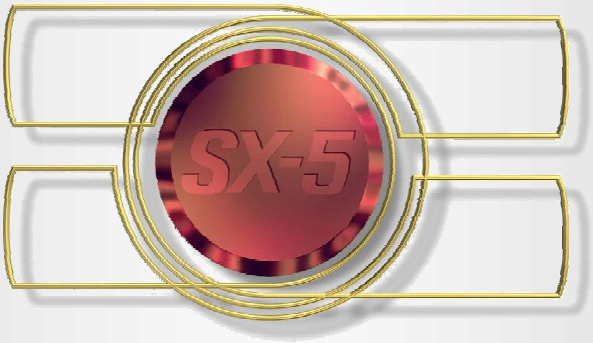
Shared Memory Parallelization on NEC supercomputers



Dr. E. Focht / H. Berger / Dr. A. Findling
NEC EHPCTC, Stuttgart

Overview

- OpenMP on SX
- Multitasking Intro
- Mikrotasking
  - Microtasking: Step by Step
  - Compile
  - Reserve/Release
  - Directives
  - Autotasking
- Makrotasking

Levels of Parallelism

1. data parallelism
2. parallel operations
3. parallel outer loops in loop nests
4. parallel subroutines
5. parallel processes (of different granularity)

granularity

Parallelization on NEC SX Systems

- Shared Memory
  - OpenMP
  - Microtasking
  - Macrotasking
- Distributed Memory
  - MPI
  - HPF
  - MPI with micro/macrotasking

OpenMP

- Available on SX for F90 only
- C/C++ is expected beginning of next year
- OpenMP 2.0 for Fortran is expected end of this year
- On HWW: Only on SX-5

OpenMP usage

- F90: `f90 -Popenmp`
- be carefull: `OMP_DYNAMIC` is FALSE!
- You can not use automatic parallelization and OpenMP and the same time
- But: You can mix files using both methods
  Just link with -Popenmp
- NEC's Parallelizatin directives are ignored
- Always specify `OMP_NUM_THREADS`
  Default is 16!!!

Microtasking

- parallelization on a bottom-up basis
- SPMD
- parallelism of:
  - loop iterations (PARDO)
  - statement groups (PARCASE)
  - subroutine calls
- task creation and synchronization is implicitly done by using compiler directives in the code
- work assignement is done automatically
- parallelization effort: medium to small

Microtasking: Step by Step

- Recompile whole source
  - **FORTRAN:** `f90 -P multi -Wf"-pvctl res=whole"`
  - **C:** `cc -hmulti`
- Reserve logical tasks
  - **Directives:** !CDIR RESERVE, !CDIR RELEASE
  - **Environment Variables:** F_RSVTASK and others
- Determine junks of code that should run in parallel
  - **SX-5 needs big junks of work before parallelization is efficient**
  - **Insert directives:** !ODIR CONCUR, !ODIR INNER, !PDIR PARDO

Microtasking: Task Creation

`*PDIR RESERVE[=N]`

- creates N logical tasks, if N is omitted, the number of tasks created is:
  - specified by the `-reserve` compiler option, or
  - given in `$F_RSVTASK` (environment variable), or
  - MAXCPU in the RSG or no. of physical processors
- similar to PTFORK

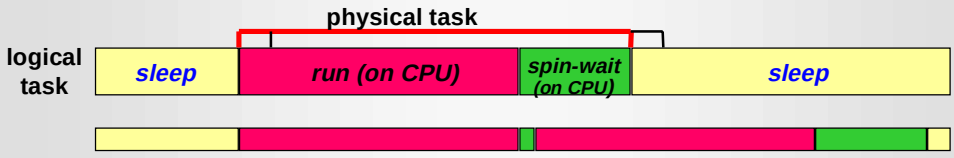
`*PDIR RELEASE`

- requests release of the logical tasks (of the microtasking group)
- similar to PTJOIN
- synchronization at RELEASE

NEC EHPCTC Stuttgart 9

Microtasking: Task Creation (2)

- RESERVE/RELEASE ☞ overhead
- if RESERVE/RELEASE are omitted, logical tasks are created and released at the beginning and the end of each parallel section
- try RESERVE/RELEASE only once, in the MAIN program
- only subroutines containing PARDO or PARCASE directives are executed parallely
- scheduling:



NEC EHPCTC Stuttgart 10

Typical situations

- original

```
SUBROUTINE ...
...
DO J=1,M
   DO I=1,N
      A(I,J)=0.0
   ENDDO
ENDDO
...
RETURN
END
```

- Case 1: Only little code before and after J-loop
- Case 2: Lot of code before and after J-loop
- Concept: Only outer loops are parallelized

