CRAY
THE SUPERCOMPUTER COMPANY

A methodical approach for scaling applications to 100,000 cores

John Levesque
CTO Office
Applications
Supercomputing Center of Excellence

The steps – 1) Formulate the problem

- Formulate the problem
  - It should be a production style problem
    - Weak scaling
      - Finer grid as processors increase
      - Fixed amount of work when processors increase
    - Strong scaling
      - Fixed problem size as processors increase
        - Less and less work for each processor as processors increase
  - It should be small enough to measure on a current system; however, able to scale to larger processor counts
  - The problem identified should make good science sense
    - Climate models cannot always reduce grid size if the initial conditions don't warrant it

Think Bigger

The steps – 2) Instrument the application

- Instrument the application
  - Run the production case
    - Run long enough that the initialization does not use > 1% of the time
    - Run with normal I/O
  - Use Craypat's APA
    - First gather sampling for line number profile
    - Second gather instrumentation (-g mpi,io)
      - Hardware counters
      - MPI message passing information
      - I/O information

load module
make
pat_build -O apa a.out
Execute
pat_report *.xf
pat_build –O *.apa
Execute

Using Craypat on large numbers of processors

- Pat_report can use an inordinate amount of time on the front-end system
  - Try submitting the pat_report as a batch job
  - Only give Pat_report a subset of the .xf files
    - Pat_report fms_cs_test13.x+apa+25430-12755tdt/*3.xf

Using Craypat MPI statistics

```
MPI Msg Bytes |  MPI Msg |   MsgSz | 16B<= |  256B<= |   4KB<= |Experiment=1
              |    Count |    <16B | MsgSz |   MsgSz |   MsgSz |Function
              |          |   Count | <256B |    <4KB |   <64KB | Caller
              |          |         | Count |   Count |   Count |  PE[mmm]
 3062457144.0 | 144952.0 | 15022.0 |  39.0 | 64522.0 | 65369.0 |Total
|-------------------------------------------------------------------------
| 3059984152.0 | 129926.0 |      -- |  36.0 | 64522.0 | 65368.0 |mpi_isend_
||------------------------------------------------------------------------
|| 1727628971.0 |  63645.1 |      -- |   4.0 | 31817.1 | 31824.0 |MPP_DO_UPDATE_R8_3DV.in.MPP_DOMAINS_MOD
3|              |          |         |       |         |         | MPP_UPDATE_DOMAIN2D_R8_3DV.in.MPP_DOMAINS_MOD
||||----------------------------------------------------------------------
4||| 1680716892.0 |  61909.4 |      -- |    -- | 30949.4 | 30960.0 |DYN_CORE.in.DYN_CORE_MOD
5|||              |          |         |       |         |         | FV_DYNAMICS.in.FV_DYNAMICS_MOD
6|||              |          |         |       |         |         |  ATMOSPHERE.in.ATMOSPHERE_MOD
7|||              |          |         |       |         |         |   MAIN__
8|||              |          |         |       |         |         |    main
|||||||||-----------------------------------------------------------------
9|||||||| 1680756480.0 |  61920.0 |      -- |    -- | 30960.0 | 30960.0 |pe.13666
9|||||||| 1680756480.0 |  61920.0 |      -- |    -- | 30960.0 | 30960.0 |pe.8949
9|||||||| 1651777920.0 |  54180.0 |      -- |    -- | 23220.0 | 30960.0 |pe.12549
|||||||||=================================================================
```

Memory allocation data from Craypat

```
Table 7:  Heap Leaks during Main Program

 Tracked | Tracked | Tracked |Experiment=1
  MBytes |  MBytes | Objects |Caller
     Not |     Not |     Not | PE[mmm]
 Freed % |   Freed |   Freed |

  100.0% | 593.479 |   43673 |Total
|-----------------------------------------
|   97.7% | 579.580 |   43493 |_F90_ALLOCATE
||----------------------------------------
||   61.4% | 364.394 |     106 |SET_DOMAIN2D.in.MPP_DOMAINS_MOD
3|         |         |         | MPP_DEFINE_DOMAINS2D.in.MPP_DOMAINS_MOD
4|         |         |         |  MPP_DEFINE_MOSAIC.in.MPP_DOMAINS_MOD
5|         |         |         |   DOMAIN_DECOMP.in.FV_MP_MOD
6|         |         |         |    RUN_SETUP.in.FV_CONTROL_MOD
7|         |         |         |     FV_INIT.in.FV_CONTROL_MOD
8|         |         |         |      ATMOSPHERE_INIT.in.ATMOSPHERE_MOD
9|         |         |         |       ATMOS_MODEL_INIT.in.ATMOS_MODEL
10         |         |         |        MAIN__
11         |         |         |         main
||||||||||-----------------------------
12||||||||||    0.0% | 364.395 |     110 |pe.43
12||||||||||    0.0% | 364.394 |     107 |pe.8181
12||||||||||    0.0% | 364.391 |      88 |pe.1047
```

