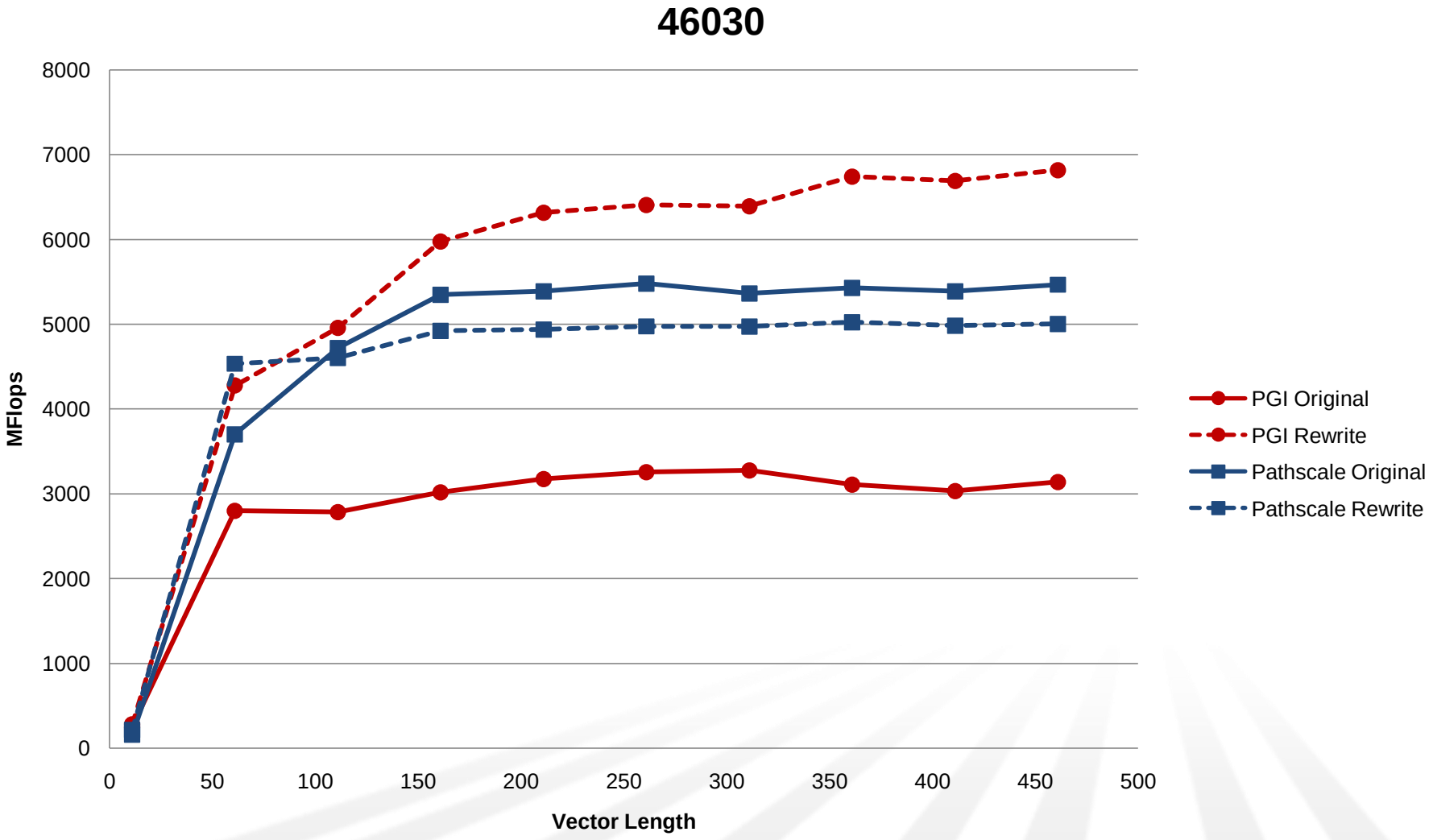
Pathscale

(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.