Typical situations

- original

```
SUBROUTINE ...
...
DO J=1,M
   DO I=1,N
      A(I,J)=0.0
   ENDDO
ENDDO
...
RETURN
END
```

- Case 1: Only little code before and after J-loop
- Insert directive: !pdir

```
!PDIR PARDO
DO J=1,M
   DO I=1,N
      A(I,J)=0.0
   ENDDO
ENDDO
```

- DANGER: Whole subroutine is executed in parallel

Microtasking: PARDO FOR

***PDIR PARDO FOR[=N] (#pragma pdir parloop)**

- iterations of following loop are executed in parallel, each task will be assigned 1/N of the total number of iterations

```
*pdir pardo for=4
      do i=1,16
         a(i)=a(i)+b*c(i)
      enddo
```

task	1	2	3	4
	1 2 3 4	5 6 7 8	9 10 11 12	13 14 15 16

NEC EHPCTC Stuttgart 13

Microtasking: PARDO BY

***PDIR PARDO BY[=N]**

- iterations of following loop are executed in parallel, each task will be assigned N iterations, until all iterations have been completed
- default: BY=1
- interaction with scheduler at start and end of each work chunk

```
*pdir reserve=4
...
*pdir pardo by=2
      do i=1,16
         a(i)=a(i)+b*c(i)
      enddo
```

task	1	2	3	4
	1 2	3 4	5 6	7 8
	9 10	11 12	13 14	15 16

NEC EHPCTC Stuttgart 14

Microtasking: PARDO barrier

***PDIR PARDO FOR, NOBARR=(ENTRY,EXIT)**

- switch off barrier at start or end of the DO loop
- faster (e.g. for successive loops)
- be careful!

```
      subroutine sub(a,b,c,N)
      real a(N),b,c(N)
      ...
*pdir pardo for, nobarr=(entry,exit)
      do i=1,N
         a(i)=a(i)+b*c(i)
      enddo
      ...
      return
```

synchronization on entry to subroutine

synchronization on exit from subroutine

Microtasking: Critical Section

- code block between `CRITICAL` and `END CRITICAL` is executed only by one task at a time
- example: reduction (s = s + a(:))

```
      subroutine parsum(a,n,sum)
      real a(n)
      sum=0.0
      sumlocal = 0.0
*pdir pardo for, nobarr=(exit)
      do i = 1, n
        sumlocal = sumlocal + a(i)
      end do
*pdir critical
      sum = sum + sumlocal
*pdir end critical
      return
      end
```

Microtasking: Serial Section

- code block between SERIAL and END SERIAL is executed only by one task
- other tasks are waiting until the serial block execution is completed
- example: reduction (s = s + a(:))

```
      subroutine parsum(a,n,sum)
      real a(n)
*pdir serial
      sum=0.0
*pdir end serial
      sumlocal = 0.0
*pdir pardo for, nobarr=(entry,exit)
      do i = 1, n
        sumlocal = sumlocal + a(i)
      end do
*pdir critical
      sum = sum + sumlocal
*pdir end critical
      return
      end
```


General Remark: Reduction

- Parallel Summation is Random: Result not Reproducable
- Work around: Serial summation of task sums.

```
      sumlocal = 0.0
*pdir pardo by=1, nobarr=(entry,exit)
      do it=1,nt
      ibg= ...
      ied= ...
      do i = ibg, ied
        sumlocal = sumlocal + a(i)
      end do
      sumtask(it)=sumlocal
      enddo
*pdir serial
      do it=1,nt
        sum = sum + sumtask(it)
      enddo
*pdir end serial
      return
      end
```

Microtasking: Other Directives

`*PDIR BARRIER` : specify a barrier

`*PDIR WAIT[=N]/POST[=N]` : synchronization

`*PDIR EXIT` : terminate parallel processing

`*PDIR SAVE [var,...]` : allocate data to the static area

`*PDIR PARTFLUSH` : partial cache flush (not .stack section)

`*PDIR PRIVATE (var,...)` : specified variables are replaced by local work variables. The values of the original variables are not taken over by the work variables. The values of the work variables are not taken over by the original variables after the PARDO or PARCASE section.