The steps – 3) Examine Results

- Examine Results
  - Is there load imbalance?
    - Yes – fix it first – go to step 4
    - No – you are lucky
  - Is computation > 50% of the runtime
    - Yes – go to step 5
  - Is communication > 50% of the runtime
    - Yes – go to step 6
  - Is I/O > 50% of the runtime
    - Yes – go to step 7

Always fix load imbalance first

Craypat load-imbalance data

Table 1: Profile by Function Group and Function

Time %	Time	Imb. Time	Imb. Time %	Calls	Experiment=1 Group Function PE='HIDE'
100.0%	1061.141647	--	--	3454195.8	Total
70.7%	750.564025	--	--	280169.0	MPI_SYNC
45.3%	480.828018	163.575446	25.4%	14653.0	mpi_barrier_(sync)
18.4%	195.548030	33.071062	14.5%	257546.0	mpi_allreduce_(sync)
7.0%	74.187977	5.261545	6.6%	7970.0	mpi_bcast_(sync)
15.2%	161.166842	--	--	3174022.8	MPI
10.1%	106.808182	8.237162	7.2%	257546.0	mpi_allreduce_
3.2%	33.841961	342.085777	91.0%	755495.8	mpi_waitall_
14.1%	149.410781	--	--	4.0	USER
14.0%	148.048597	446.124165	75.1%	1.0	main

The steps – 4) Application is load imbalanced

- What is causing the load imbalance
  - Computation
    - Is decomposition appropriate?
    - Would RANK_REORDER help?
  - Communication
    - Is decomposition appropriate?
    - Would RANK_REORDER help?
    - Are recevies pre-posted
- OpenMP may help
  - Able to spread workload with less overhead
    - Large amount of work to go from all-MPI to Hybrid
      - Must accept challenge to OpenMP-ize large amount of code
- Go back to step 2
    - Re-gather statistics

Need Craypat reports

Is SYNC time due to computation?

The steps – 5) Computation is Major Bottleneck

- What is causing the Bottleneck?
  - Computation
    - Is application Vectorized
      - No – vectorize it
    - What library routines are being used?
  - Memory Bandwidth
    - What is cache utilization?
      - Bad – go to step 7
    - TLB problems?
      - Bad – go to step 8
- OpenMP may help
  - Able to spread workload with less overhead
    - Large amount of work to go from all-MPI to Hybrid
      - Must accept challenge to OpenMP-ize large amount of code
- Go back to step 2
    - Re-gather statistics

Need Hardware counters
&
Compiler listing in hand

Hardware Counters

```
USER / MPP_DO_UPDATE_R8_3DV.in.MPP_DOMAINS_MOD
------------------------------------------------------------------------
  Time%                                              10.2%
  Time                                           49.386043 secs
  Imb.Time                                        1.359548 secs
  Imb.Time%                                           2.7%
  Calls                        167.1 /sec          8176.0 calls
  PAPI_L1_DCM                10.512M/sec        514376509 misses
  PAPI_TLB_DM                 2.104M/sec        102970863 misses
  PAPI_L1_DCA               155.710M/sec       7619492785 refs
  PAPI_FP_OPS                                          0 ops
  User time (approx)          48.934 secs  112547914072 cycles  99.1%Time
  Average Time per Call                           0.006040 sec
  CrayPat Overhead : Time      0.0%
  HW FP Ops / User time                                0 ops  0.0%peak(DP)
  HW FP Ops / WCT
  Computational intensity      0.00 ops/cycle         0.00 ops/ref
  MFLOPS (aggregate)           0.00M/sec
  TLB utilization             74.00 refs/miss        0.145 avg uses
  D1 cache hit,miss ratios    93.2% hits              6.8% misses
  D1 cache utilization (M)    14.81 refs/miss        1.852 avg uses
```