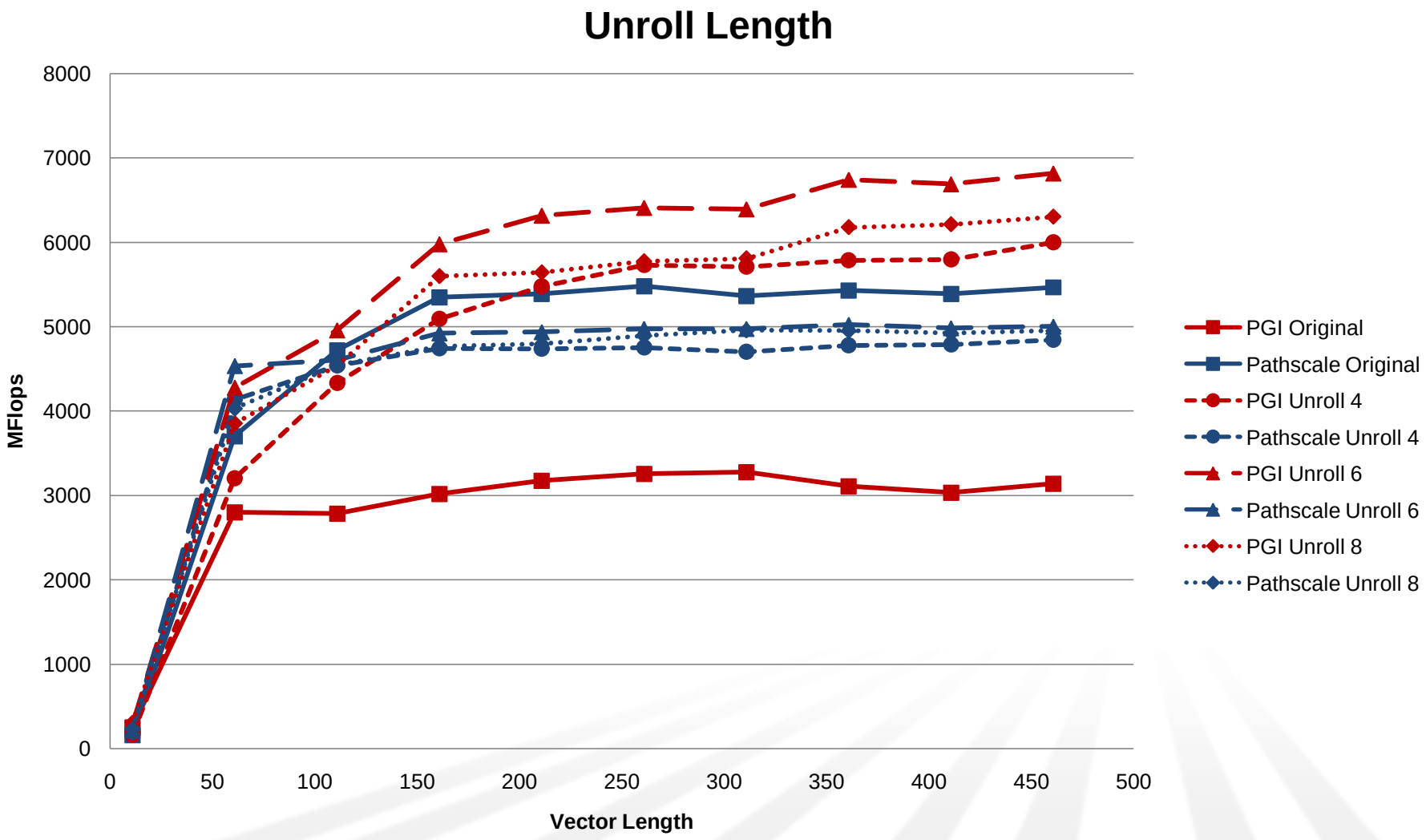
Loop Unrolling: Results (8)



Loop Unrolling: Results (6)



Loop Unrolling: Comparing 4, 6, and 8



Fully Unrolling: Original

```
(   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.

Fully Unrolling: 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)
```

