Porting/Optimizing POP to the X1

John M Levesque
Senior Technologist

Opening

- Creating the fastest POP in the world
  - Run the original
    - Gather Statistics
  - Optimize three heavily used routines
    - Gather Statistics
  - Modify solver to use Co-Array Fortran

POP's History

- Developed for the CM-5 at LANL
  - Written in Fortran 90
- Ported to Cache Based systems with disappointing results.
  - Fortran 90 was very Cache un-friendly
  - Partially rewritten to lay out data and combine array assignments into loops
- Still has significant Fortran 90 syntax – lots of vector content

POP's Structure

- Two major parts of the code
  - 3-D Baroclinic
    - Work intensive, highly scalable
  - 2-D Barotropic
    - Communication intensive, dominated by scalar reductions

Creating the fastest POP in the World

- Take a significant problem size which could be compared to data on other systems
  - X1 (Times one) Problem provided by LANL
  - Data on IBM 690 provided by ORNL, O3K by LANL
- Approach
  - A) Use Cray profiling tools to identify bottlenecks
  - B) Eliminate bottlenecks
  - C) Go To A

First Iteration

- Seventy Percent of Time spent in MAXLOC routine

NO LONGER NEEDED

Global_Real_Maxloc

```
val = -1.0E+30
 do j=jphys_b,jphys_e
   do i=iphys_b,iphys_e
     if (WORK(i,j) .gt. val) then
       val = WORK(i,j)
       local_val(1) = val
       local_val(2) = i_global(i)
       local_val(3) = j_global(j)
     endif
   end do
 end do
```

Analysis of Vectorization Listing

- Compiler did not recognize this as special function
- Restructure to facilitate recognition

Restructured code

```
val = -1.0E+30
   do j=jphys_b,jphys_e
     do i=iphys_b,iphys_e
       if (WORK(i,j) .gt. val) then
         val = WORK(i,j)
         isave = i
         jsave = j
       endif
     end do
   end do
         local_val(1) = val
         local_val(2) = i_global(isave)
         local_val(3) = j_global(jsave)
```

PAT_REPORT on SHMEM 4 Processor Run

```
25.5       25.5  11725   pe.1

  8.3        8.3   3832   impvmixt_IN_vertical_mix_
  6.4       14.8   2964   _sma_barrier_local
  2.9       17.6   1324   baroclinic_driver_IN_baroclinic_
  2.5       20.1   1143   impvmixu_IN_vertical_mix_
  0.5       20.6    229   boundary_2d_real_IN_boundary_
  0.5       21.1    221   tracer_update_IN_baroclinic_
  0.5       21.6    210   hdifft_del2_IN_hmix_del2_
  0.4       22.0    200   global_real_maxloc_IN_global_reduct:
  0.4       22.4    195   vmix_coeffs_rich_IN_vmix_rich_
  0.4       22.8    170   state_IN_state_mod_
  0.2       23.0    111   fivept_nobndy_IN_stencils_
  0.2       23.3    106   advu_IN_advection_
  0.2       23.5     94   clinic_IN_baroclinic_
  0.2       23.7     92   cfl_advect_IN_diagnostics_
  0.1       23.8     55   advt_IN_advection_
  0.1       23.9     54   vdiffu_IN_vertical_mix_
  0.1       24.0     52   ne_4pt_nobndy_IN_stencils_
  0.1       24.1     49   del2u_IN_operators_
```

Impvmixt(u)

```
do n = 1,nt
   mt2 = min(n,size(VDC,DIM=4))
   do j=jphys_b,jphys_e
   do i=iphys_b,iphys_e
     o
     o
     do k=2,KMT(i,j)
       c = a
       a = afac_t(k)*VDC(i,j,k,mt2)
       if (k == KMT(i,j)) then
         d = hfac_t(k)+b
       else
         d = hfac_t(k)+a+b
       endif
       e(k) = a/d
       b = (hfac_t(k) + b)*e(k)
       f(k) = (hfac_t(k)*TRACER(i,j,k,n,newtime) + c*f(k-1))/d
     end do
     do k=KMT(i,j)-1,1,-1
       f(k) = f(k) + e(k)*f(k+1)
     end do
     do k = 1,km
       TRACER(i,j,k,n,newtime) = TRACER(i,j,k,n,oldtime) + f(k)
     end do
   end do ! end of i-j loops
   end do
```