Table 2: Profile by Group, Function, and Line

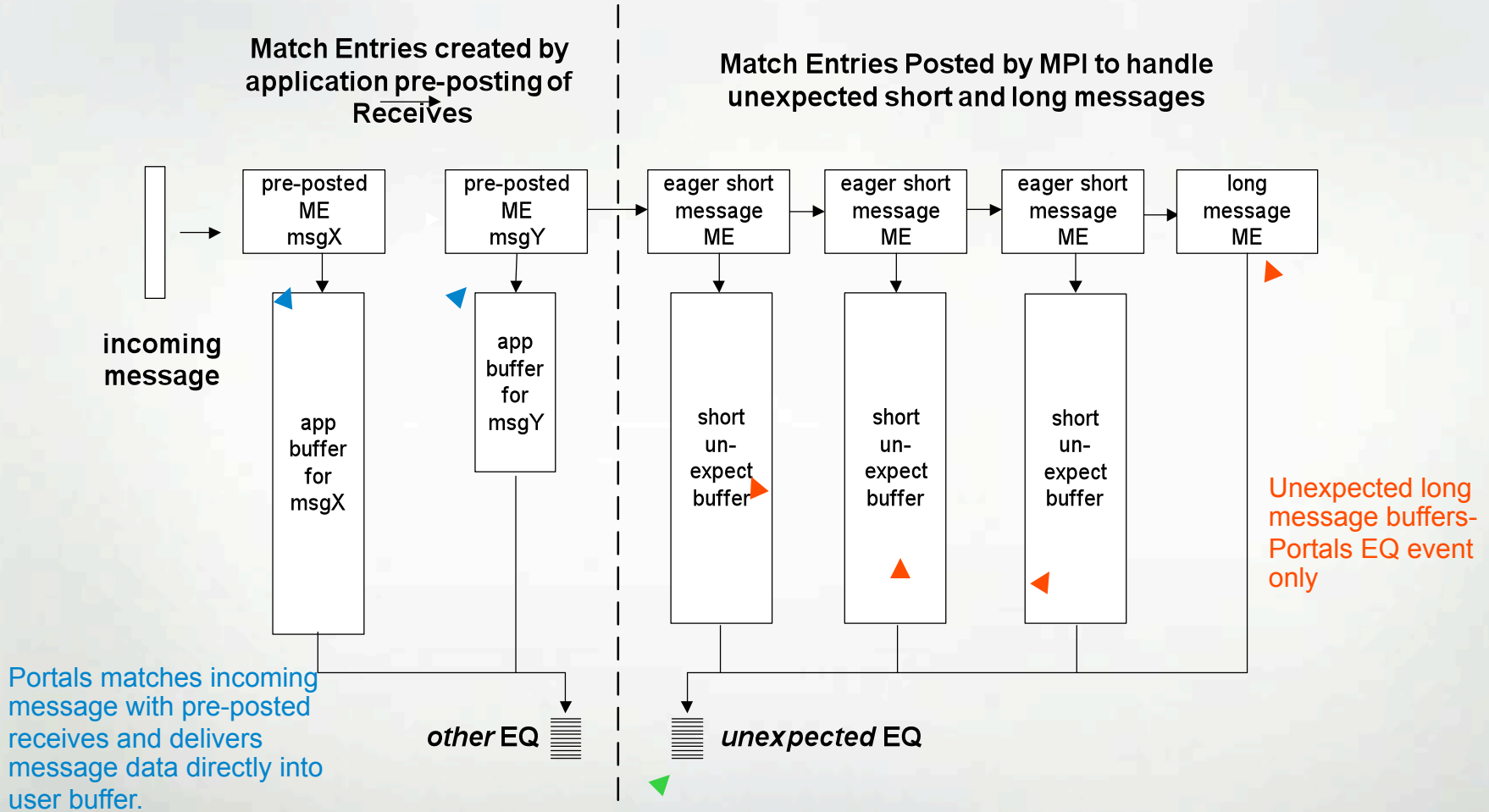
```
 Samp % |   Samp |Imb. Samp |   Imb. |Experiment=1
        |        |          | Samp % |Group
        |        |          |        | Function
        |        |          |        |  Source
        |        |          |        |   Line
        |        |          |        |    PE='HIDE'

 100.0% | 103828 |       -- |     -- |Total
|-----------------------------------------------
|  48.9% |  50784 |       -- |     -- |USER
||----------------------------------------------
||  11.0% |  11468 |       -- |     -- |MPP_DO_UPDATE_R8_3DV.in.MPP_DOMAINS_MOD
3|       |        |          |        | shared/mpp/include/mpp_do_updateV.h
||||------------------------------------------
4|||   2.9% |   3056 |   238.53 |   7.2% |line.380
4|||   2.8% |   2875 |   231.97 |   7.5% |line.967
4|||   2.0% |   2071 |   310.19 |  13.0% |line.1028
||||==========================================
```

The steps – 6) Communication is Major Bottleneck

- What is causing the Bottleneck?
  - Collectives
    - MPI_ALLTOALL
    - MPI_ALLREDUCE
    - MPI_REDUCE
    - MPI_VGATHER/MPI_VSCATTER
  - Point to Point
    - Are receives pre-posted
      - Don't use MPI_SENDRECV
    - What are the message sizes
      - Small – Combine
      - Large – divide and overlap
- OpenMP may help
  - Able to spread workload with less overhead
    - Large amount of work to go from all-MPI to Hybrid
      - Must accept challenge to OpenMP-ize large amount of code
- Go back to step 2
    - Re-gather statistics

Look at craypat report
MPI message sizes

XT MPI – Receive Side



Portals matches incoming message with pre-posted receives and delivers message data directly into user buffer.