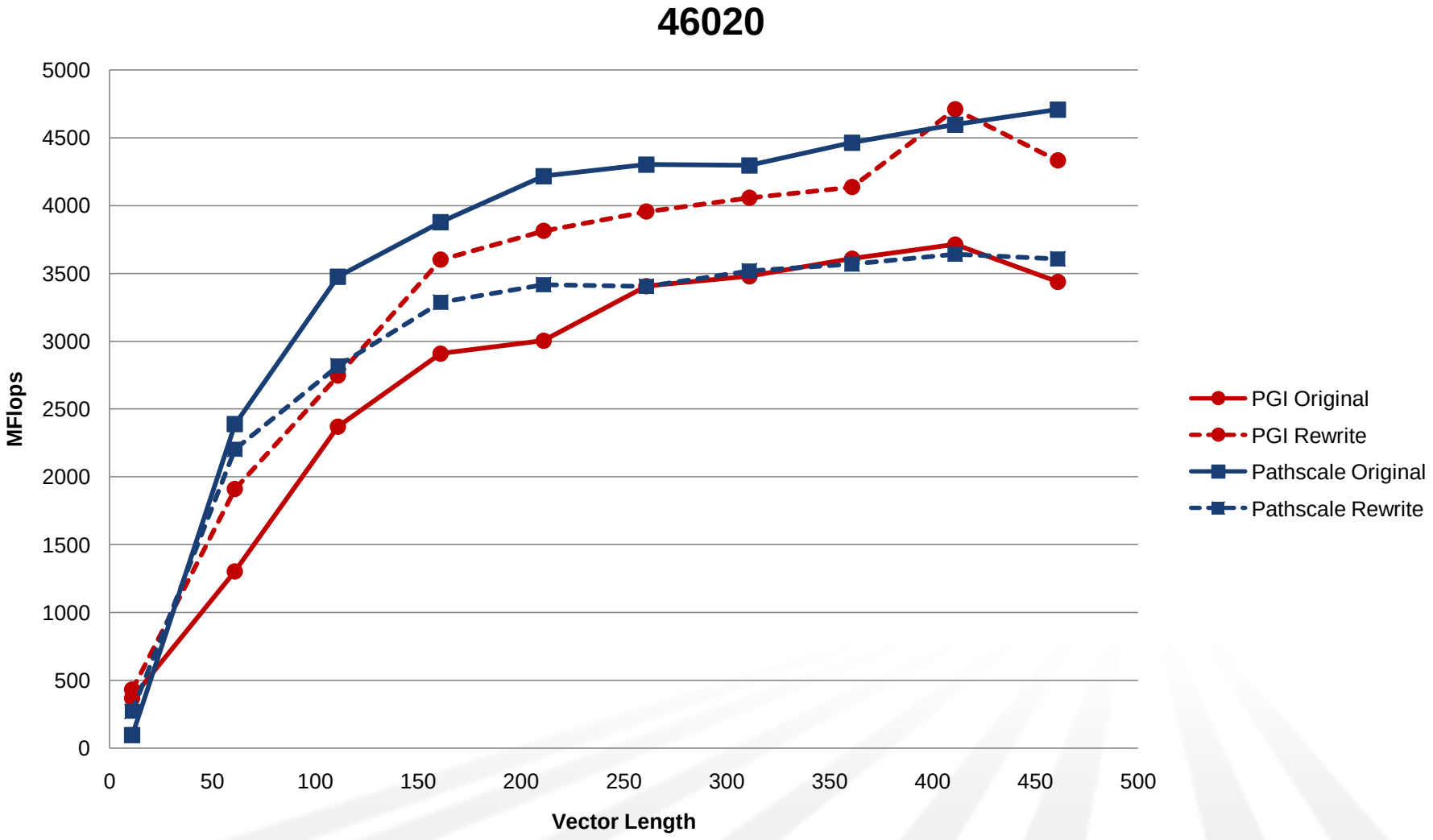
PGI

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.

Fully Unrolling: Results



VECTORIZATION LOGIC REORDERING

This is a scalar machine!

WHY VECTORIZE?

Why Vectorize? VVTVP

```
(  35) C     NON-RECURSIVE DO LOOP FOR TIMING COMPARISON
(  36)
(  37)       DO 43010 I = 2, N
(  38)        A(I) = A(I+1) * B(I) + C(I)
(  39) 43010 CONTINUE
(  40)
```

PGI

37, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 3 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 3 prefetch instructions for this loop

Pathscale

(lp43010.f:37) LOOP WAS VECTORIZED.

Why Vectorize? FOLR

```
(   52) C       RECURSIVE DO LOOP
(   53)
(   54)       DO 43011 I = 2, N
(   55)        A(I) = A(I-1) * B(I) + C(I)
(   56) 43011 CONTINUE
(   57)
```

PGI

54, Loop not vectorized: data dependency
Loop unrolled 2 times

Pathscale

(lp43010.f:54) Loop has dependencies. Loop was not vectorized.

Why Vectorize? FOLR - Unrolled

```
(   71) C       UNROLLED TO DEPTH FOUR
(   72)
(   73)       DO 43012 I = 2, N-3, 4
(   74)        A(I)   = A(I-1) * B(I)   + C(I)
(   75)        A(I+1) = A(I)   * B(I+1) + C(I+1)
(   76)        A(I+2) = A(I+1) * B(I+2) + C(I+2)
(   77)        A(I+3) = A(I+2) * B(I+3) + C(I+3)
(   78) 43012 CONTINUE
(   79)
(   80) C       CLEANUP LOOP FOR DEPTH FOUR UNROLLING
(   81)
(   82)       DO 43013 J = I,N
(   83)        A(J) = A(J-1) * B(J) + C(J)
(   84) 43013 CONTINUE
(   85)
```