Compiler Listing

```
637.  1----------<       do n = nfirst,nlast
638.  1
639.  1                  mt2 = min(n,size(VDC,DIM=4))
640.  1
641.  1 2--------<        do j=jphys_b,jphys_e
642.  1 2 3------<        do i=iphys_b,iphys_e
643.  1 2 3
656.  1 2 3 Vs--<>         f(KMT(i,j)+1:km) = c0
657.  1 2 3
658.  1 2 3 Vpr--<         do k=2,KMT(i,j)
659.  1 2 3 Vpr
660.  1 2 3 Vpr            c = a
661.  1 2 3 Vpr            a = afac_t(k)*VDC(i,j,k,mt2)
662.  1 2 3 Vpr            if (k == KMT(i,j)) then
663.  1 2 3 Vpr              d = hfac_t(k)+b
664.  1 2 3 Vpr            else
665.  1 2 3 Vpr              d = hfac_t(k)+a+b
666.  1 2 3 Vpr            endif
667.  1 2 3 Vpr
668.  1 2 3 Vpr            e(k) = a/d
669.  1 2 3 Vpr            b = (hfac_t(k) + b)*e(k)
670.  1 2 3 Vpr            if (rhs_present) then
```

Compiler Listing (Cont)

```
671.  1 2 3 Vpr                 f(k) = c*f(k-1)/d
672.  1 2 3 Vpr               else
673.  1 2 3 Vpr                 f(k) = (hfac_t(k)*TRACER(i,j,k,n,newtime) + c*f(k-1))/d
674.  1 2 3 Vpr               endif
675.  1 2 3 Vpr
676.  1 2 3 Vpr-->            end do
677.  1 2 3
678.  1 2 3               !*** back substitution
679.  1 2 3
680.  1 2 3 r----<          do k=KMT(i,j)-1,1,-1
681.  1 2 3 r                 f(k) = f(k) + e(k)*f(k+1)
682.  1 2 3 r---->          end do
692.  1 2 3 Vs---<          do k = 1,km
693.  1 2 3 Vs                TRACER(i,j,k,n,newtime) =
694.  1 2 3 Vs          &      TRACER(i,j,k,n,update_index) + f(k)
695.  1 2 3 Vs--->          end do
696.  1 2 3
697.  1 2 3------>        end do ! end of i-j loops
698.  1 2-------->        end do
```

Restructure of impvmixt(u)

- Pull I loop on inside, streaming on j
  - Obtain best vectorization
  - Use Cray Streaming Directives to force streaming of j loop

Restructured impvmixt(u)

```
!CSD$ PARALLEL DO
!CSD$&PRIVATE(n,a,d,e,b,f,i,j,k,b,mt2)
        do j=jphys_b,jphys_e
      do n = 1,nt
          0  0  0
CDIR$ NOINTERCHANGE
          do k=2,km
CDIR$ CONCURRENT
        do i=iphys_b,iphys_e
          if(k.gt.KMT(i,j))then
          f(i,k) = c0
          else
            a(i,k) = afac_t(k)*VDC(i,j,k,mt2)
            if (k == KMT(i,j)) then
              d = hfac_t(k)+b(i,k-1)
            else
              d = hfac_t(k)+a(i,k)+b(i,k-1)
            endif
            e(i,k) = a(i,k)/d
            b(i,k) = (hfac_t(k) + b(i,k-1))*e(i,k)
            f(i,k) = (hfac_t(k)*TRACER(i,j,k,n,newtime)
     .                                  + a(i,k-1)*f(i,k-1))/d
          endif
          end do
          end do
          o  o   o
        end do ! end of n loop
        end do ! end of j loop
!CSD$ END PARALLEL DO
```