An unexpected message generates two entries on unexpected EQ

Unexpected short message buffers

Unexpected long message buffers- Portals EQ event only

Not Pre-posted Receives

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

18

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, _
```

The steps – 7) I/O is Major Bottleneck

- What type of I/O?
  - One writer – large files
    - Stripe across most OSTs
  - All writers – small files
    - Stripe across one OST
  - MPI-I/O?
    - Try using subset of writers
  - Go back to step 2
    - Re-gather statistics

Look at craypat report on file statistics
Look at read/write sizes

Vectorization

- Stride one memory accesses
- No IF tests
- No subroutine calls
  - Inline
- What is size of loop
- Loop nest
  - Stride one on inside
  - Longest on the inside
- Unroll small loops
- Increase computational intensity
  - CU = (vector flops/number of memory accesses)

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

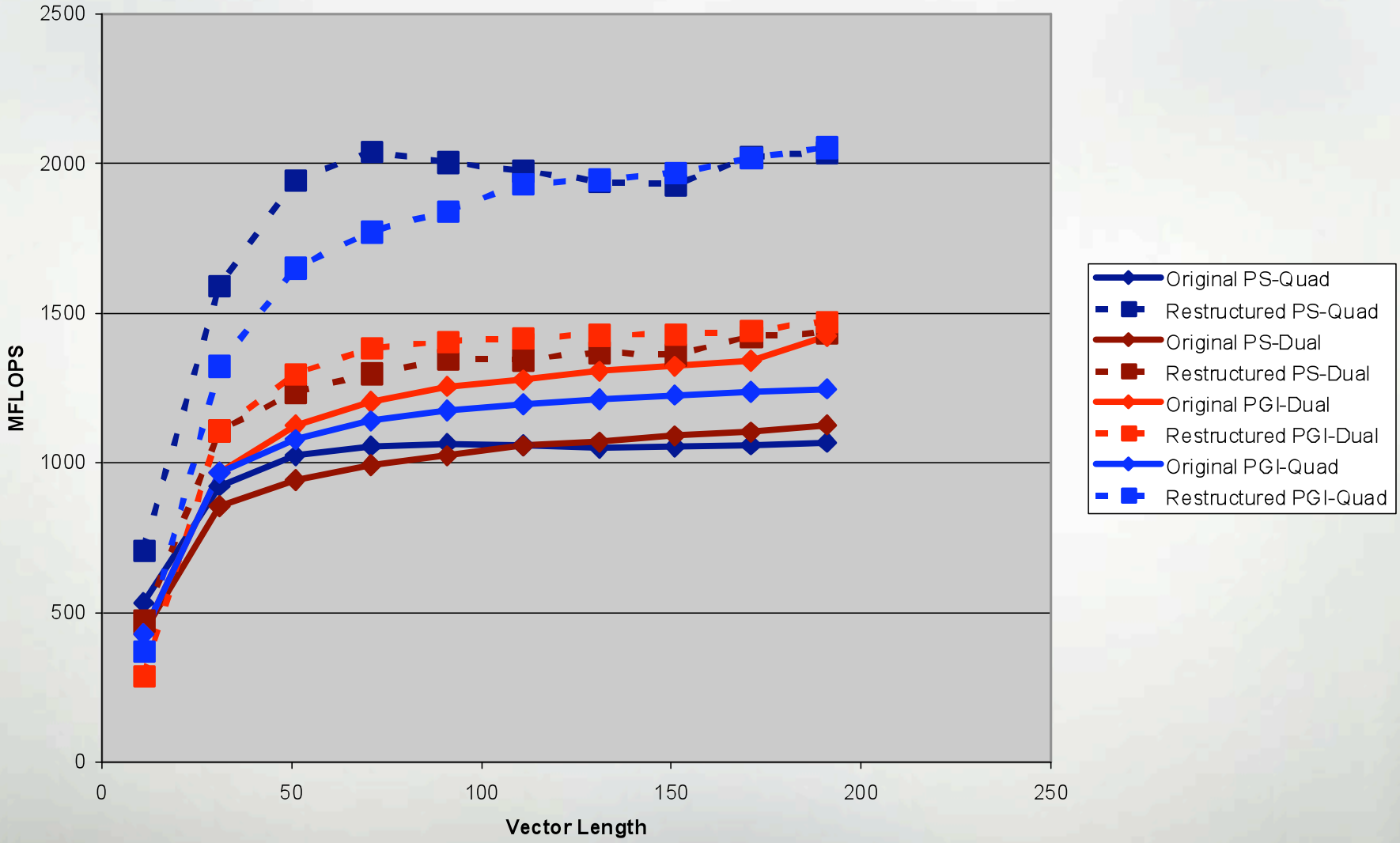
```
(  209)         I = 1
(  210)         I1         = 2
(  211)         I2         = 3
(  212)         VAI(I) = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
(  213)         VBI(I) = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
(  214)         VCI(I) = (-3. * C(I,J,K) + 4. * C(I1,J,K) - 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

