Empowered by Innovation NEC

Run-time environment

Environment variables

F_FTRACE – C_FTRACE = NO/YES/FMT1/FMT2

Enable flow trace summary, define format.

F_PROGINF – C_PROGINF (MPIPROGINF)

= NO/YES/DETAIL/(ALL_DETAIL)

Enable hardware statistics output.

F_RECLUNIT = WORD/BYTE

RECL specified in WORD/BYTE.

Environment variables

F_SETBUF = n

F_SETBUFu = n

Set I/O buffers to a value of n KB (globally / for unit u).

F_SYSLEN = n

Set max line length for stdout to n characters.

…and many others:

Fortran90/SX Programmer's Guide, chapter 4

Empowered by Innovation NEC

Some popular traps

- Unexpected zero divides
- Incorrect assumed vector length
- Insufficient record size for formatted output
- pre-defined CPP variables

unexpected zero divides

FORTRAN:

```
program t
real a(100)
data a / 100 * 1.0 /
b = 0.0
…
do I = 1, 100
  if( a(I) .eq. 0.0 ) a(I) = a(I) / b
end do
print *, a(1)
end
```

- **b** is constant, compiler will move expression outside of the loop
- Solution:
  `-Wf "-O nomove"`

Incorrect assumed vector length

FORTRAN:

```
program t
real a(1000)
call sub(a,1000)
...
end
subroutine sub(a,m)
                ion a(63)
                1, m
                = float(I)
end
```

```
-Wf, -pvctl noassume
-Wf, -pvctl loopcnt=1000000

!CDIR LOOPCNT
```

```
% ./a.out
 ****  96 Loop count is greater than that assumed by the compiler
: loop-count=1000 eln=13 PROG=sub ELN=14(400001584)
                Called from t ELN=5(4000008f4)
%
```


Formatted I/O – insufficient record size

Runtime Messages (FORTRAN90/SX Programmer's Guide, App. D)

M	**186 *** Formatted record insufficient in WRITE**
A	**Add to job script (ksh case):** **F_SYSLEN=1024; export F_SYSLEN** (default: 134)

Doesn't work for stderr (unit 0) –

use list-directed I/O instead : **`write(0,*)`**

CPP: Predefined Variables

```
-Uname
      Unname the previously defined reserve symbol of the
      preprocessor.  The current reserve symbols in SUPER-UX
      are "unix", "_FLOAT0", "_FLOAT1", "_FLOAT2", and "SX".

-Dname
-Dname=def
      Define name with value def as if by a #define. If no
      =def is given, name is defined with the value 1. The -D
      option has lower precedence than the -U option. That
      is, if the same name is used in both a -U option and a
      -D option, the name will be undefined regardless of the
      order of the options. Characters from supplementary
      code sets can be used in def.
```

(…not so popular traps)

CPP: Predefined Variables

```
   program cpp_test
      dimension SX(20),temp(20)
      SX=10
      do i=1,20
      temp(i)=SX(i)
      enddo
      print*, temp(10)
      end
```

```
iwrsx8vor1 55: sxf90 cpp_test.F
f90: error(106): i.cpp_test.F, line 2: Syntax error in DIMENSION statement.
f90: error(106): i.cpp_test.F, line 3: Syntax error in assignment statement.
f90: error(110): i.cpp_test.F, line 5: Extra text follows the end of statement.
f90: i.cpp_test.F, cpp_test: There are 3 errors.
sxf90 fatal : /SX/usr/lib/f90com command error : 1

iwrsx8vor1 56: sxf90 cpp_test.F -USX
```


Empowered by Innovation NEC

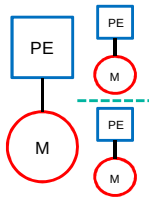
Parallel Programming Basics

Dr. Jens-Olaf Beismann
Senior Benchmarking Analyst, NEC Deutschland GmbH

Overview

- Parallelization paradigms
- OpenMP
- MPI
- “How to…”
- Performance considerations

Motivation



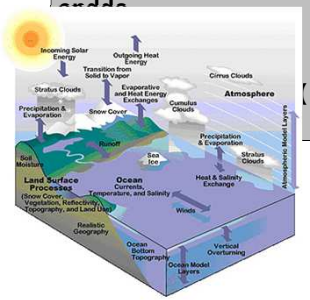

```
do i = 1, 500
     b(i) = s * a(i)
enddo

do i = 501, 1000
     b(i) = s * a(i)
enddo
```

