Introduction to OpenMP

George Mozdzynski

George.Mozdzynski@ecmwf.int

ECMWF

Outline

- **What is OpenMP?**
- **Why OpenMP?**
- **OpenMP standard**
- **Key directives**
- **Using OpenMP on CRAY**
- **OpenMP parallelisation strategy**
- **Performance issues**
- **‘Automatic’ checking of OpenMP applications**
- **Class exercises**

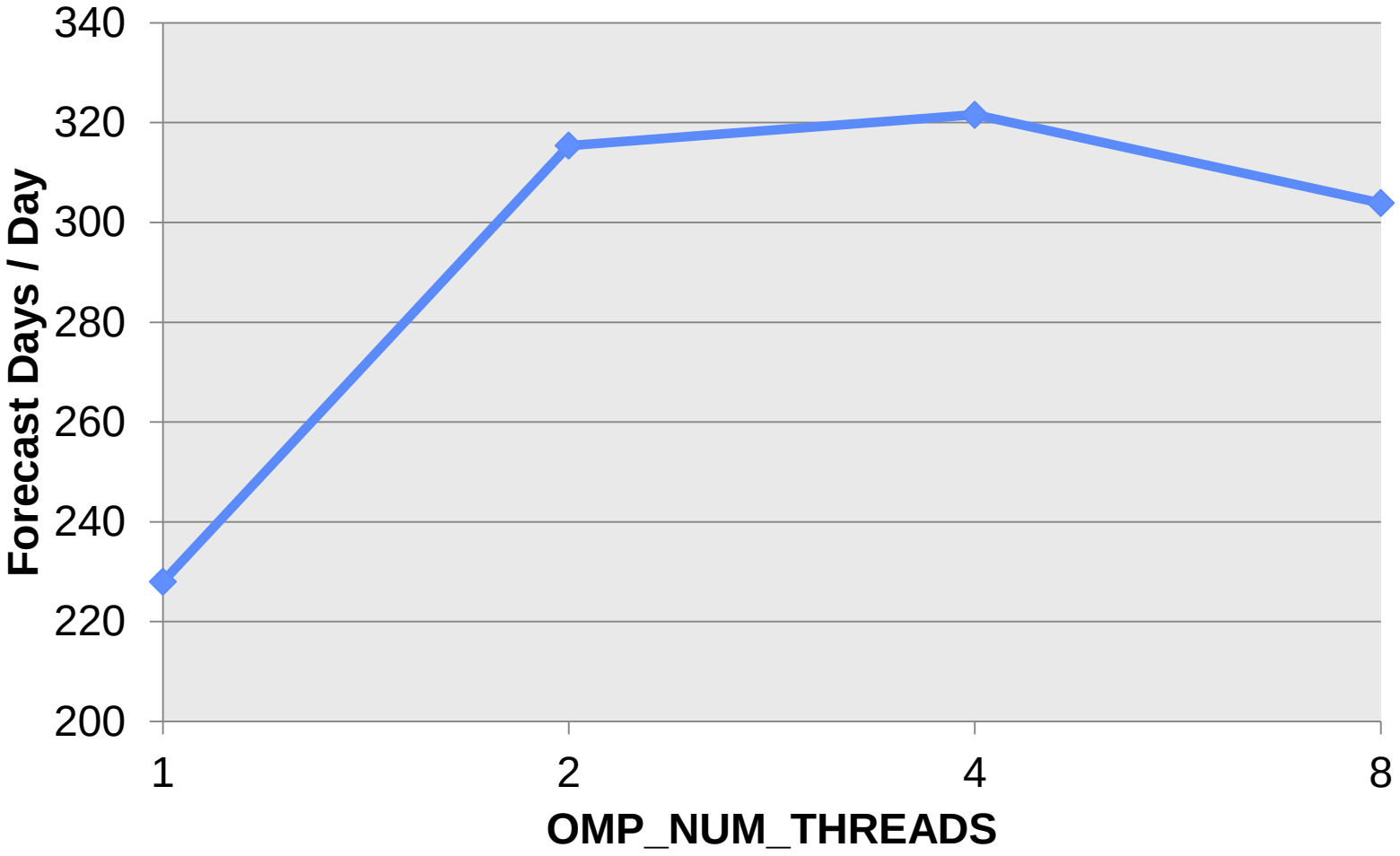
What is OpenMP?

- **OpenMP is a directive based language for expressing parallelism on Shared Memory Multiprocessor systems**
  - **Multiple threads of a task (process)**
  - **One or more tasks on a node**
- **F77, F90, F95 C, C++ supported**
- **www.openmp.org for info, specifications, FAQ, mail group**
- **CRAY ftn compiler supports OpenMP version 3.1**
  - **aprun … –d depth … omp_program**
- **http://openmp.org/wp/openmp-compilers/**