PGI

73, Loop not vectorized: data dependency

82, Loop not vectorized: data dependency

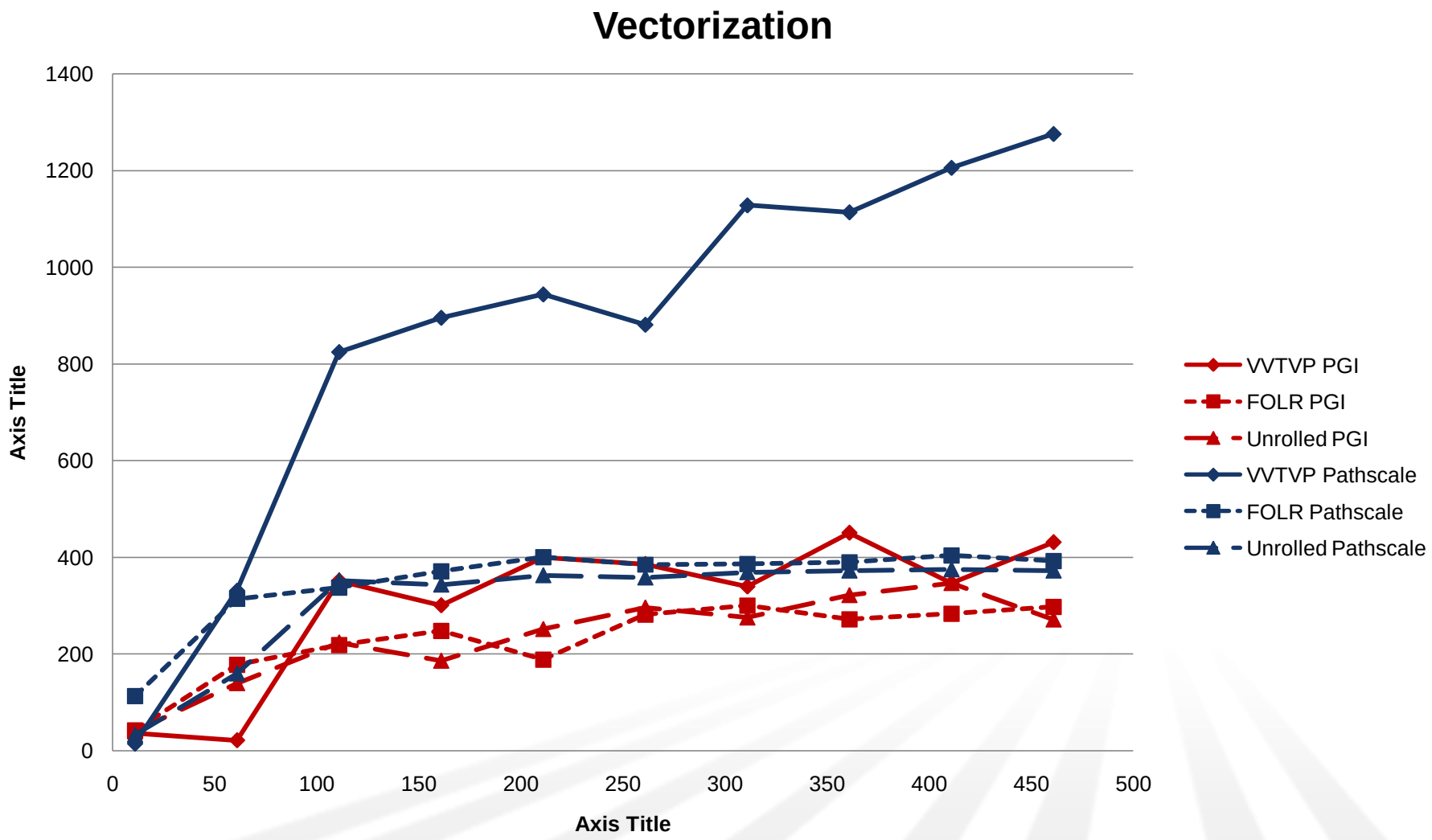
Loop unrolled 2 times

Pathscale

(lp43010.f:73) Non-contiguous array "C(_BLNK__.8000.0)" reference exists. Loop was not vectorized.

(lp43010.f:82) Loop has dependencies. Loop was not vectorized.

Why Vectorize?



Original Code :

```
do NBC=1_ST,MAXFRIC
   do NC=1_ST,NCELLS
    DX(1) = XC(1,NC) - MUDWAL(1,NBC)
    DX(2) = XC(2,NC) - MUDWAL(2,NBC)
    DX(3) = XC(3,NC) - MUDWAL(3,NBC)
    DOT = MUDWAL(4,NBC)*DX(1) + (MUDWAL(5,NBC)*DX(2) +
MUDWAL(6,NBC)*DX(3))
    if (DOT > 0.0_FP) then
     DST = DX(1)*DX(1) + DX(2)*DX(2) + DX(3)*DX(3)
     if (DST < DWALL(NC)) then
      DWALL(NC) = DST
      ICHNG(NC) = NBC
     end if
    end if
   end do
end do
```

Finds ‘smallest’ ‘positive’ distance : note the two ‘IF’ statements

The Loop count of MAXFRIC and NCELLS is in the 100,000’s

Totally memory bound code, XC and/or MUDWALL do not fit into cache.