Original

```
(   42) C      THE ORIGINAL
(   43)
(   44)       DO 48070 I = 1, N
(   45)        A(I) =  (B(I)**2 + C(I)**2)
(   46)        CT   = PI * A(I) + (A(I))**2
(   47)        CALL SSUB (A(I), CT, D(I), E(I))
(   48)        F(I) = (ABS (E(I)))
(   49) 48070 CONTINUE
(   50)
```

PGI

44, Loop not vectorized: contains call

Pathscale

Nothing

Restructured

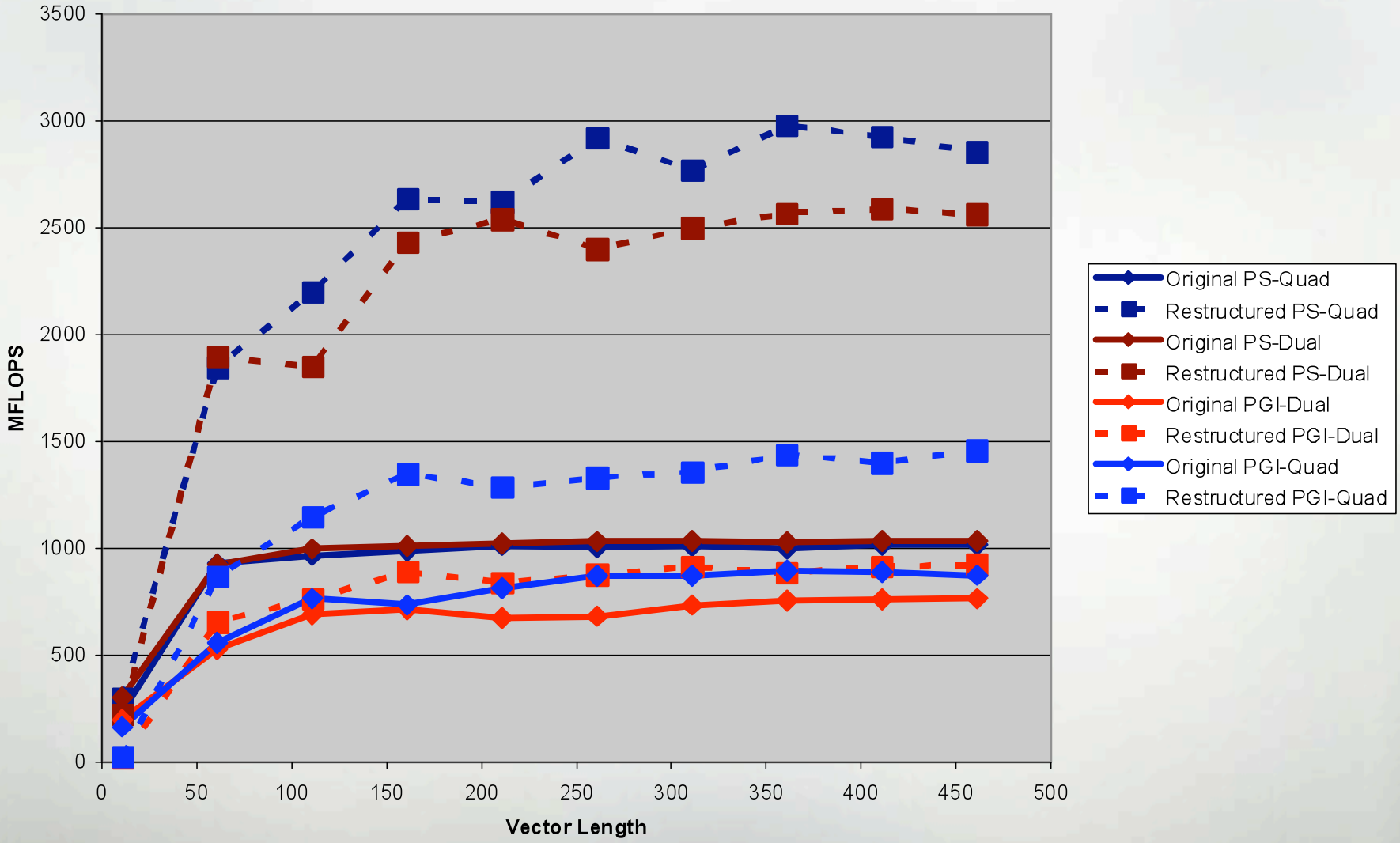
```
(   69) C      THE RESTRUCTURED
(   70)
(   71)       DO 48071 I = 1, N
(   72)        A(I) = (B(I)**2 + C(I)**2)
(   73)        CT   = PI * A(I) + (A(I))**2
(   74)        E(I) = A(I)**2 + (ABS (A(I) + CT)) * (CT * ABS (A(I) - CT))
(   75)        D(I) = A(I) + CT
(   76)        F(I) = (ABS (E(I)))
(   77) 48071 CONTINUE
(   78)
```

PGI

71, Generated an alternate loop for the inner loop
    Unrolled inner loop 4 times
    Used combined stores for 2 stores
    Generated 2 prefetch instructions for this loop
    Unrolled inner loop 4 times
    Used combined stores for 2 stores
    Generated 2 prefetch instructions for this loop

Pathscale

(lp48070.f:71) LOOP WAS VECTORIZED.