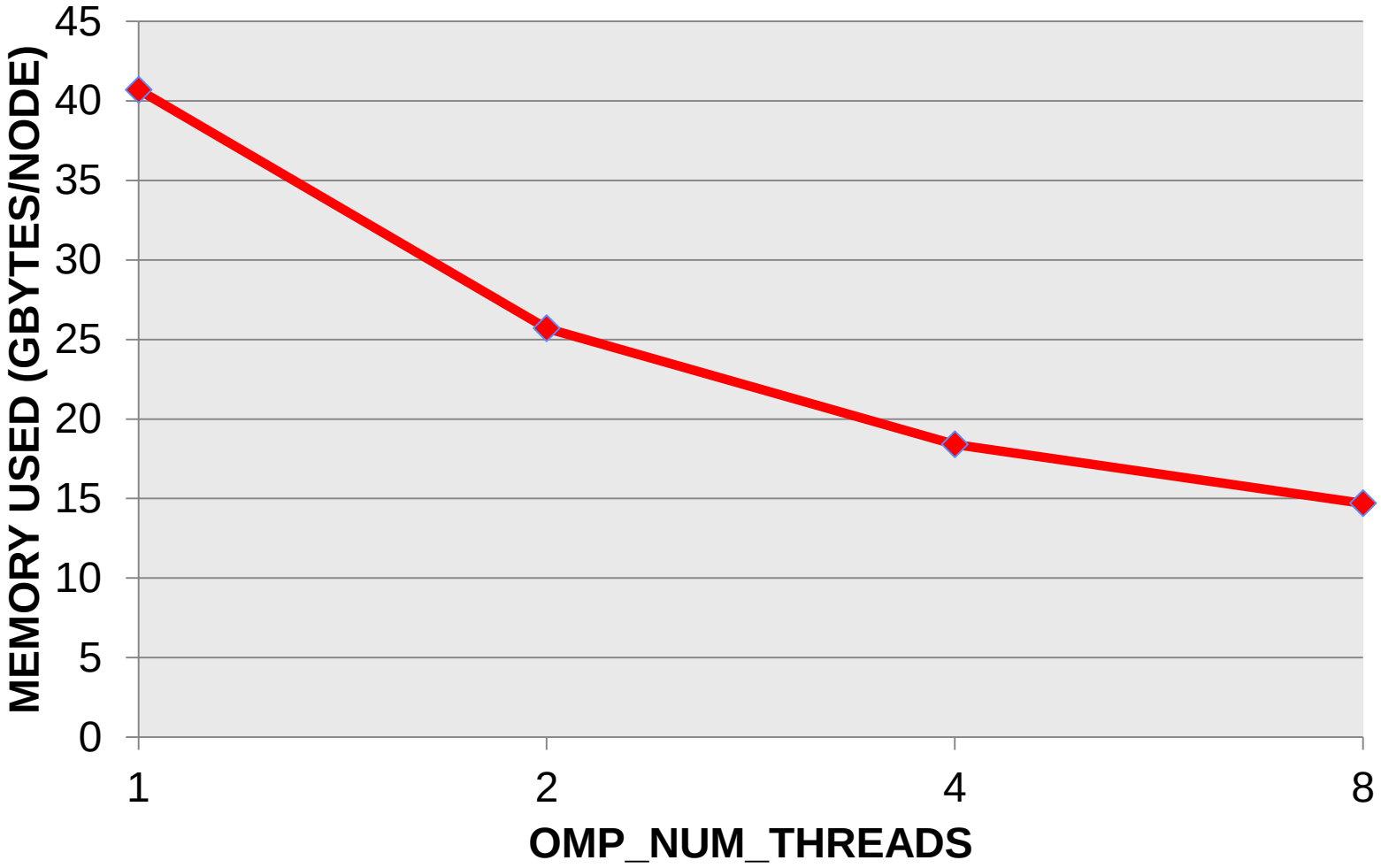
Why OpenMP?

- **Most new ‘supercomputers’ use shared memory multiprocessor technology**
- **MPI considered difficult to program**
- **SMP’s already had vendor specific directives for parallelisation**
- **OpenMP is a standard (just like MPI)**
- **OpenMP is relatively easy to use**
- **OpenMP can work together with MPI**
- **MPI/OpenMP v. MPI-only performance**

IFS T799L91 Forecast Model (1024 user threads)



IFS T799L91 Forecast Model (1024 user threads)



OpenMP directives

```
!$OMP PARALLEL DO PRIVATE(J,K,&
!$OMP& L,M)
  DO J=1,N
    CALL SUB(J,…)
  ENDDO
!$OMP END PARALLEL DO
```