MPI – 64 ranks : Timing is 261.2 seconds

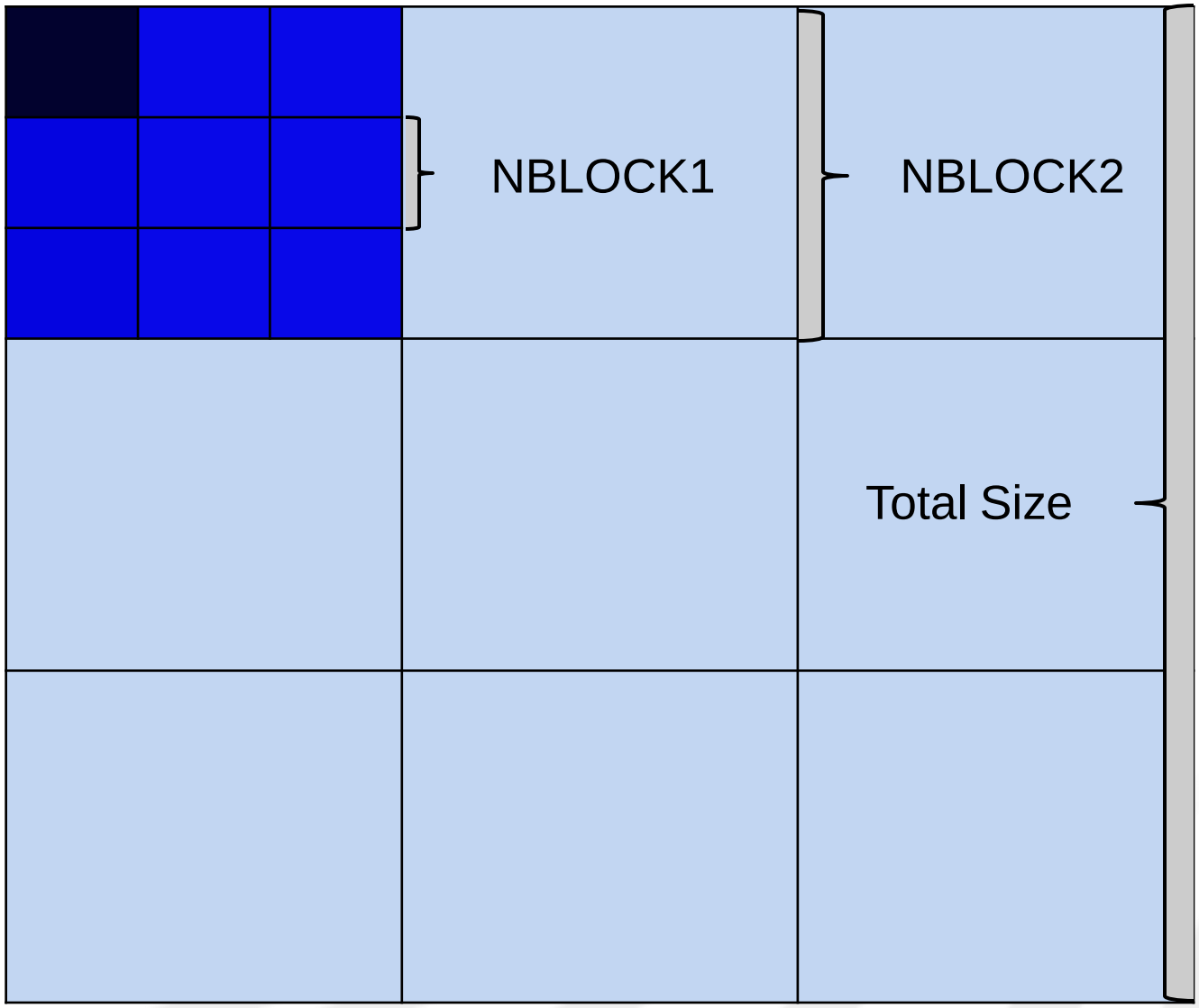
Original Code : Plan

```
 do NBC=1_ST,MAXFRIC : Double block
    do NC=1_ST,NCELLS : Double block
    DX(1) = XC(1,NC) - MUDWAL(1,NBC)
    DX(2) = XC(2,NC) - MUDWAL(2,NBC)
    DX(3) = XC(3,NC) - MUDWAL(3,NBC)
    DOT = MUDWAL(4,NBC)*DX(1) + (MUDWAL(5,NBC)*DX(2) +
MUDWAL(6,NBC)*DX(3))
    if (DOT > 0.0_FP) then
     DST = DX(1)*DX(1) + DX(2)*DX(2) + DX(3)*DX(3)
     if (DST < DWALL(NC)) then
      DWALL(NC) = DST
      ICHNG(NC) = NBC
     end if
    end if
   end do
  end do
```

Maybe we can ‘Block’ the MAXFRIC and NCELLS to keep things in cache.

Try to use both L1 and L2 cache blocking.