LP48070
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
3000
3500
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

Cache Utilization

- Fortran 90 syntax and/or lots of DO loops
  - Stripe mine outside of block of loops
- Multi-nested loops
  - Look at blocking example

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
========================================================================
```

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
```

MHD3D Original

```
      DO 200 K=0,KX
      DO 200 J=0,JX
      DO 200 I=0,IX
         F(I,J,K)=RVX(I,J,K)
         G(I,J,K)=RVY(I,J,K)
         H(I,J,K)=RVZ(I,J,K)
         S(I,J,K)=0.
200   CONTINUE
      CALL HALF(RO,ROH,DRO,F,G,H,S)
```

Original HALF

```
C=======================================================================
      DO 100 K=1,KXS1
      DO 100 J=1,JXS1
      DO 100 I=1,IXS1
         DU(I,J,K)=DU(I,J,K)-0.5*DT*
     &            (0.5*RDXM(I)*(F(I+1,J,K)-F(I-1,J,K))
     &            +0.5*RDYM(J)*(G(I,J+1,K)-G(I,J-1,K))
     &            +0.5*RDZM(K)*(H(I,J,K+1)-H(I,J,K-1))
     &            +S(I,J,K))
100   CONTINUE
C=======================================================================
C***  proceed half step using flux across cell boundary               ***
C=======================================================================
```

Original HALF

```
      DO 200 K=0,KXS1
      DO 200 J=0,JXS1
      DO 200 I=0,IXS1
C----------- cell average --------------------
        UH =0.125*(U(I+1,J+1,K+1)+U(I,J+1,K+1)
     &             +U(I+1,J+1,K)  +U(I,J+1,K)
     &             +U(I+1,J,K+1)  +U(I,J,K+1)
     &             +U(I+1,J,K)    +U(I,J,K))
        SH =0.125*(S(I+1,J+1,K+1)+S(I,J+1,K+1)
     &             +S(I+1,J+1,K)  +S(I,J+1,K)
     &             +S(I+1,J,K+1)  +S(I,J,K+1)
     &             +S(I+1,J,K)    +S(I,J,K))
C----------- flux across cell boundary ----------------------
        DFDX = 0.25*RDX(I)*(F(I+1,J+1,K+1)-F(I,  J+1,K+1)
     &                     +F(I+1,J+1,K)  -F(I,  J+1,K)
     &                     +F(I+1,J,  K+1)-F(I,  J,  K+1)
     &                     +F(I+1,J,  K)  -F(I,  J,  K))
        DGDY = 0.25*RDY(J)*(G(I+1,J+1,K+1)-G(I+1,J,  K+1)
     &                     +G(I+1,J+1,K)  -G(I+1,J,  K)
     &                     +G(I,  J+1,K+1)-G(I,  J,  K+1)
     &                     +G(I,  J+1,K)  -G(I,  J,  K))
        DHDZ = 0.25*RDZ(K)*(
     &                      H(I+1,J+1,K+1)-H(I+1,J+1,K)
     &                     +H(I+1,J,  K+1)-H(I+1,J,  K)
     &                     +H(I,  J+1,K+1)-H(I,  J+1,K)
     &                     +H(I,  J,  K+1)-H(I,  J,  K))
C------------ summation of all terms -----------------------
        UN(I,J,K)  = UH-DT*(DFDX+DGDY+DHDZ+SH)
200   CONTINUE
      RETURN
      END
```

Storage Analysis

Original

	Variables	NX	NY	NZ	Mwords	MB	L2	TLBs
Loop 200		7	259	255	9	20.8	37	75	38
Half Do 100		5	259	255	9	2.972025	11.8881	23.7762	11
Half Do 200		6	259	255	9	3.56643	14.26572	28.53144	14

MHD3D Restructured

```
DO K = 0,KX
  KDOWN=K+1
  KUP=K+1
  IF(K.EQ.0)THEN
  KDOWN=k
  KUP=k+1
  ENDIF
  IF(K.EQ.KX)THEN
  KDOWN=K+1
  KUP=K
  ENDIF
DO JJ = 0,JX,JBLOCK
  JSTART = JJ
  JSTOP = MIN(JSTART+JBLOCK,JX)
    IF(JJ.NE.0)THEN
    JSTART=JSTART+1
    ENDIF
```