See p7-9

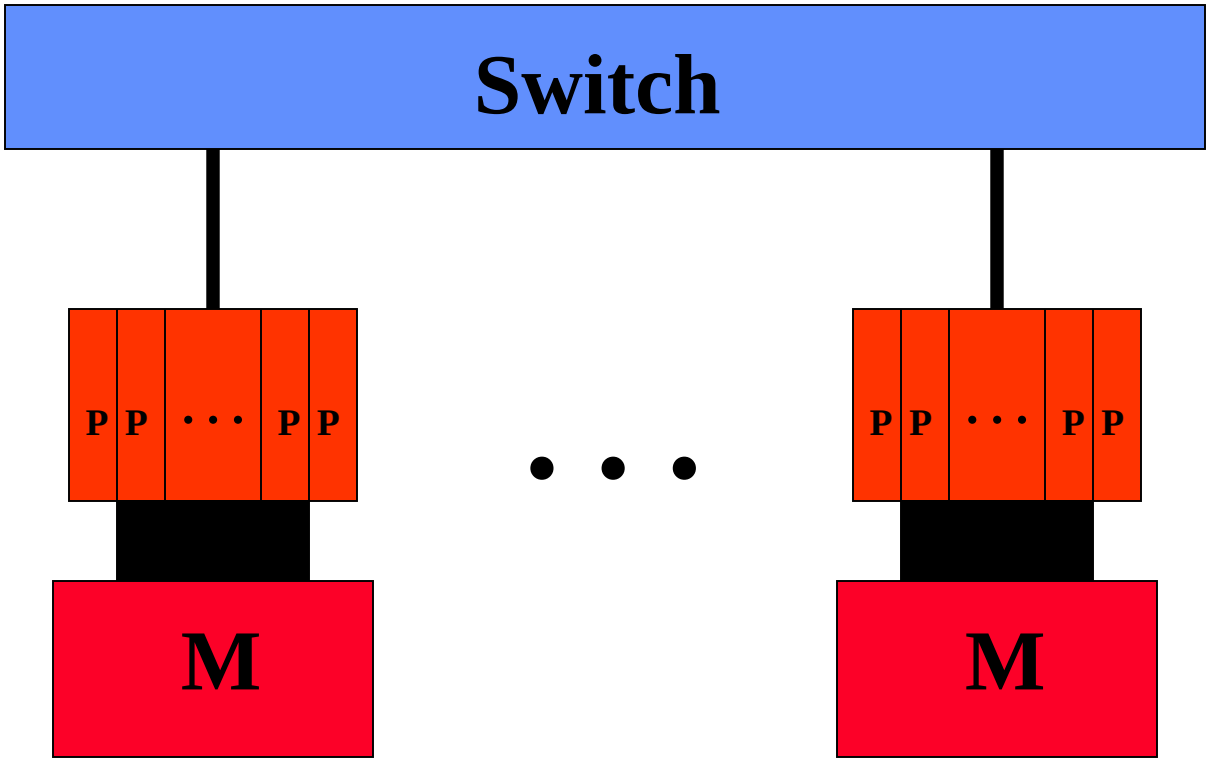
```
!$ ITHR=OMP_GET_MAX_THREADS()
```

Conditional compilation

e.g. ifsaux/module/yomoml.F90

OpenMP parallel programs are valid serial programs

Supercomputers today



```
#! /bin/ksh
#PBS -q np
#PBS -j oe
#PBS -N OMP1
#PBS -o omptest.out
#PBS -l EC_nodes=1
#PBS -l EC_total_tasks=1
#PBS -l EC_tasks_per_node=1
#PBS -l EC_threads_per_task=24
#PBS -l EC_hyperthreads=1
#PBS -l walltime=00:01:00

cd $PBS_O_WORKDIR

ftn -ra -homp -o omptest omptest1.F90

for omp in 1 2 4 8 16 24
do
  echo Using $omp threads
  export OMP_NUM_THREADS=$omp
  aprun -d $OMP_NUM_THREADS omptest
done
```

```
program omptest1
parameter(k=1000000)
real*8 timef,a(k),b(k)
a(:)=0.
b(:)=0.
write(0,'("start timing")')
zbeg=timef()*1.0e-3
do irep=1,10000
  call work(k,a,b)
enddo
zend=timef()*1.0e-3
write(0,'("end timing")')
write(*,'("time=",F10.3)')zend-zbeg
stop
end

subroutine work(k,a,b)
real*8 a(k),b(k)
!$OMP PARALLEL DO PRIVATE(I)
do i=1,k
  a(i)=b(i)
enddo
!$OMP END PARALLEL DO
return
end
```

Threads	time	s/u
1	5.46	1.00
2	2.79	1.96
4	1.58	3.46
8	0.90	6.07
16	0.44	12.41
32	0.26	21.00

16 MB problem fits in L3 cache (2 sockets x 30 MB)

```
program omptest1
parameter(k=10000000)
real*8 a(k),b(k)
a(:)=0.
b(:)=0.
write(0,'("start timing")')
zbeg=timef()*1.0e-3
do irep=1,200
  call work(k,a,b)
enddo
zend=timef()*1.0e-3
write(0,'("end timing")')
write(*,'("time=",F10.2)')zend-zbeg
stop
end

subroutine work(k,a,b)
real*8 a(k),b(k)
!$OMP PARALLEL DO PRIVATE(I)
do i=1,k
  a(i)=b(i)
enddo
!$OMP END PARALLEL DO
return
end
```

Threads	time	s/u
1	3.92	1.00
2	2.02	1.94
4	1.10	3.56
8	0.68	5.96
16	0.97	4.04
24	1.02	3.84

160 MB problem, no longer fits in L3 Caches on node (60 MB)

```
K=10M, IREP=200
a(I)=b(I)*cos(b(I))*sin(b(I))
```

Threads	time	s/u
1	24.45	1.00
2	12.73	1.92
4	6.78	3.61
8	3.63	6.73
16	1.82	13.43
24	1.37	17.84

OpenMP performance tip: For perfect speedup avoid using memory ☺

Key directives – Parallel Region

```
!$OMP PARALLEL [clause,[clause…]]
  block
!$OMP END PARALLEL
```

See p12-14

Where *clause* can be

- PRIVATE(*list*)
- etc.,

Key directives – Work-sharing constructs/1

```
!$OMP DO [clause,[clause…]]
  do_loop
!$OMP END DO
```

See p15-18

Where *clause* can be

- **PRIVATE(*list*)**
- **SCHEDULE(*type*[,*chunk*])**
- **etc.,**

Key directives – Work-sharing constructs/2

```
!$OMP WORKSHARE
  block
!$OMP END WORKSHARE
```

See p20-22

No PRIVATE or SCHEDULE options

A good example for *block* would be array assignment statements (I.e. no DO)

Key directives – combined parallel work-sharing/1

```
!$OMP PARALLEL DO [clause,[clause…]]
  do_loop
!$OMP END PARALLEL DO
```

See p23

Where *clause* can be

- **PRIVATE(*list*)**
- **SCHEDULE(*type*[,*chunk*])**
- **etc.,**

Key directives – combined parallel work-sharing/2

```
!$OMP PARALLEL WORKSHARE [clause,[clause…]]
  block
!$OMP END PARALLEL WORKSHARE
```