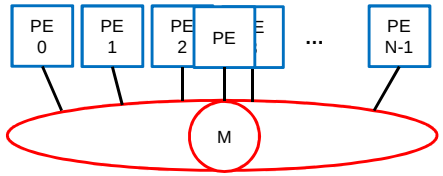

Parallelization

- Prerequisites
  - Data parallelism
  - Functional parallelism
- Hardware concepts
  - Shared memory
  - Distributed memory

```
do i = 1, 500
     b(i) = s * a(i)
enddo
```



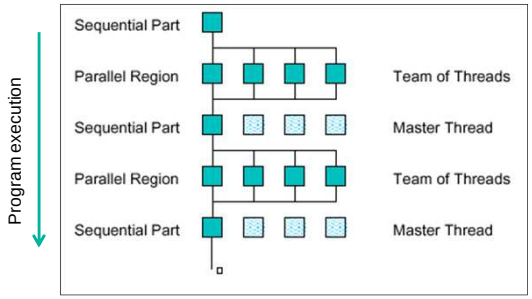
Shared memory parallelization



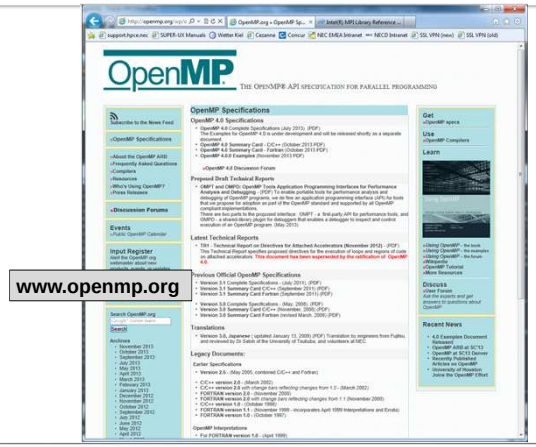
Shared memory parallelization – OpenMP

- Fork-join model of parallel execution
- Begin execution as a single process (master thread)
- Start of parallel construct:
  Master thread creates team of threads
- Completion of a parallel construct:
  Threads in the team synchronize: implicit barrier
- Only master thread continues execution

OpenMP – execution model



OpenMP – shared memory parallelization standard



OpenMP – shared memory parallelization standard

- Parallel programming model for shared-memory architectures
- Work sharing between parallel “threads”
- Based on compiler directives

- Portable
- Incremental

- Available for Fortran and C/C++
- Current standard OpenMP 4.0 (July 2013)

OpenMP – my first parallel loop

- Initiate and end parallel region
- Worksharing construct : indicate what to parallelize, and how
- Implicit barrier at the end of worksharing loop

```
!$omp parallel
!$omp do
do j = 1, 20
 do i = 1,512
   a(i,j) = float(i) * x(j)
 end do
end do
!$omp end do
!$omp end parallel
```


OpenMP - syntax

- Directive format :
  - sentinel - directive name [- clause]

```
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(ALPHA)
```

OpenMP – worksharing constructs

- **`!$omp do`** – iterations of a loop shared by a team of threads
- **`!$omp workshare`** – share parallel operations in Fortran90 constructs (array syntax, FORALL, WHERE, …)
- **`!$omp sections`** – distribute independent code blocks between threads
- **`!$omp single`** – code executed only by one thread

OpenMP – data scoping

```
!$omp parallel default (shared), private(i,j ,x,y)
!$omp do
do j = 1, 20
 do i = 1,512
   x=sqrt(b(j))
   y=sin(x)
   a(i,j) = float(i) * (x+y)
 end do
end do
!$omp do
!$omp end parallel
```

Shared by all threads !

OpenMP – data scoping

- Default rules :
  - Variables are shared between all threads in a parallel region
  - Loop index variables are private !
  - Local variables in subroutines called from parallel regions are private
- Privatize variables by **`PRIVATE`** clause
  - Check for assignments to scalar variables in parallel regions

OpenMP

```
subroutine sum_a( a, sum )
    implicit none
    integer         :: i
    real, dimension(:)     :: a
    real                   :: sum

    sum = 0.0

!$omp parallel do default(shared), private(i)
    do i = 1, size(a)
!$omp critical
       sum = sum + a(i)
!$omp end critical
    end do
!$omp end parallel do

return
end subroutine sum_a
```