Compiler Listing

```
648.             !CSD$ PARALLEL DO
649.             !CSD$&PRIVATE(n,a,d,e,b,f,i,j,k,b,mt2)
650.  M---------<         do j=jphys_b,jphys_e
651.  M 2-------<       do n = nfirst,nlast
652.  M 2                mt2 = min(n,size(VDC,DIM=4))
653.  M 2 Vw----<          do i=iphys_b,iphys_e
658.  M 2 Vw               a(i,1) = afac_t(1)*VDC(i,j,1,mt2)
659.  M 2 Vw               d = hfac_t(1) + a(i,1)
660.  M 2 Vw               e(i,1) = a(i,1)/d
661.  M 2 Vw               b(i,1) = hfac_t(1)*e(i,1)
662.  M 2 Vw               if (rhs_present) then
663.  M 2 Vw               f(i,1) = hfac_t(1)*RHS(i,j,n)/d
664.  M 2 Vw               else
665.  M 2 Vw               f(i,1) = hfac_t(1)*TRACER(i,j,1,n,newtime)/d
666.  M 2 Vw               endif
667.  M 2 Vw---->          enddo
```

Compiler Listing (Cont.)

```
669.  M 2        CDIR$ NOINTERCHANGE
670.  M 2 3-----<          do k=2,km
671.  M 2 3       CDIR$ CONCURRENT
672.  M 2 3 V---<         do i=iphys_b,iphys_e
673.  M 2 3 V             if(k.gt.KMT(i,j))then
674.  M 2 3 V             f(i,k) = c0
675.  M 2 3 V             else
676.  M 2 3 V
677.  M 2 3 V               a(i,k) = afac_t(k)*VDC(i,j,k,mt2)
678.  M 2 3 V               if (k == KMT(i,j)) then
679.  M 2 3 V                 d = hfac_t(k)+b(i,k-1)
680.  M 2 3 V               else
681.  M 2 3 V                 d = hfac_t(k)+a(i,k)+b(i,k-1)
682.  M 2 3 V               endif
683.  M 2 3 V
```

HMIX_ANISO

1+ ,0
1+ ,0

A MB NE NQ KK N C K O K

(
0+3
(
(
(
(

,
R00 + + ( )D00 (, A)D11 (* B)D01 (
R11 + + (, A)D00 (*)D11 (, B)D01 (
R01 + + (B) D00 (, D11 ((* C)D01 (

(
2)E O Q + (
3)E ODQO + (!

2
3

(
(
(
(

0+3
R00 + + ( )D00 (, A)D11 (* B)D01 (
R11 + + (, A)D00 (*)D11 (, B)D01 (
R01 + + (B) D00 (, D11 ((* C)D01 (
D

, (
, (

HMIX_ANISO (Rest)

- Propagate numerous scalars to arrays
  - This will probably hurt cache performance – not as much as the original hurts vector performance
- Could also be done with array syntax

PCG - Solver

- Write Global Sum in Co-Array Fortran
- Write Ninept_4 in Co-Array Fortran

PCG - Solver

+A(

(+ + (MOQNB +)

Q+P

0+

NQ 0 Q) /Q

NQ / Q) NQ 0

0 NQ /+QB KB (

R NQ 0 * R) 0. /(

3 P+ /+ M+ D+ MD+R(

/ 0

NQ / P)R

0 /. NQ /+QB KB (

* 0)R

Q Q , 0)P

+ (- -/(

3 Q+ /+ M+ D+ MD+ (

Q A , Q

NQ / Q)Q

NQ /+QB KB (

- - (

0//

0//

Global_real_sum - MPI

```
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
                  /
                 +
                   +
                             *     + ()L R    + (


          LOH   KKQDC BD          +                 + 0+
                        LOH CN AKD OQDBHRHNM+ LOH R L+
                        LOH BNLL NBM+     (
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

Co-Array Global Sum