See p24-25

Where *clause* can be

- PRIVATE(*list*)
- etc.,

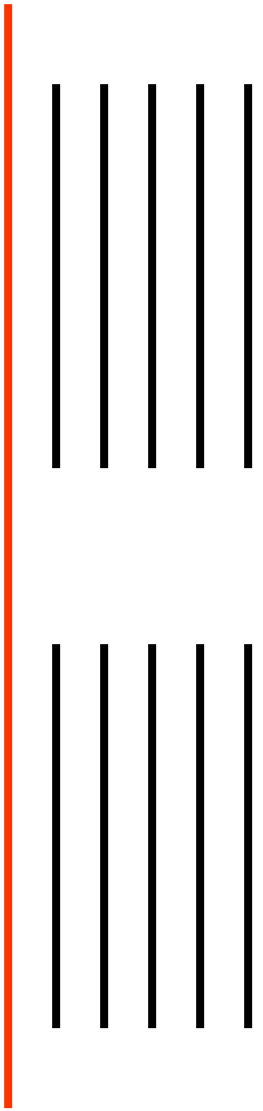
K=10M, IREP=200 (on IBM Power7)

```
!$OMP PARALLEL DO PRIVATE(I)
DO I=1,K
  A(I)=B(I)*COS(C(I))
ENDDO
!$OMP END PARALLEL DO
```

```
!$OMP PARALLEL WORKSHARE
  A(:)=B(:)*COS(C(:))
!$OMP END PARALLEL WORKSHARE
```

Threads	time	s/u	time	s/u
1	49.10	1.0	49.10	1.0
2	24.54	2.0	24.53	2.0
4	12.29	4.0	12.28	4.0
8	6.14	8.0	6.13	8.0
16	3.09	15.9	3.07	16.0
32	1.70	28.9	1.62	30.3

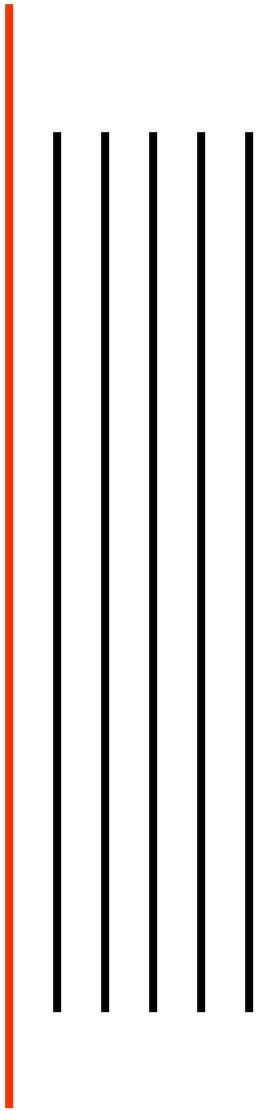
Master thread



```
!$OMP PARALLEL DO ...
      DO J=1,NWORK1
         ...
      ENDDO
!$OMP END PARALLEL DO

!$OMP PARALLEL DO ...
      DO J=1,NWORK2
         ...
      ENDDO
!$OMP END PARALLEL DO
```

Master thread



```
!$OMP PARALLEL ...
!$OMP DO ...
      DO J=1,NWORK1
         ...
      ENDDO
!$OMP END DO
!$OMP DO ...
      DO J=1,NWORK2
         ...
      ENDDO
!$OMP END DO
!$OMP END PARALLEL
```

```
program omptest2

INTEGER OMP_GET_THREAD_NUM
INTEGER OMP_GET_NUM_THREADS
INTEGER OMP_GET_MAX_THREADS
!$ WRITE(0,'("NUM_THREADS=",I3)')OMP_GET_NUM_THREADS()
!$ WRITE(0,'("MAX_THREADS=",I3)')OMP_GET_MAX_THREADS()
!$OMP PARALLEL DO SCHEDULE(STATIC) PRIVATE(J,ITH,ITHS)
DO J=1,10
!$ ITH=OMP_GET_THREAD_NUM()
!$ ITHS=OMP_GET_NUM_THREADS()
!$ WRITE(0,'("J=",I3," THREAD_NUM=",I3," NUM_THREADS=",I3)')J,ITH,ITHS
ENDDO
!$OMP END PARALLEL DO

stop
end
```

```
export OMP_NUM_THREADS=4

aprun -d 4 omptest
```

```
NUM_THREADS=  1
MAX_THREADS=  4
J=  1 THREAD_NUM=  0 NUM_THREADS=  4
J=  2 THREAD_NUM=  0 NUM_THREADS=  4
J=  3 THREAD_NUM=  0 NUM_THREADS=  4
J=  4 THREAD_NUM=  1 NUM_THREADS=  4
J=  7 THREAD_NUM=  2 NUM_THREADS=  4
J=  8 THREAD_NUM=  2 NUM_THREADS=  4
J=  5 THREAD_NUM=  1 NUM_THREADS=  4
J=  6 THREAD_NUM=  1 NUM_THREADS=  4
J=  9 THREAD_NUM=  3 NUM_THREADS=  4
J= 10 THREAD_NUM=  3 NUM_THREADS=  4
```

```
!$OMP PARALLEL PRIVATE(JKGLO,ICEND,IBL,IOFF)
!$OMP DO SCHEDULE(DYNAMIC,1)
    DO JKGLO=1,NGPTOT,NPROMA
      ICEND=MIN(NPROMA,NGPTOT-JKGLO+1)
      IBL=(JKGLO-1)/NPROMA+1
      IOFF=JKGLO

      CALL EC_PHYS(NCURRENT_ITER,LFULLIMP,LNHDYN,CDCONF(4:4)&
                  &,IBL,IGL1,IGL2,ICEND,JKGLO,ICEND &
                  &,GPP(1,1,IBL),GEMU(IOFF)&
                  &,GELAM(IOFF),GESLO(IOFF),GECLO(IOFF),GM(IOFF)&
                  &,OROG(IOFF),GNORDL(IOFF),GNORDM(IOFF)&
                  &,GSQM2(IOFF),RCOLON(IOFF),RSILON(IOFF)&
                  &,RINDX(IOFF),RINDY(IOFF),GAW(IOFF)&

                           ...

                  &,GPBTP9(1,1,IBL),GPBQP9(1,1,IBL)&
                  &,GPFORCEU(1,1,IBL,0),GPFORCEV(1,1,IBL,0)&
                  &,GPFORCET(1,1,IBL,0),GPFORCEQ(1,1,IBL,0))

    ENDDO
!$OMP END DO
!$OMP END PARALLEL
```

