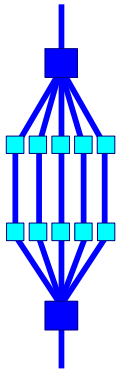
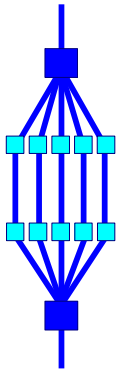


OMPlab on Sun Systems

Ruud van der Pas
Nawal Copty
Yuan Lin
Eric Duncan

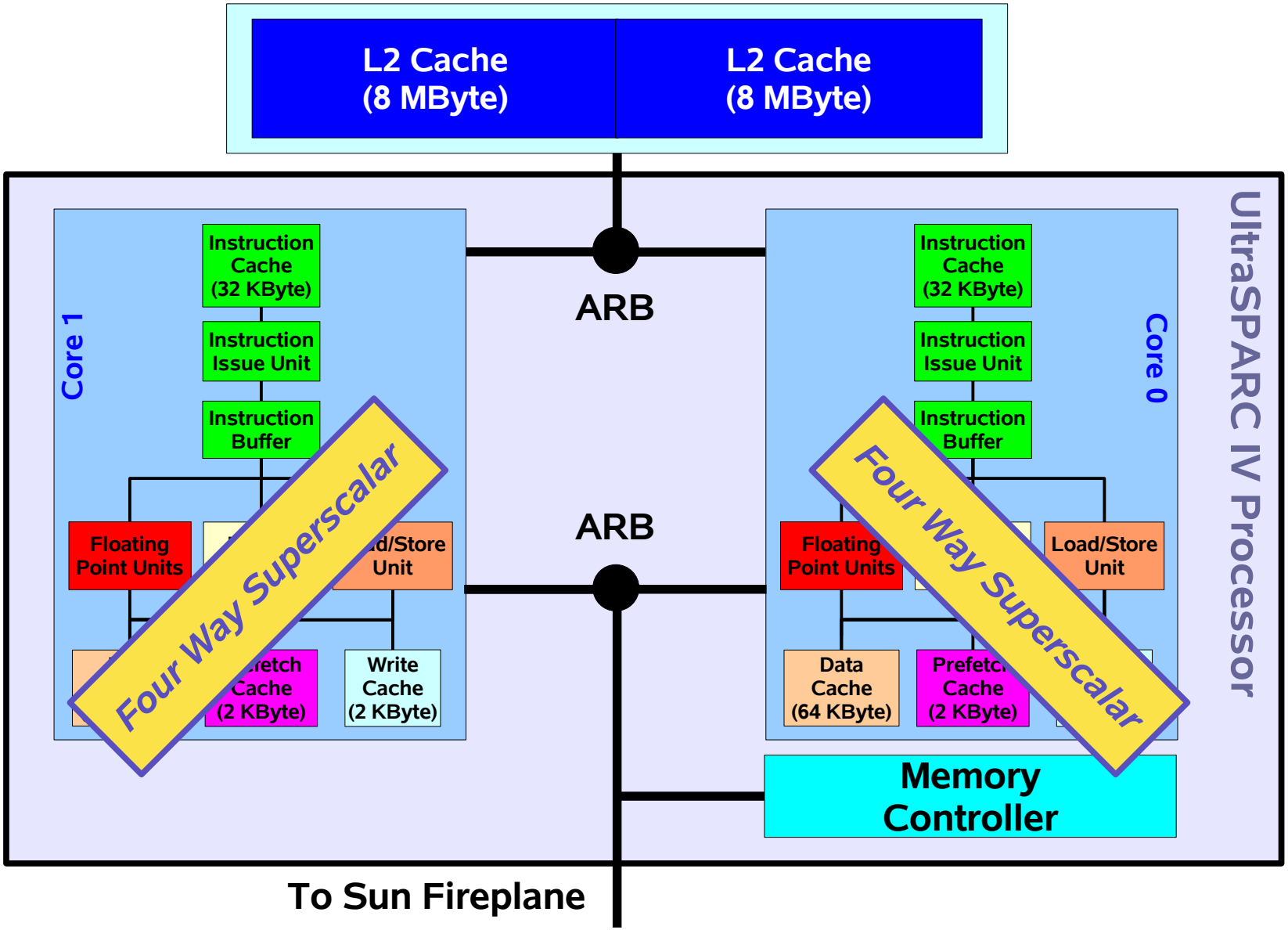
Scalable Systems Group
Sun Microsystems

IWOMP 2005