```
(
MOQNB  +) +                    MOQNB  +)
reduce_real_local = c0
  do j=jphys_b,jphys_e
    do i=iphys_b,iphys_e
      reduce_real_local = reduce_real_local + X(i,j)*MASK(i,j)
    end do
  end do
call sync_images()
if(my_pe().eq.0)then
  reduce_real_global = c0
    do j = 1,NPROC_Y
    do i = 1,NPROC_X
  reduce_real_global = reduce_real_global +reduce_real_local[i,j]
    enddo
    enddo
    do j = 1,NPROC_Y
    do i = 1,NPROC_X
       reduce_real_global[i,j] = reduce_real_global
    enddo
    enddo
endif
  call sync_images()
  global_sumx = reduce_real_global
```

Ninept_4

```
!***********************************************************************

   subroutine ninept_4x(XOUT,CC,CN,CE,CNE,X,me)


   real (kind=dbl_kind),dimension(imt,jmt)[NPROC_X,*],intent(out) ::
  &   XOUT              ! nine point operator result



   do j=jphys_b,jphys_e
     do i=iphys_b,iphys_e
       XOUT(i,j) = CC(i,j)*X(i  ,j  ) +
  &                 CN(i,j)*X(i  ,j+1) +  CN(i,j-1)*X(i  ,j-1) +
  &                 CE(i,j)*X(i+1,j  ) +  CE(i-1,j)*X(i-1,j  ) +
  &                CNE(i,j)*X(i+1,j+1) + CNE(i,j-1)*X(i+1,j-1) +
  &                CNE(i-1,j)*X(i-1,j+1) + CNE(i-1,j-1)*X(i-1,j-1)
     end do
   end do
```