ifs/control/gp_model.F90

```
!$OMP PARALLEL DO SCHEDULE(STATIC,1)&
!$OMP& PRIVATE(JMLOCF,IM,ISTA,IEND)
      DO JMLOCF=NPTRMF(MYSETN),NPTRMF(MYSETN+1)-1
        IM=MYMS(JMLOCF)
        ISTA=NSPSTAF(IM)
        IEND=ISTA+2*(NSMAX+1-IM)-1
        CALL SPCSI(CDCONF,IM,ISTA,IEND,LLONEM,ISPEC2V,&
         &ZSPVORG,ZSPDIVG,ZSPTG,ZSPSPG)
      ENDDO
!$OMP END PARALLEL DO
```

ifs/control/spcm.F90

```
!$OMP PARALLEL PRIVATE(JKGLO,ICEND,IBL,IOFF,ZSLBUF1AUX,JFLD,JROF)
  IF (.NOT.ALLOCATED(ZSLBUF1AUX))ALLOCATE (ZSLBUF1AUX(NPROMA,NFLDSLB1))
!$OMP DO SCHEDULE(DYNAMIC,1)

  DO JKGLO=1,NGPTOT,NPROMA
    ICEND=MIN(NPROMA,NGPTOT-JKGLO+1)
    IBL=(JKGLO-1)/NPROMA+1
    IOFF=JKGLO
    ZSLBUF1AUX(:,:)=_ZERO_
    CALL CPG25(CDCONF(4:4) &
       &,ICEND,JKGLO,NGPBLKS,ZSLBUF1AUX,ZSLBUF2X(1,1,IBL) &
       &,RCORI(IOFF),GM(IOFF),RATATH(IOFF),RATATX(IOFF)&
           ...
       &,GT5(1,MSPT5M,IBL))
!        move data from blocked form to latitude (NASLB1) form
    DO JFLD=1,NFLDSLB1
      DO JROF=JKGLO,MIN(JKGLO-1+NPROMA,NGPTOT)
        ZSLBUF1(NSLCORE(JROF),JFLD)=ZSLBUF1AUX(JROF-JKGLO+1,JFLD)
      ENDDO
    ENDDO
  ENDDO

!$OMP END DO
  IF (ALLOCATED(ZSLBUF1AUX)) DEALLOCATE(ZSLBUF1AUX)
!$OMP END PARALLEL
```