What double-blocking looks like



Total Memory Footprint

Fits in L2

Fits in L1

Double block Code :

```
do NBC_2=1_ST,MAXFRIC,BLOCK2
  do NC_2=1_ST,NCELLS,BLOCK2
   do NBC_1=NBC_2,MIN(NBC_2+BLOCK2-1,MAXFRIC),BLOCK1
    do NC_1=NC_2,MIN(NC_2+BLOCK2-1,NCELLS),BLOCK1
     do NBC=NBC_1,MIN(NBC_1+BLOCK1-1,MAXFRIC)
      do NC=NC_1,MIN(NC_1+BLOCK1-1,NCELLS)
       DX(1) = XC(1,NC) - MUDWAL(1,NBC)
       DX(2) = XC(2,NC) - MUDWAL(2,NBC)
       DX(3) = XC(3,NC) - MUDWAL(3,NBC)
       DOT=MUDWAL(4,NBC)*DX(1)+(MUDWAL(5,NBC)*DX(2) &
        + MUDWAL(6,NBC)*DX(3))
       if (DOT > 0.0_FP) then
        DST = DX(1)*DX(1) + DX(2)*DX(2) + DX(3)*DX(3)
        if (DST < DWALL(NC)) then
         DWALL(NC) = DST
         ICHNG(NC) = NBC
        end if
       end if
      end do
     end do
    end do
   end do
  end do
 end do
```

The L2 blocking is done via the BLOCK2
The L1 blocking is doen via the BLOCK1

Optimal sizes for BLOCK2/1 are found by figuring out what would fit, and then refined by testing.

BLOCK2 = 8*1024 : BLOCK1 = 1024

Progress so far :

- Original code 261.2 seconds
- Blocked code is 94.4 seconds…. ☺
- Speed up of 2.76X faster…..
- NOTES : NONE of this vectorized due to non unit stride….
- Would getting it to vectorize speed things up, or is the code still memory bandwidth bound to/from the L1 cache ??
- We would need to restructure the 'XC' arrays, which we can not do, but we could copy them…. I.E. strip mine / break the logic to vectorize part of the loop.

SSE packed Code :

Copy XC to XC_T
Break ‘NC’ loop into two
First NC loop vectorizes
Second loop does test

```
do NC=1_ST,NCELLS
   XC_T(NC,1) = XC(1,NC)
   XC_T(NC,2) = XC(2,NC)
   XC_T(NC,3) = XC(3,NC)
 enddo
 do NBC_2=1_ST,MAXFRIC,BLOCK2
  do NC_2=1_ST,NCELLS,BLOCK2
   do NBC_1=NBC_2,MIN(NBC_2+BLOCK2-1,MAXFRIC),BLOCK1
    do NC_1=NC_2,MIN(NC_2+BLOCK2-1,NCELLS),BLOCK1
     do NBC=NBC_1,MIN(NBC_1+BLOCK1-1,MAXFRIC)
      do NC=NC_1,MIN(NC_1+BLOCK1-1,NCELLS)   :Break loop
       DX_T(NC,1) = XC_T(NC,1) - MUDWAL(1,NBC)
       DX_T(NC,2) = XC_T(NC,2) - MUDWAL(2,NBC)
       DX_T(NC,3) = XC_T(NC,3) - MUDWAL(3,NBC)
       DOT_T(NC)=MUDWAL(4,NBC)*DX_T(NC,1)+ &
        (MUDWAL(5,NBC)*DX_T(NC,2)+MUDWAL(6,NBC)*DX_T(NC,3))
      enddo
      do NC=NC_1,MIN(NC_1+BLOCK1-1,NCELLS)
       if (DOT_T(NC) > 0.0_FP) then
        DST = DX_T(NC,1)*DX_T(NC,1) &
            + DX_T(NC,2)*DX_T(NC,2)+DX_T(NC,3)*DX_T(NC,3)
        if (DST < DWALL(NC)) then
         DWALL(NC) = DST
         ICHNG(NC) = NBC
        end if
       end if
      end do; end do; end do; end do; end do; end do
```

Progress or not :

- Original code 261.2 seconds
- Blocked code is 94.4 seconds…. ☺
- Speed up of 2.76X faster…..
- SSE packed code is 92.1 seconds… 2.83X faster…
- Not much faster; code still very memory (L1 bound)
- Time to give up……
- NEVER!!!
- Look at ‘IF’ logic; would switching the IF’s make it faster???

Logic switch:

Original logic

```
DOT_T(NC)=MUDWAL(4,NBC)*DX_T(NC,1)+
          (MUDWAL(5,NBC)*DX_T(NC,2)+MUDWAL(6,NBC)*DX_T(NC,3))
  if (DOT_T(NC) > 0.0_FP) then


 DST=DX_T(NC,1)*DX_T(NC,1)+DX_T(NC,2)*DX_T(NC,2)+DX_T(NC,3)*DX_T(NC,3)
 if (DST < DWALL(NC)) then
```

Or

Switched logic

```
DST_T(NC)  = DX_T(NC,1)*DX_T(NC,1) +
   DX_T(NC,2)*DX_T(NC,2)+DX_T(NC,3)*DX_T(NC,3)
  if (DST_T(NC) < DWALL(NC)) then

  if((MUDWAL(4,NBC)*DX_T(NC,1)+ &
           (MUDWAL(5,NBC)*DX_T(NC,2)+MUDWAL(6,NBC)*DX_T(NC,3))) >
   0.0_FP) then
```

The DST cost is 3 loads, 3*, 2+

The DOT cost is 6 loads, 3*, 2+

The DST is 50/50 branching, the DOT goes to zero if we get the best DOT early on…

It just might be faster…..