`*PDIR SUPPOSE[=(RESERVERD,NOCONCALL,NONESTCALL)]` : skip some tests when entering a microtask procedure

- further reading: *Fortran90/SX Multitasking User's Guide*

Typical situations

- original

```
SUBROUTINE ...
...
DO J=1,M
   DO I=1,N
      A(I,J)=0.0
   ENDDO
ENDDO
...
RETURN
END
```

- Case 2: Lot of code before and after J-loop
- Insert directive: !odir concur
- Parallel regions are placed automatically into new subroutines
- Directives are discribed in the manual as Automatic parallelization Autotasking

Autotasking: concept

- Pacific Sierra preprocessor ‘fopp’, (‘copp’ for C) is used
  - also used by other vendors!
- parallel regions are placed into new subroutines automatically inserts *pdir
  - for loop-nests
  - for inner loops (needs directive or switch)
  - apart from that some optimizations are performed as well
  - can be used as a learning tool for microtasking
- can handle sequential and critical regions automatically
- subroutines already containing *pdir directives are skiped!

Autotasking: concept (2)

- Data scoping (automatically!!!):
  - shared variables are passed as arguments to new subroutines
  - private variables are declared locally within the new routines
- Transformed code
  - use compiler directive
    f90 -Pmulti -Wf“-L transform“
    to see transformed code

Autotasking Example: Threshold Test

- original

```
!CDIR CONCUR
DO J=1,M
   DO I=1,N
      A(I,J)=0.0
   ENDDO
ENDDO
```

- translation

```
  IF (M*N.GT.1666)THEN
     CALL T$1(M,N,A)
  ELSE
     DO J=1,M
        DO I=1,N
           A(I,J)=0.0
        ENDDO
     ENDDO
  ...
  SUBROUTINE T$1(M,N,A)
  INTEGER M,N,I,J
  REAL A(999,999)
*PDIR PARDO FOR=4
  DO J=1,M
     DO I=1,N
        A(I,J)=0.0
     ENDDO
  ENDDO
  RETURN
  END
```

NEC EHPCTC Stuttgart 23

Autotasking Example: Reduction

- original

```
!CDIR CONCUR
DO J=1,M
   DO I=1,N
      S = S +  A(I,J)
   ENDDO
ENDDO
```

- translation

```
IF (M*N.GT.1666)THEN
     CALL R$1(A,N,M,S)
  ELSE
  ...
  SUBROUTINE R$1(A,N,M,S)
  INTEGER M,N,I,J
  REAL A(M,N),S,S1
  S1=0
*PDIR PARDO FOR=4
  DO J=1,M
     DO I=1,N
        S1=S1+A(I,J)
     ENDDO
  ENDDO
*PDIR CRITICAL
  S=S+S1
*PDIR END CRITICAL
  RETURN
  END
```

NEC EHPCTC Stuttgart 24

Autotasking: Inner Loops

- original

```
!CDIR INNER
 DO I=1,N
   A(I)= SQRT(B(I)**2 C(I)**2)
 ENDDO
```

- default: parallelize outer loop

- translation

```
  IF (N.GT.294)THEN
     CALL S$1(N,B,C,A)
  ELSE
  ...
  SUBROUTINE S$1(N,B,C,A)
  INTEGER N,I
  REAL B(N),C(N),A(N)
*VDIR NODEP
*PDIR PARDO FOR=4
  DO I=1,N
     A(I)=SQRT(B(I)**2+C(I)**2)
  ENDDO
  END
```


Autotasking: Concurrent Subr. Calls

- original

```
!CDIR CNCALL
 DO I=1,N
   CALL SUB1(I,A(I))
 ENDDO
```

- Allow concurrent calls in loop
- Almost like macrotasking, but more efficient

Typical situations

- original

```
DO I=1,N
  C(I) = MYFUNC(A(I)) + ATAN(B(I))
ENDDO
```

- Case 2: Lot of code before and after J-loop
  BUT
  simple directive does not work
- Forced Parallel Directive
  !CDIR PARALLEL DO
  !CDIR FORCEPARDO
- Task specific variables or arrays have to be declared as PRIVAT
  !CDIR FORCEPRIVAT

Autotasking: fopp Directives

ODIR[F,R,L] *directive

- F: directive valid for the whole file
- R: valid for the routine
- L: valid for the next loop

- *ODIR (NO)CONCUR : switch on/off parallelization
- *ODIR SKIP : fopp skips parts of the code
- *ODIR (NO)VECTOR : switch on/off vector optimizations
- *ODIR INNER : parallelize inner loop(s)
- *ODIR (NO)THRESHOLD : switch on/off threshold code generation
- *ODIR (NO)SYNC : do(n't) ignore potential overlaps of array sections
- ...