Ninept_4 MPI

,,,

-

,,,

LOH HQDB N 0+0(+ 0+ + +
\+ LOH BNLL NBM+ 2(+ (
LOH HQDB N *0+0(+ 0+ + +
\+ LOH BNLL NBM+ 3(+ (

LOH HRDMC N *0, +0(+ 0+
\+ +
\+ LOH BNLL NBM+ 0(+ (
LOH HRDMC N +0(+ 0+ + +
\+ LOH BNLL NBM+ 1(+ (

LOH H KK 3+ + + (

LOH HQDB N 0+ *0(+ 0+ + +
\+ LOH BNLL NBM+ 2(+ (
LOH HQDB N 0+0(+ 0+ + +
\+ LOH BNLL NBM+ 3(+ (

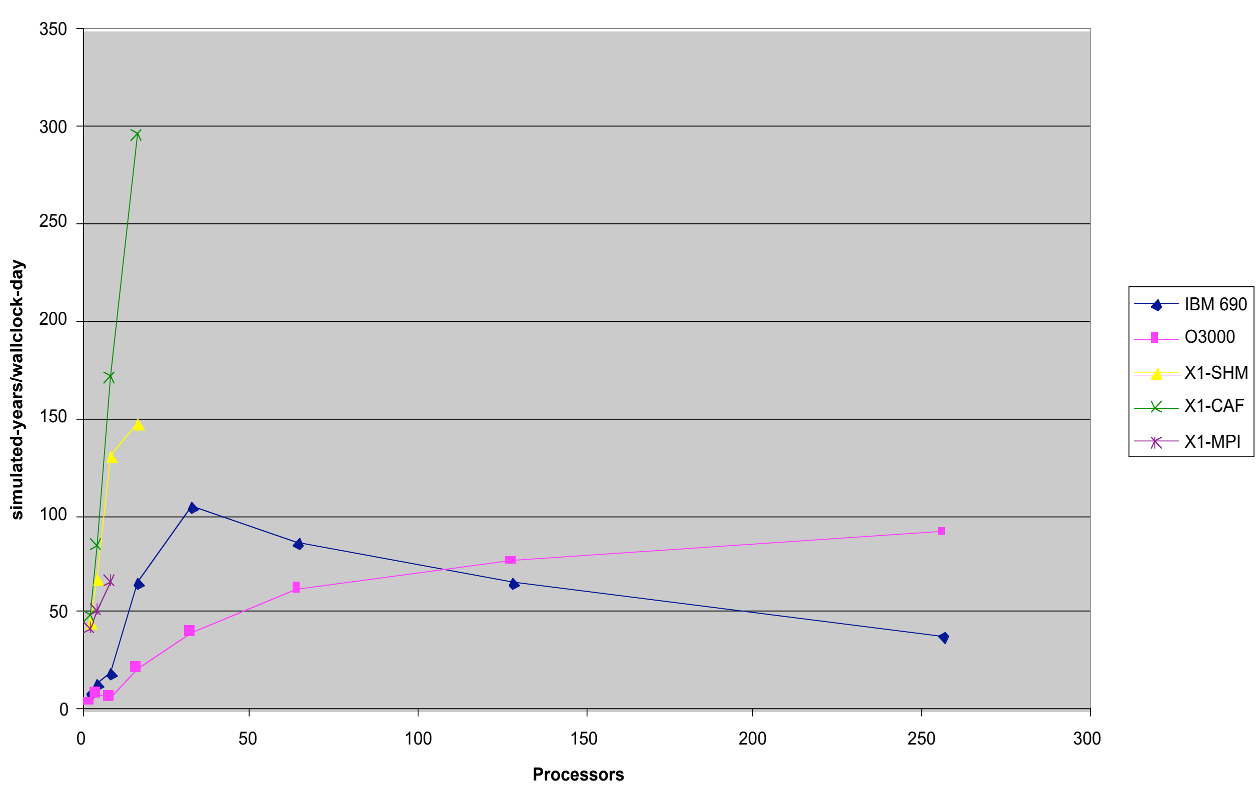
LOH HRDMC N 0+ (+ 0+ + +
\+ LOH BNLL NBM+ 0(+ (
LOH HRDMC N 0+ *0, (+ 0+
\+ +
\+ LOH BNLL NBM+ 1(+ (

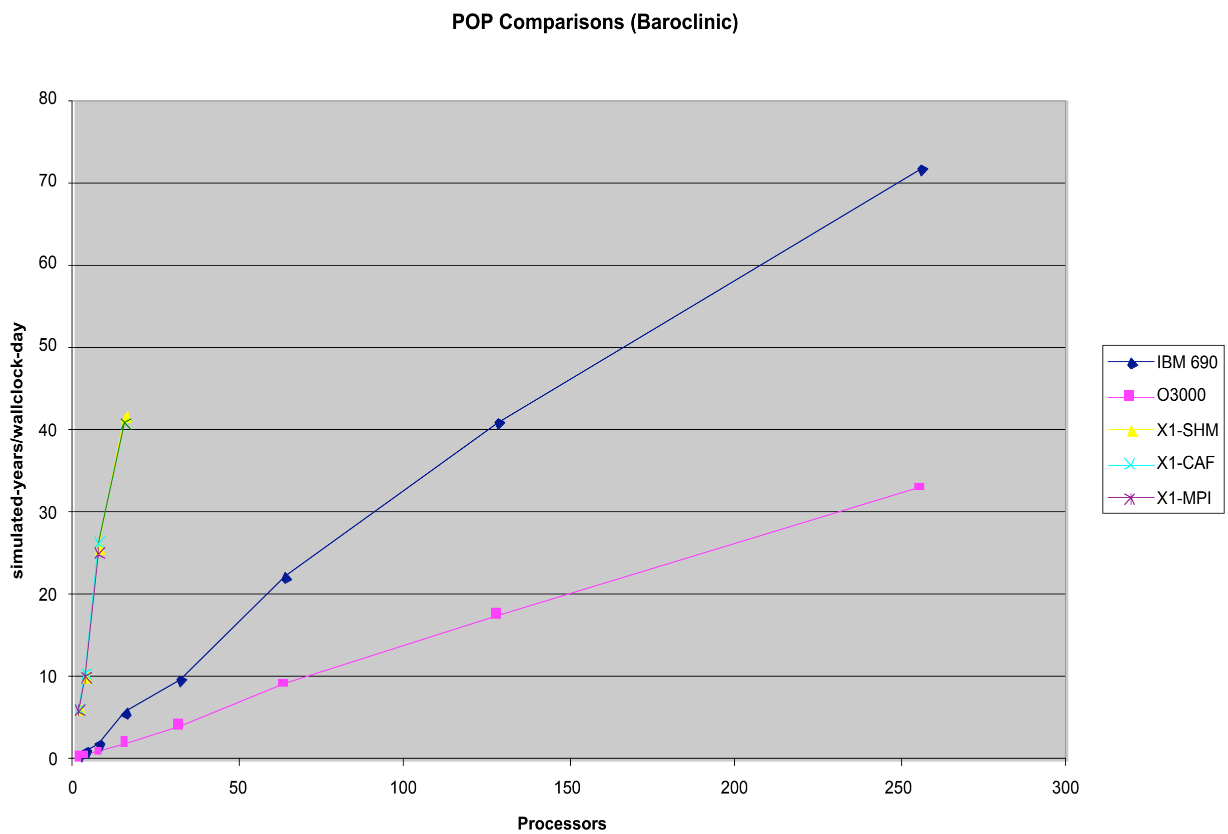
LOH H KK 3+ + + (

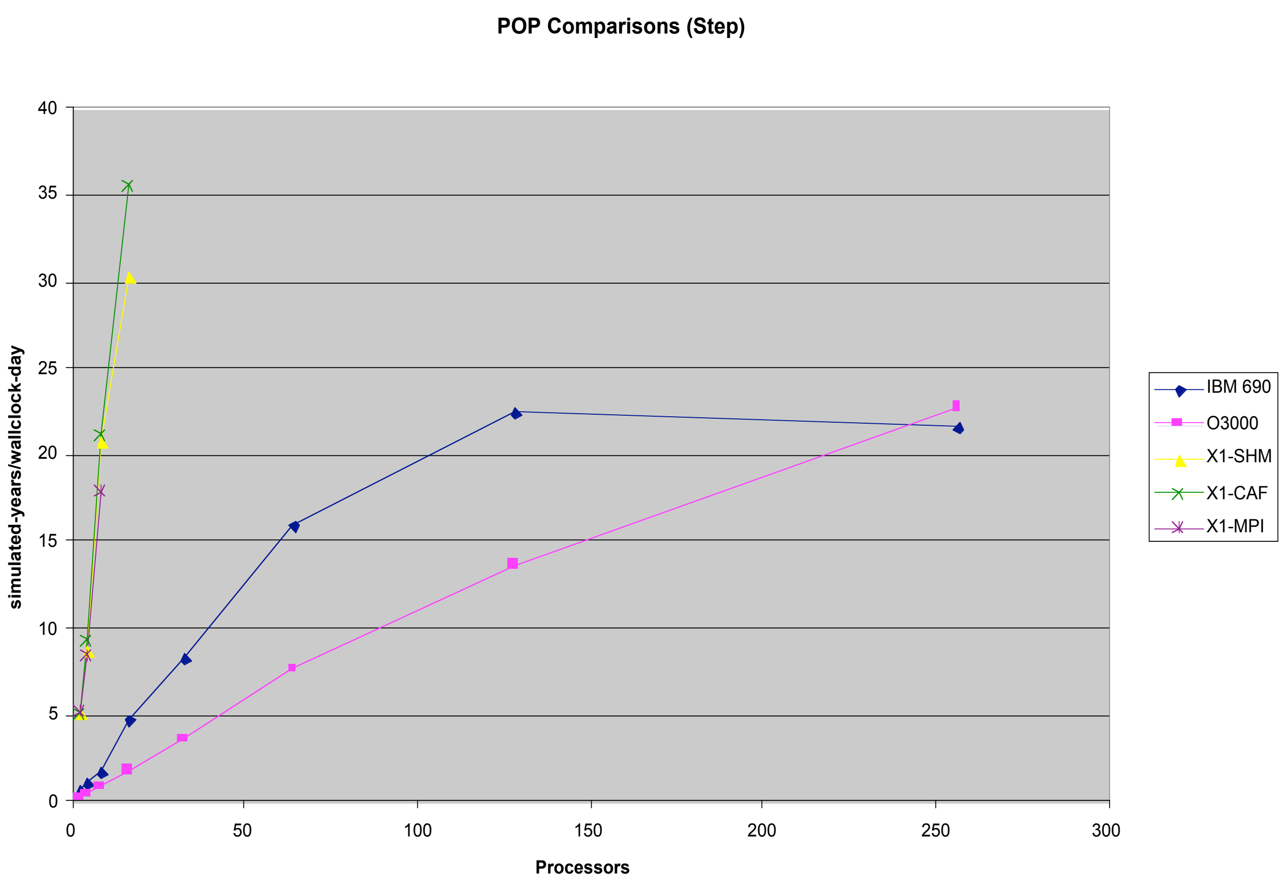
Co-Array Ninept_4 Communication