Switched logic Code :

Switch logic test.
Put ‘best’ test on outside.
Put largest work on inside

```
do NC=1_ST,NCELLS
    XC_T(NC,1)  = XC(1,NC)
    XC_T(NC,2)  = XC(2,NC)
    XC_T(NC,3)  = XC(3,NC)
  enddo
  do NBC_2=1_ST,MAXFRIC,BLOCK2
   do NC_2=1_ST,NCELLS,BLOCK2
    do NBC_1=NBC_2,MIN(NBC_2+BLOCK2-1,MAXFRIC),BLOCK1
     do NC_1=NC_2,MIN(NC_2+BLOCK2-1,NCELLS),BLOCK1
      do NBC=NBC_1,MIN(NBC_1+BLOCK1-1,MAXFRIC)
       do NC=NC_1,MIN(NC_1+BLOCK1-1,NCELLS)
        DX_T(NC,1)  = XC_T(NC,1)  - MUDWAL(1,NBC)
        DX_T(NC,2)  = XC_T(NC,2)  - MUDWAL(2,NBC)
        DX_T(NC,3)  = XC_T(NC,3)  - MUDWAL(3,NBC)
        DST_T(NC)   = DX_T(NC,1)*DX_T(NC,1)  &
             + DX_T(NC,2)*DX_T(NC,2)+DX_T(NC,3)*DX_T(NC,3)
       enddo
       do NC=NC_1,MIN(NC_1+BLOCK1-1,NCELLS)
        if (DST_T(NC) < DWALL(NC)) then
         if((MUDWAL(4,NBC)*DX_T(NC,1)+ &
          (MUDWAL(5,NBC)*DX_T(NC,2)+MUDWAL(6,NBC)*DX_T(NC,3))) > 0.0_FP) then
          DWALL(NC)  = DST_T(NC)
          ICHNG(NC)  = NBC
         end if
        end if
       end do; end do; end do; end do; end do; end do
```

Progress or not number 2 :

- Original code 261.2 seconds
- Blocked code is 94.4 seconds… 2.76X faster…..
- SSE packed code is 92.1 seconds… 2.83X faster…
- Switched logic code is 83.0 seconds ☺
- Speed up of 3.15X faster
- Are we done yet …
- NEVER!!!
- Did the reversing of logic change the BLOCKING factors…. ?
- Go back and test BLOCK2 and BLOCK1…

Progress or not number 3 :

- Increasing BLOCK2 larger then 8*1024 slows things down FAST….
- Decreasing BLOCK2 smaller then 8*1024 slows things down slowly….
- Expected behavior, BLOCK2 is L2 factor, our work was done on L1..
- Making BLOCK1 larger then 1024 slows things down FAST….
- Making BLOCK1 512 … ☺ 74.8 seconds
- Making BLOCK1 256 … ☺ 71.7 seconds
- Making BLOCK1 smaller (128) slow things down (80.3)

Final result (or is it…) :

- Original code 261.2 seconds
- Blocked code is 94.4 seconds…. 2.76X faster…..
- SSE packed code is 92.1 seconds… 2.83X faster…
- Switched logic code is 83.0 seconds… 3.15X faster

- Re-block L1 code is 71.7 seconds ☺

- Code is now 3.64X FASTER….

Original

```
do NBC=1_ST,MAXFRIC
   do NC=1_ST,NCELLS
    DX(1) = XC(1,NC) -
 MUDWAL(1,NBC)
    DX(2) = XC(2,NC) -
 MUDWAL(2,NBC)
    DX(3) = XC(3,NC) -
 MUDWAL(3,NBC)
    DOT = MUDWAL(4,NBC)*DX(1) + &
 (MUDWAL(5,NBC)*DX(2) + &
 MUDWAL(6,NBC)*DX(3))
    if (DOT > 0.0_FP) then
     DST = DX(1)*DX(1) +
 DX(2)*DX(2) + DX(3)*DX(3)
     if (DST < DWALL(NC)) then
      DWALL(NC) = DST
      ICHNG(NC) = NBC
     end if
    end if
   end do
  end do
```