ifs/control/gp_model_ad.F90

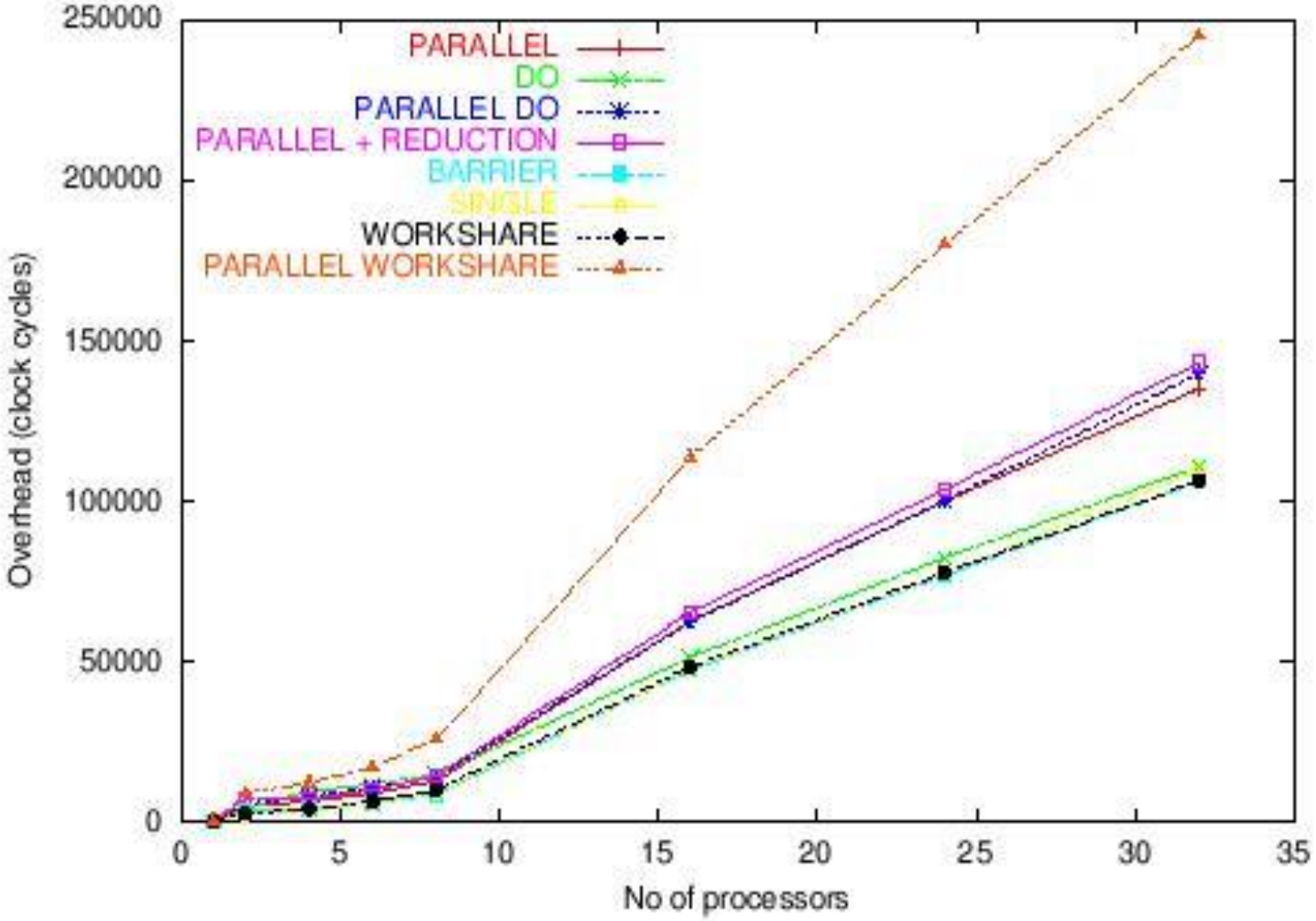
OpenMP parallelisation strategy

- **Start with a correct serial execution of the application**
- **Apply OpenMP directives to time-consuming do loops one at a time and TEST**
- **Use high level approach where possible**
- **Use ‘thread checker’ (Intel Inspector) to perform a correctness check**
- **Results may change slightly**
- **Results should be bit reproducible for different numbers of threads**
- **Avoid reductions for reals (except: max, min)**
  - **as they cause different results for different #’s of threads**
- **Fortran array syntax supported with WORKSHARE**

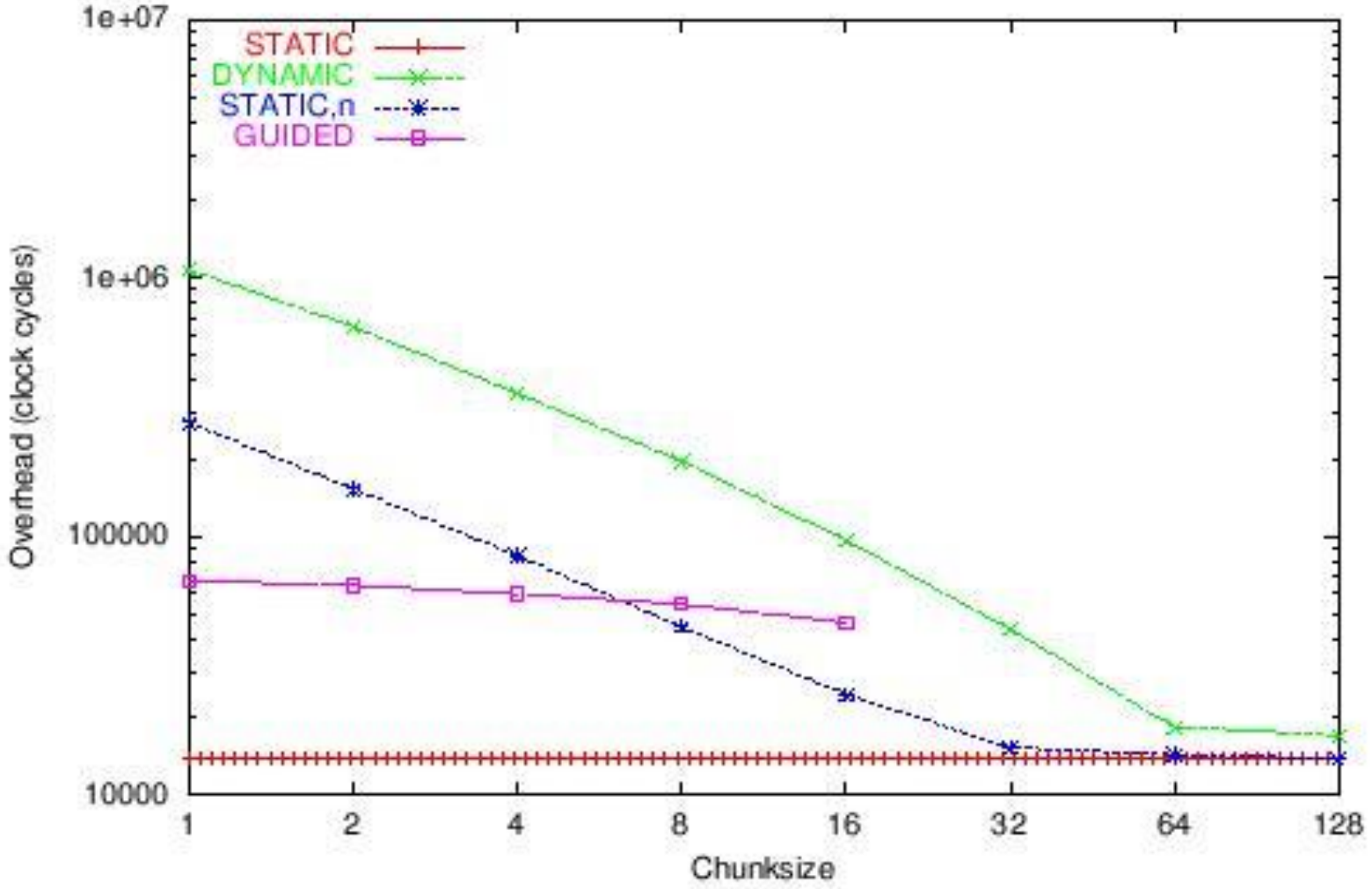
Performance issues

- **Amdahl's Law**
- **Work imbalance**
- **False sharing (cache line ping-pong)**
- **Memory overheads**
- **OpenMP overheads**
  - **Fiona J.L. Reid and J. Mark Bull, HPCx UK**
  - **http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0411.pdf**
  - **Copied to OpenMP_Course/SPECS/**

Synchronisation overheads on p690+ (Power4)



SCHEDULE overheads on P690+ (8 CPUs)



Store1: Is this safe?

```
   ...
!$OMP PARALLEL DO PRIVATE(J)
   DO J=1,N
      CALL WORK(J)
   ENDDO
!$OMP END PARALLEL DO
   ...

SUBROUTINE WORK(K)
USE MYMOD, ONLY : X, A
INTEGER K
X=A(K)
CALL SUB(X)
END
```