OpenMP – reductions

```
subroutine sum_a( a, sum )
    implicit none
    integer         :: i
    real, dimension(:)     :: a
    real                   :: sum

    sum = 0.0

!$omp parallel do default(shared), private(i) &
!$omp             reduction(+:sum)
    do i = 1, size(a)
       sum = sum + a(i)
    end do
!$omp end parallel do

return
end subroutine sum_a
```

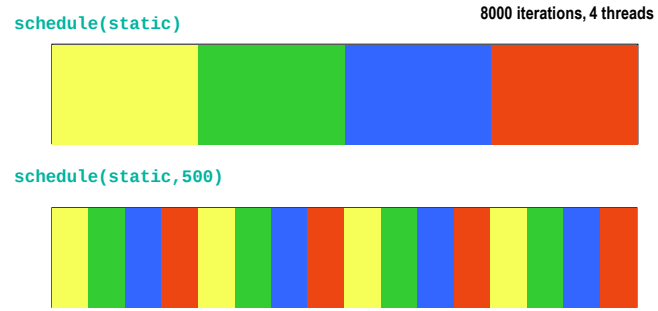

OpenMP – number of threads

- Number of parallel threads determined at runtime
- Environment variable
  - **`OMP_NUM_THREADS = <n>`**
- Sets number of threads once for all parallel constructs
- Library routine
  - **`call omp_set_num_threads(<m>)`**
- Modifies number of threads for subsequent parallel regions
- Default – implementation dependent

OpenMP – scheduling

- Format: **`schedule`**(*kind[,chunksize]*)
- Purpose: specification of work-sharing method *kind*, optionally modified with *chunksize* parameter
- Parameter *kind*:
  - **`static`** iterations divided statically: execution of *chunksize* iterations by the threads in round-robin fashion
  - **`dynamic`** iterations divided dynamically: execution of *chunksize* iterations by the first thread finishing its previous work
  - **`guided`** as dynamic, but starting with large chunks, decreasing chunk size approx. exponentially until of size *chunksize*
  - **`runtime`** choice deferred until runtime, using environment variable OMP_SCHEDULE

OpenMP – STATIC scheduling



OpenMP – DYNAMIC scheduling

8000 iterations, 4 threads

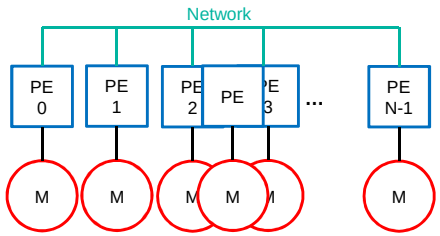
```
schedule(dynamic,500)
```

OpenMP – GUIDED scheduling

8000 iterations, 4 threads

```
schedule(guided,50)
```

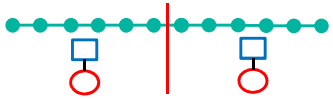
Distributed memory parallelization



Distributed memory parallelization – domain decomposition

```
do i = 2, 500
     b(i) = s * (a(i+1) + a(i-1))
enddo

do i = 501, 999
     b(i) = s * (a(i+1) + a(i-1))
enddo
```

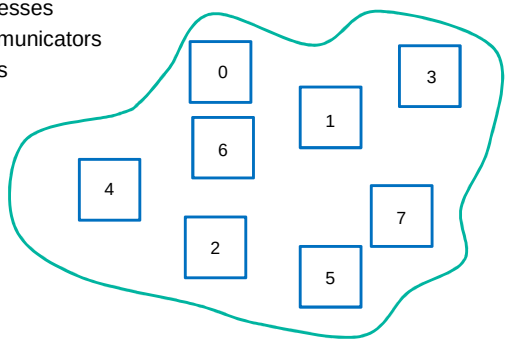


Distributed memory parallelization – MPI

- Domain decomposition – data exchange (“message passing”) between PEs
- **M**essage **P**assing **I**nterface – standardization of communication procedures
  - PEs run the same (SPMD) or different programs (MPMD)
  - Programs are written in sequential languages (Fortran, C, …)
  - All variables are local to a PE
- Current standard MPI 3.0
- Supported by several free and commercial implementations
- Vendor-optimized implementations (Intel MPI, MPI/SX, …)

Distributed memory parallelization – MPI

- processes
- communicators
- ranks



Distributed memory parallelization – MPI

- parallel MPI **processes**
- **communicators** – groups of MPI processes
  - **MPI_COMM_WORLD**
  - user-defined communicators
- within a communicator : processes identified by their **rank**