```
do n=1,num_ghost_cells
  do j=1,jmt
      XOUT(iphys_e+n,j) = XOUT(iphys_b+n-1,j)[me1p1,me(2)]
      XOUT(        n,j) = XOUT(iphys_e-num_ghost_cells+n,j)
 .                                              [me1m1,me(2)]
    end do
  end do
  call sync_images()
  do n=1,num_ghost_cells
    do i=1,imt
      XOUT(i,jphys_e+n) = XOUT(i,jphys_b+n-1)[me(1),me2p1]
      XOUT(i,        n) = XOUT(i,jphys_e-num_ghost_cells+n)
 .                                              [me(1),me2m1]
    end do
  end do
 call sync_images()
```

Pop Comparisons (Barotropic)




POP Comparisons (Baroclinic)
simulated-years/wallclock-day
Processors
0
10
20
30
40
50
60
70
80
0
50
100
150
200
250
300
IBM 690
O3000
X1-SHM
X1-CAF
X1-MPI


POP Comparisons (Step)
simulated-years/wallclock-day
Processors
0
5
10
15
20
25
30
35
40
0
50
100
150
200
250
300
IBM 690
O3000
X1-SHM
X1-CAF
X1-MPI

Running 1/10° Model

- 3600x2400x40
  - Used Large Benchmark pop_in
- Runs on 16 MSPs
- Results

```
 Timing information:

 Time in timer: IMPVMIXT
Timer number  1 =   688.03 seconds
 Time in timer: IMPVMIXU
Timer number  2 =     5.17 seconds
 Time in timer: TOTAL
Timer number  3 =   944.23 seconds
 Time in timer: STEP
Timer number  4 =   944.23 seconds
 Time in timer: BAROCLINIC
Timer number  5 =   881.32 seconds
 Time in timer: BAROTROPIC
Timer number  6 =    23.90 seconds
------------------------------------------------------------------------

 POP exiting...
 Successful completion of POP run
------------------------------------------------------------------------
```

Close

- There are numerous modes of computation on the X1. Multi-streaming/vectorization will obtain the best sustained performance
- When the best CPU performance is coupled with X1's best communication – remote addressing, significant improvements can be obtained
- Within POP these changes were not significant – less than 1% of the code.