Store1: Is this safe?

```
   ...
!$OMP PARALLEL DO PRIVATE(J)
   DO J=1,N
      CALL WORK(J)
   ENDDO
!$OMP END PARALLEL DO
   ...

SUBROUTINE WORK(K)
USE MYMOD, ONLY : X, A
INTEGER K
X=A(K)
CALL SUB(X)
END
```

NO, X has a Write/Read conflict

Store2: Is this safe?

```
   ...
!$OMP PARALLEL DO PRIVATE(J)
   DO J=1,N
      CALL WORK(J)
   ENDDO
!$OMP END PARALLEL DO
   ...

SUBROUTINE WORK(K)
USE MYMOD, ONLY : X, A
INTEGER K
A(K)=X
CALL SUB(X)
END
```

Store2: Is this safe?

```
   ...
!$OMP PARALLEL DO PRIVATE(J)
   DO J=1,N
      CALL WORK(J)
   ENDDO
!$OMP END PARALLEL DO
   ...

SUBROUTINE WORK(K)
USE MYMOD, ONLY : X, A
INTEGER K
A(K)=X
CALL SUB(X)
END
```

YES, different locations in A are being written to

- It is unsafe to store to the same location (scalar or array) from multiple threads of a parallel region

- A safe strategy would be to
  - only read shared variables
  - only write to non-shared variables
    - parallel region PRIVATE variables
    - subroutine local variables within parallel region
    - threads writing to different memory locations in a module

Reduction: example 1

```
   ...
!$OMP PARALLEL DO PRIVATE(J)
  DO J=1,N
    CALL WORK(J)
  ENDDO
!$OMP END PARALLEL DO
   ...

SUBROUTINE WORK(K)
USE MYMOD, ONLY : M
...
IVAL= time consuming calc
!$OMP CRITICAL
M=M+IVAL
!$OMP END CRITICAL
...
END
```

Reduction: example 2

```
USE MYMOD, ONLY : M
   ...
IVAL=0
!$OMP PARALLEL DO PRIVATE(J) REDUCTION(+:IVAL)
  DO J=1,N
     CALL WORK(J,IVAL)
  ENDDO
!$OMP END PARALLEL DO
M=M+IVAL
   ...

SUBROUTINE WORK(K,KVAL)
...
IVAL = time consuming calc
KVAL = KVAL + IVAL
...
END
```

Reduction: example 3

```
USE MYMOD, ONLY : M
INTEGER IVAL(N)
   ...
!$OMP PARALLEL DO PRIVATE(J)
  DO J=1,N
    CALL WORK(J,IVAL(J))
  ENDDO
!$OMP END PARALLEL DO
M=M+SUM(IVAL)
   ...

SUBROUTINE WORK(K,KVAL)
...
KVAL = time consuming calc
...
END
```

Critical Region: Is this safe?

```
IF( L_DO_ONCE )THEN

!$OMP CRITICAL REGION

  <work that must only be done once>

  L_DO_ONCE=.FALSE.

!$OMP END CRITICAL REGION

ENDIF
```

L_DO_ONCE is initialised to .TRUE. outside of the parallel region where this block of code appears.

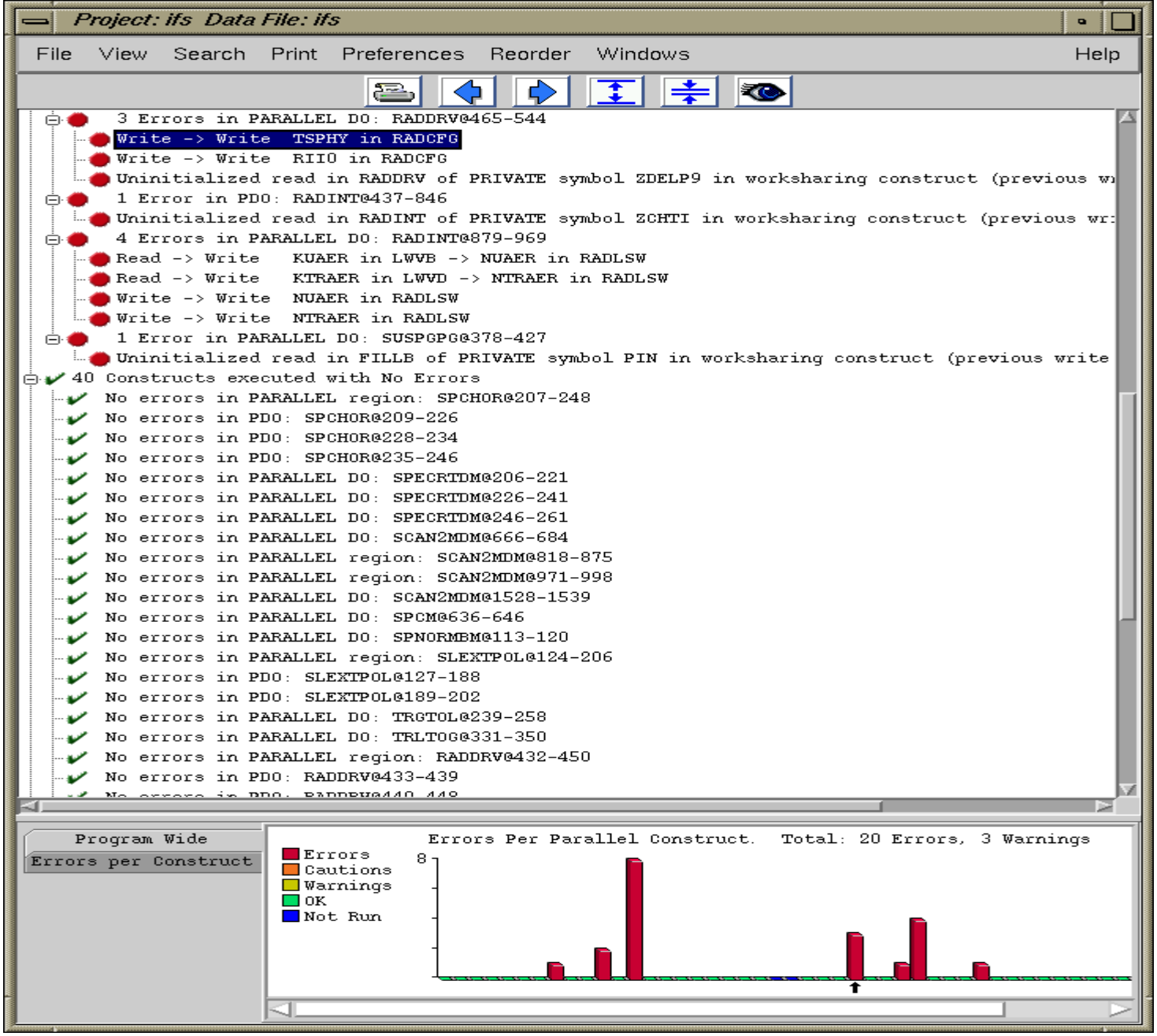
ANSWER is NO! Why?