MDH3D Restructured

```
      DO KK=KDOWN,KUP
      DO 200 J=JSTART,JSTOP
      DO 200 I=0,IX
         F(I,J,KK)=RVX(I,J,KK)
         G(I,J,KK)=RVY(I,J,KK)
         H(I,J,KK)=RVZ(I,J,KK)
         S(I,J,KK)=0.
200   CONTINUE
      ENDDO
      CALL HALF(JSTART,JSTOP,K,RO,ROH,DRO,F,G,H,S,0)
```

RESTRUCTURED HALF

```
      IF(K.GT.0.AND.K.LE.KXS1)THEN
      DO 100 J=MAX(1,JSTART),MIN(JXS1,JSTOP)
      DO 100 I=1,IXS1
         DU(I,J,K)=DU(I,J,K)-0.5*DT*
     &            (0.5*RDXM(I)*(F(I+1,J,K)-F(I-1,J,K))
     &            +0.5*RDYM(J)*(G(I,J+1,K)-G(I,J-1,K))
     &            +0.5*RDZM(K)*(H(I,J,K+1)-H(I,J,K-1))
     &            +S(I,J,K))
100   CONTINUE
      ENDIF
C=======================================================================
C***  proceed half step using flux across cell boundary               ***
C=======================================================================
```

RESTRUCTURED HALF

```
      IF(K.LT.KX)THEN
      DO 200 J=MAX(0,JSTART),MIN(JXS1,JSTOP)
      DO 200 I=0,IXS1
C----------- cell average ---------------------
        UH =0.125*(U(I+1,J+1,K+1)+U(I,J+1,K+1)
     &             +U(I+1,J+1,K)  +U(I,J+1,K)
     &             +U(I+1,J,K+1)  +U(I,J,K+1)
     &             +U(I+1,J,K)    +U(I,J,K))
        SH =0.125*(S(I+1,J+1,K+1)+S(I,J+1,K+1)
     &             +S(I+1,J+1,K)  +S(I,J+1,K)
     &             +S(I+1,J,K+1)  +S(I,J,K+1)
     &             +S(I+1,J,K)    +S(I,J,K))
C----------- flux across cell boundary ----------------------
        DFDX = 0.25*RDX(I)*(F(I+1,J+1,K+1)-F(I,  J+1,K+1)
     &                     +F(I+1,J+1,K)  -F(I,  J+1,K)
     &                     +F(I+1,J,  K+1)-F(I,  J,  K+1)
     &                     +F(I+1,J,  K)  -F(I,  J,  K))
        DGDY = 0.25*RDY(J)*(G(I+1,J+1,K+1)-G(I+1,J,  K+1)
     &                     +G(I+1,J+1,K)  -G(I+1,J,  K)
     &                     +G(I,  J+1,K+1)-G(I,  J,  K+1)
     &                     +G(I,  J+1,K)  -G(I,  J,  K))
        DHDZ = 0.25*RDZ(K)*(
     &                      H(I+1,J+1,K+1)-H(I+1,J+1,K)
     &                     +H(I+1,J,  K+1)-H(I+1,J,  K)
     &                     +H(I,  J+1,K+1)-H(I,  J+1,K)
     &                     +H(I,  J,  K+1)-H(I,  J,  K))
C------------ summation of all terms -----------------------
        UN(I,J,K)  = UH-DT*(DFDX+DGDY+DHDZ+SH)
200   CONTINUE
```

Storage Analysis

Restructured

Variables	NX	NY	NZ	Mwords	MB	L2	TLB	
Loop 200	7	259	32	2	.116	..935	2	..95
Half Do 100	5	259	32	2	0.08288	.66	1.32	.66
Half Do 200	6	259	32	2	0.099456	.79	1.6	.79