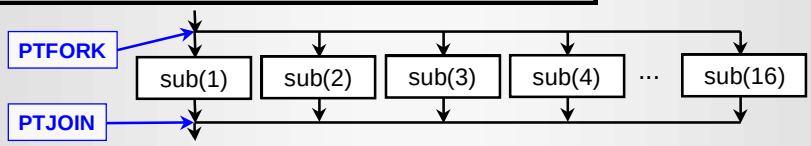
Macrotasking

- coarse grained parallelization on a top-down basis
  - subroutine
  - groups of subroutines
- task creation & synchronization performed explicitely by library calls
- FORK/JOIN, BARRIER, EVENTS, LOCKS
- compile with *-P stack* or *-P multi*
- C Programmers: use pthreads
- parallelization effort: medium to large

Macrotasking: PTFORK, PTJOIN

```
program MAIN
parameter (nproc=16)
integer tid(nproc), isave(nproc)
character*32 par(nproc)
...
do I = 2, nproc
  isave(I)=I
  call PTFORK(tid(I),par(I),sub,isave(I))
end do
call sub(1)
...
do I = 2, nproc
  call PTJOIN(tid(I))
end do
...
end
```

call by reference !



Macrotask Synchronization: LOCK

- ‘critical section’: use lock variable as semaphore

```
subroutine SUMUP( nstart, nend, sum, a )
common / locks / ilock
real a(*)
sumlocal = 0.0
do I = nstart, nend
   sumlocal = sumlocal + a(I)
enddo
call PLLOCK( ilock )
sum = sum + sumlocal
call PLUNLOCK( ilock )
return
end
```

- additional calls outside the parallel region needed:
  - `call PLASGN(ilock)` : assign a lock to variable `ilock`
  - `call PLFREE(ilock)` : release the lock variable `ilock`
- check lock status: `IPLSTAT(ilock)`, `IPLSTATL(ilock)`

Macrotask Synchronization: BARRIER

- synchronous barrier control

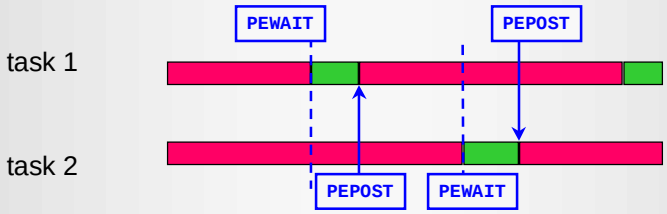
```
program MAIN
common /barr/ ibarr
...
call PBASGN(ibarr,NCPU)
do i=2,NCPU
   call PTFORK(tid(I),par(I),sub)
enddo
...
call PBSYNC(ibarr)
...
call PBFREE(ibarr)
...
```

```
subroutine SUB
common /barr/ ibarr
...
call PBSYNC(ibarr)
return
end
```

- related function call:
  - `IPBSTAT(ibarr)` : check status of barrier `ibarr`

Macrotask Signalling: EVENTS

- subroutine & function calls:
  - `call PEASGN(ievnt)` : assign an event
  - `call PEWAIT(ievnt)` : wait for an event
  - `call PEPOST(ievnt)` : post an event
  - `call PECLEAR(ievnt)`: clear an event
  - `call PEFREE(ievnt)` : release an event
  - `I=IPESTAT(ievnt)` : check the status of an event



Other Macrotasking Intrinsics

- subroutine & function calls:
  - `call PTPARAM(tpar)` : fetches task parameters
  - `I = IPTSTAT(tvar)` : checks subtask status
  - `I = IPTID()` : returns task identification number
  - `I = IPSMAX()` : returns max. number of physical tasks
  - `I = IPTNAP()` : returns no. of CPUs
  - `call PTLIST([unit])`: outputs a list showing task states
  - `call PSTUNE(key,val,...)` : change tuning parameters of task scheduler
- cf. *Fortran90/SX Multitasking User's Guide*

Macrotasking: Remarks

- increase the size of a task as much as possible
- reduce synchronization and exclusive control (locks) to improve the performance
- be careful about data scoping:
  - private / shared data
  - use locks when accessing shared data
- preferably use SPMD
  - load balancing
  - simplicity
  - replace EVENTs by BARRIERs

Multitasking Memory Management

- Task shared data:
  - COMMON or GLOBAL COMMON data
  - data specified in SAVE statement
  - actual arguments of PTFORK and microtask subroutines (microtask shared)
  - initialized data
  - data declared in LOCAL COMMON statements (microtask shared)
- Task private data
  - LOCAL COMMON data (macrotask private, cf. compiler options)
  - allocated on stack
  - stack size is estimated by the linker for each task
  - stack space is dynamically allocated at runtime for each task

Questions?