CRITICAL REGION: Correct coding

```
!$OMP CRITICAL REGION
IF( L_DO_ONCE )THEN
  <work that must only be done once>
  L_DO_ONCE=.FALSE.
ENDIF
!$OMP END CRITICAL REGION
```

Intel Inspector (Thread Checker)

- Analyse correctness of an OpenMP application
- Can identify any place where a construct exists that may violate the single-threaded consistency property
- Simulates multiple threads (but uses 1 thread)
  - > 100 times slower than ‘real’ time
- No equivalent product from IBM/CRAY yet
- Following IFS example from KAI’s Assure product, bought by INTEL 10+ years ago


Project: ifs Data File: ifs
File View Search Print Preferences Reorder Windows Help
3 Errors in PARALLEL DO: RADDRV@465-544
Write -> Write TSPHY in RADCFG
Write -> Write RIIO in RADCFG
Uninitialized read in RADDRV of PRIVATE symbol ZDELP9 in worksharing construct (previous w
1 Error in PDO: RADINT@437-846
Uninitialized read in RADINT of PRIVATE symbol ZCHTI in worksharing construct (previous wr
4 Errors in PARALLEL DO: RADINT@879-969
Read -> Write KUAER in LWVB -> NUAER in RADLSW
Read -> Write KTRAER in LWVD -> NTRAER in RADLSW
Write -> Write NUAER in RADLSW
Write -> Write NTRAER in RADLSW
1 Error in PARALLEL DO: SUSPGPG@378-427
Uninitialized read in FILLB of PRIVATE symbol PIN in worksharing construct (previous write
40 Constructs executed with No Errors
No errors in PARALLEL region: SPCHOR@207-248
No errors in PDO: SPCHOR@209-226
No errors in PDO: SPCHOR@228-234
No errors in PDO: SPCHOR@235-246
No errors in PARALLEL DO: SPECRTDM@206-221
No errors in PARALLEL DO: SPECRTDM@226-241
No errors in PARALLEL DO: SPECRTDM@246-261
No errors in PARALLEL DO: SCAN2MDM@666-684
No errors in PARALLEL region: SCAN2MDM@818-875
No errors in PARALLEL region: SCAN2MDM@971-998
No errors in PARALLEL DO: SCAN2MDM@1528-1539
No errors in PARALLEL DO: SPCM@636-646
No errors in PARALLEL DO: SPNORMBM@113-120
No errors in PARALLEL region: SLEXTPOL@124-206
No errors in PDO: SLEXTPOL@127-188
No errors in PDO: SLEXTPOL@189-202
No errors in PARALLEL DO: TRGTOL@239-258
No errors in PARALLEL DO: TRLTOG@331-350
No errors in PARALLEL region: RADDRV@432-450
No errors in PDO: RADDRV@433-439
Program Wide
Errors per Construct
Errors Per Parallel Construct. Total: 20 Errors, 3 Warnings
Errors
Cautions
Warnings
OK
Not Run
8

Stack issues (CRAY)

- **Master thread stack inherits task (process) stack**
- **Non-master thread stacks**
  - **Default on CRAY 128 Mbytes (was 4 Mbytes on IBM)**
  - **OMP_STACKSIZE=256M to increase to 256MBytes**
- **Large arrays (>1 Mbyte?) should use the heap**

```
real,allocatable :: biggy(:)
allocate(biggy(100000000))
        …
deallocate(biggy)
```

```
> ulimit -a
address space limit (Kibytes)  (-M)  52857040
core file size (blocks)        (-c)  unlimited
cpu time (seconds)             (-t)  unlimited
data size (Kibytes)            (-d)  unlimited
file size (blocks)             (-f)  unlimited
locks                          (-x)  unlimited
locked address space (Kibytes) (-l)  unlimited
message queue size (Kibytes)   (-q)  800
nice                           (-e)  0
nofile                         (-n)  16000
nproc                          (-u)  516081
pipe buffer size (bytes)       (-p)  4096
max memory size (Kibytes)      (-m)  63221760
rtprio                         (-r)  0
socket buffer size (bytes)     (-b)  4096
sigpend                        (-i)  516081
stack size (Kibytes)           (-s)  unlimited
swap size (Kibytes)            (-w)  not supported
threads                        (-T)  not supported
process size (Kibytes)         (-v)  52857040
```

available memory per node ~ 54 Gbytes
i.e. 54,000,000,000 bytes


And now the
exercises