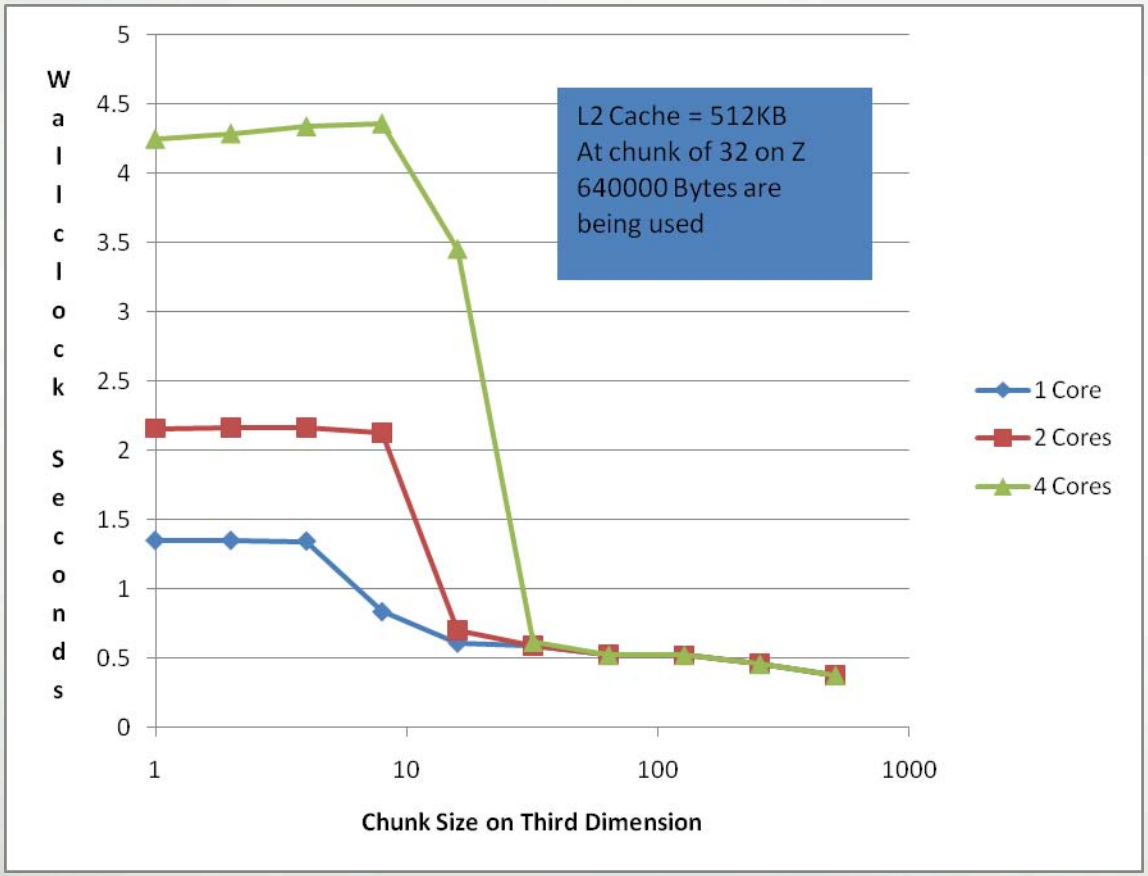
Simple Strip Mining loop

```
 integer, parameter :: nx=100, ny=100, nz=512, nc=100
  real(r4) a(nx,ny,nz),s
!..... initialize array a: a(ix,iy,iz)=ix+(nx*((iy-1)+ny*(iz-1)))
  in=1
  do il=1,10
  call system_clock(count=start_time)
  do ic=1,nc*in
    do iz=1,nz/in
    do iy=1,ny
    do ix=1,nx
      a(ix,iy,iz)=a(ix,iy,iz)*2.0
    end do
    end do
    end do
    do iz=1,nz/in
    do iy=1,ny
    do ix=1,nx
      a(ix,iy,iz)=a(ix,iy,iz)*0.5
    end do
    end do
    end do
  end do
  call system_clock(count=stop_time)
  in=in*2
  end do
  end
```

Storage Analysis

NX	NY	NZ	Ic	Mwords	MB	L1 Refills	L2 Refills	L3 Refills
100	100	512	1	5.12	40.96	625.00	81.92	40.96
100	100	256	2	2.56	20.48	312.50	40.96	20.48
100	100	128	4	1.28	10.24	156.25	20.48	10.24
100	100	64	8	0.64	5.12	78.13	10.24	5.12
100	100	32	16	0.32	2.56	39.06	5.12	2.56
100	100	16	32	0.16	1.28	19.53	2.56	1.28
100	100	8	64	0.08	0.64	9.77	1.28	0.64
100	100	4	128	0.04	0.32	4.88	0.64	0.32
100	100	2	256	0.02	0.16	2.44	0.32	0.16

Running code across 1, 2, and 4 cores



TLB Utilization

- Must be striding in array
  - Reorganize looping structures
- Use large pages

Background: Virtual Memory

- Modern programs operate in “virtual memory”
  - Each program thinks it has all of memory to itself
  - Fixed sized blocks (“pages”) vs variable sized blocks (“segments”)
- Virtual Memory benefits
  - Allow a program that is larger than physical memory to run
    - Programmer does not have to manually create overlays
  - Allow many programs to share limited physical memory
- Virtual Memory problems
  - Each virtual memory reference must be translated into a physical memory reference

Translation Speed

- Translation page table is stored in main memory
  - Each memory access logically takes twice as long – once to find the physical address, once to get the actual data
- Use a hardware cache of least recently used addresses
  - Called a Translation Lookaside Buffer or TLB

Performance Problem: TLB Refills

- AMD dual core opteron: 512 data TLB entries
- Covers 2MB of physical memory
  - OK if program fits (unlikely)
  - Large programs accessing data from all over their virtual memory range can trigger excessive TLB misses (“thrash”)
- One solution: huge pages


CRAY
THE SUPERCOMPUTER COMPANY