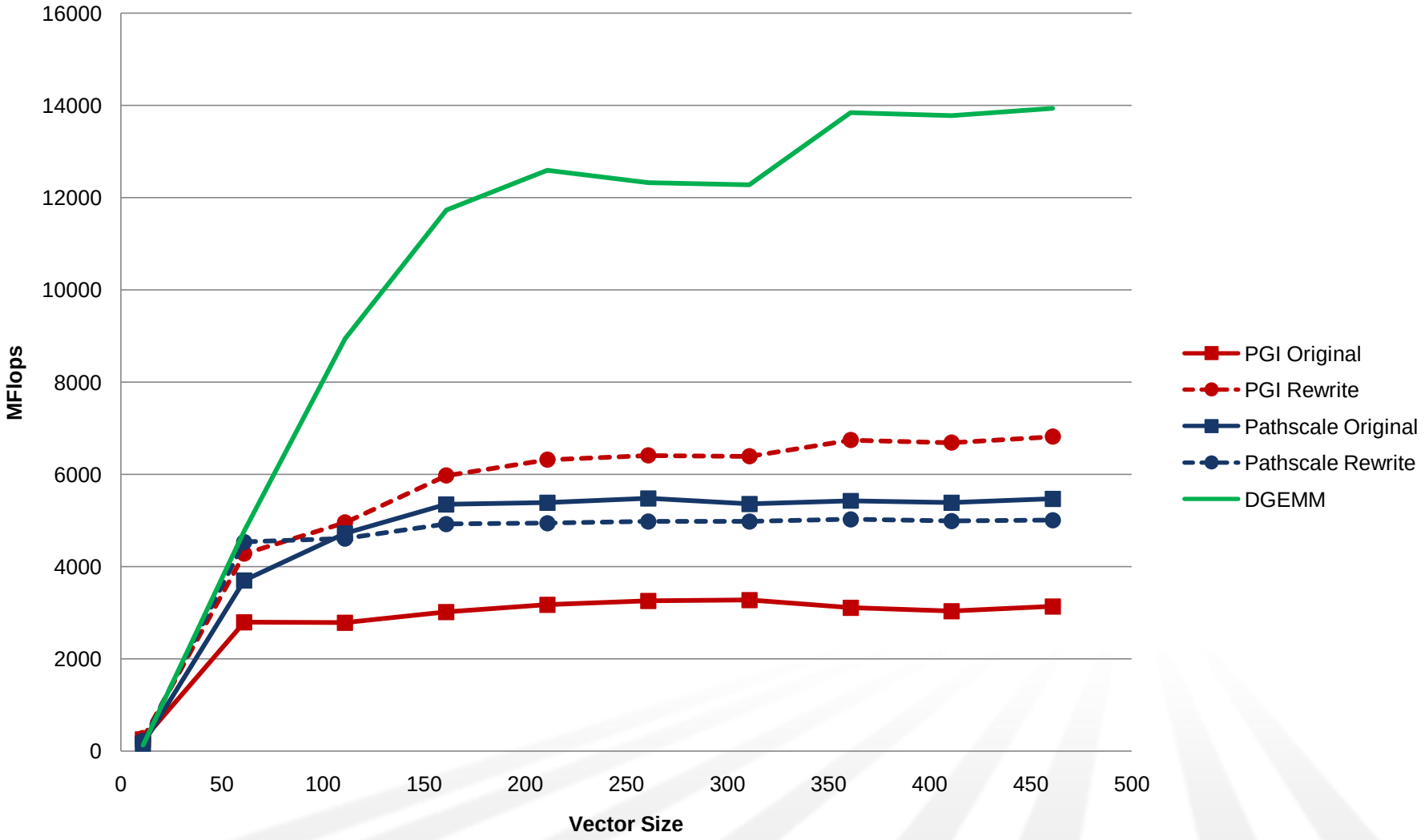
Rewritten

```
do NC=1_ST,NCELLS
     XC_T(NC,1) = XC(1,NC)
     XC_T(NC,2) = XC(2,NC)
     XC_T(NC,3) = XC(3,NC)
   enddo
   do NBC_2=1_ST,MAXFRIC,BLOCK2
    do NC_2=1_ST,NCELLS,BLOCK2
     do NBC_1=NBC_2,MIN(NBC_2+BLOCK2-1,MAXFRIC),BLOCK1
      do NC_1=NC_2,MIN(NC_2+BLOCK2-1,NCELLS),BLOCK1
       do NBC=NBC_1,MIN(NBC_1+BLOCK1-1,MAXFRIC)
        do NC=NC_1,MIN(NC_1+BLOCK1-1,NCELLS)
         DX_T(NC,1) = XC_T(NC,1) - MUDWAL(1,NBC)
         DX_T(NC,2) = XC_T(NC,2) - MUDWAL(2,NBC)
         DX_T(NC,3) = XC_T(NC,3) - MUDWAL(3,NBC)
         DST_T(NC)  = DX_T(NC,1)*DX_T(NC,1) &
              +
    DX_T(NC,2)*DX_T(NC,2)+DX_T(NC,3)*DX_T(NC,3)
        enddo
        do NC=NC_1,MIN(NC_1+BLOCK1-1,NCELLS)
         if (DST_T(NC) < DWALL(NC)) then
          if((MUDWAL(4,NBC)*DX_T(NC,1)+ &

    (MUDWAL(5,NBC)*DX_T(NC,2)+MUDWAL(6,NBC)*DX_T(NC,3)))
    > 0.0_FP) then
           DWALL(NC) = DST_T(NC)
           ICHNG(NC) = NBC
          end if
         end if
        end do;
       end do;
      end do;
     end do;
    end do;
   end do
```

USE LIBRARIES

Use Libraries!



The Best Optimization Technique:

SEEK HELP