Distributed memory parallelization – MPI

```
program simple
include 'mpif.h'
integer numtasks, rank, ierr, rc

call MPI_INIT(ierr)
if (ierr .ne. MPI_SUCCESS) then
 print *,'Error starting MPI program. Terminating.'
 call MPI_ABORT(MPI_COMM_WORLD, rc, ierr)
end if

call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
print *, 'Number of tasks=',numtasks,' My rank=',rank

call time_stepping

call MPI_FINALIZE(ierr)

end program
```

Distributed memory parallelization – MPI

- Messages :
  - Sending process
  - Data source
  - Data type
  - Amount of data
  - Receiving process
  - Data destination

```
call MPI_Send(sbuf, count, type, dest, tag, comm,          ierror)
.....
call MPI_Recv(rbuf, count, type, source, tag, comm, status, ierror)
```


Distributed memory parallelization – MPI

- Point-to-point communication
  - one sender, one receiver
  - blocking, synchronous
- Non-blocking point-to-point communication
  - Initiate message transfer and return to user code
  - Check later for successful transfer
  - Communication and computation can be overlapped

```
call MPI_Irecv(rbuf, count, type, source, tag, comm, request, ierror)
… <useful stuff>
call MPI_Wait(request, status, ierror)
.....
…
call MPI_Send(sbuf, count, type, dest, tag, comm, ierror)
```


Distributed memory parallelization – MPI

- Collective communication
  - Barriers
  - Broadcasts
  - Reductions

```
call MPI_Barrier(comm, ierror)
call MPI_Bcast(sbuf, count, type, source, comm, ierror)
call MPI_Reduce(sbuf, rbuf, count, type, OP, dest, comm, ierror)
! OP = MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, …
```


Parallel programs – compilation and execution

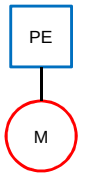
- OpenMP
- compile and link with appropriate option
  - **`-openmp, -Popenmp, -qsmp=omp, …`**
- define number of parallel threads
  - **`export OMP_NUM_THREADS=<n>`**
- start executable

Parallel programs – compilation and execution

- MPI
- use compiler wrapper scripts
  - **mpif90, sxmpif90, mpiifort, …**
- define number of MPI processes
- if necessary: list execution hosts
- start executable
  - **mpirun -f $NQSII_MPINODES -r ssh -perhost $PPN \ -np $PROC mympi.x**
  - **mpirun –nn $NODES –nnp $PPN mympi.x**

Performance considerations

- Data access
- Storage order
  - Fortran : column-major




```
…
real, dimension(3, 3) :: x
…
```

x(1,1)	x(2,1)	x(3,1)	x(1,2)	x(2,2)	x(3,2)	x(1,3)	x(2,3)	x(3,3)

```
do j = 1, 3
  do i = 1, 3
    x(i, j) = x(i, j) + …
  enddo
enddo
```

```
do i = 1, 3
  do j = 1, 3
    x(i, j) = x(i, j) + …
  enddo
enddo
```


Performance considerations – OpenMP

- Use less parallel regions
- Avoid implicit barriers
- Avoid dynamic/guided scheduling

```
!$omp parallel
!$omp do
do j = 1, 20
  do i = 1,512
    a(i) = float(j) * x(j)
  end do
end do
!$omp end do nowait
…
!more parallel loops
!$omp do
…
!$omp end parallel
```


Performance considerations – MPI

- Communication overhead
  - transfer time = latency + message length/bandwidth
- N messages
  - transfer time = N*latency + total message length/bandwidth
  - reduce the number of messages
  - reduce the total amount of bytes
  - bandwidth depends on protocol

Parallelization – performance considerations

- Minimize serial fraction
- Scaling
- Load balance
- Efficiency
- Optimal domain decomposition
- Pinning

$T_s = s + p \qquad (= 1\ !)$
$T_p = s + p/N = s + (1-s)/N$

$S = T_s/T_p = 1 / [s + (1-s)/N]$
$\rightarrow 1/s\ !$

Parallel programming – parts 2, 3, …

- Hybrid parallelization – MPI/OpenMP
- Fortran Coarrays (Fortran2008)

```
real, dimension(100), codimension[*] :: x, y
…
x(:) = y(:)[n]
```

- …