University of Oregon
Eugene, Oregon, USA
June 1-4, 2005

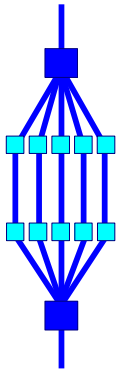


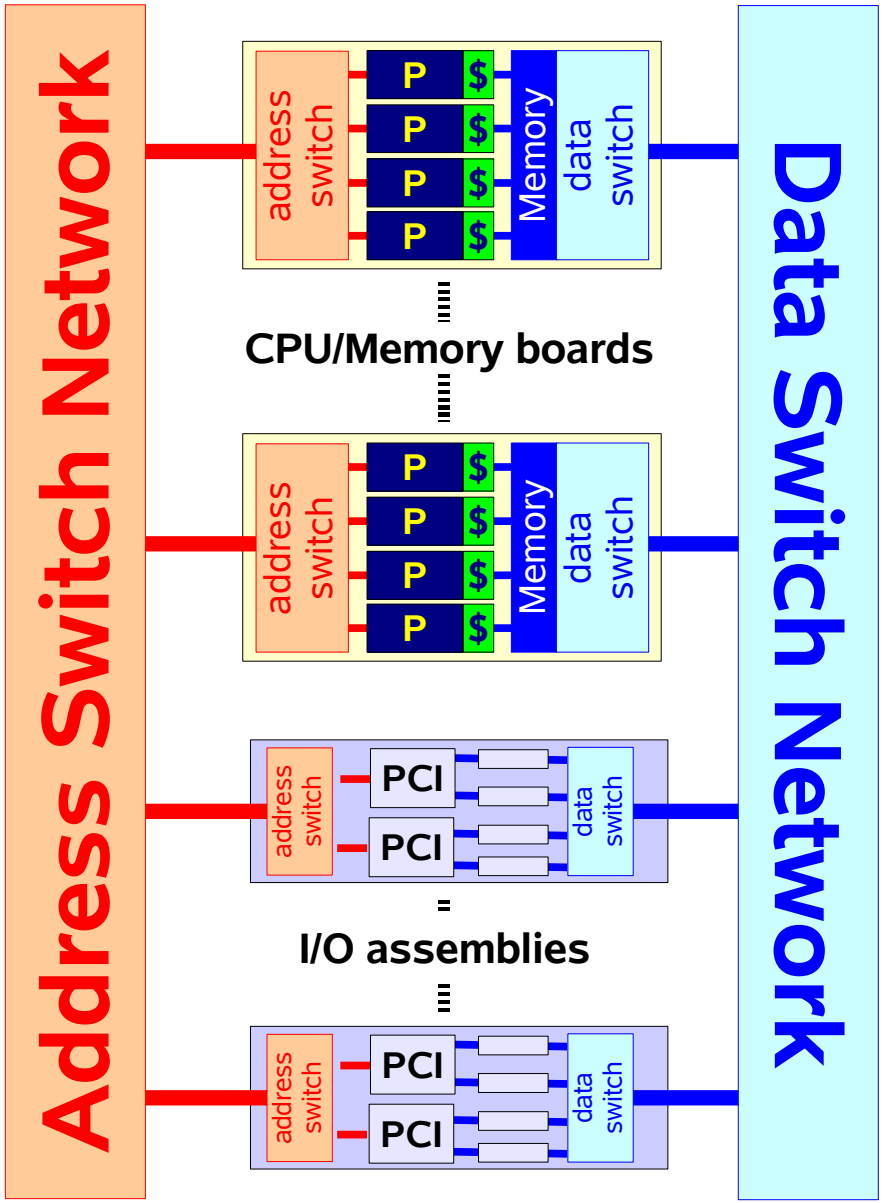
Hardware

US IV - Block diagram



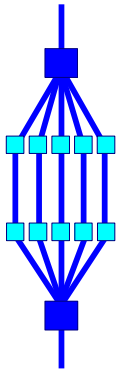
The simplified big picture

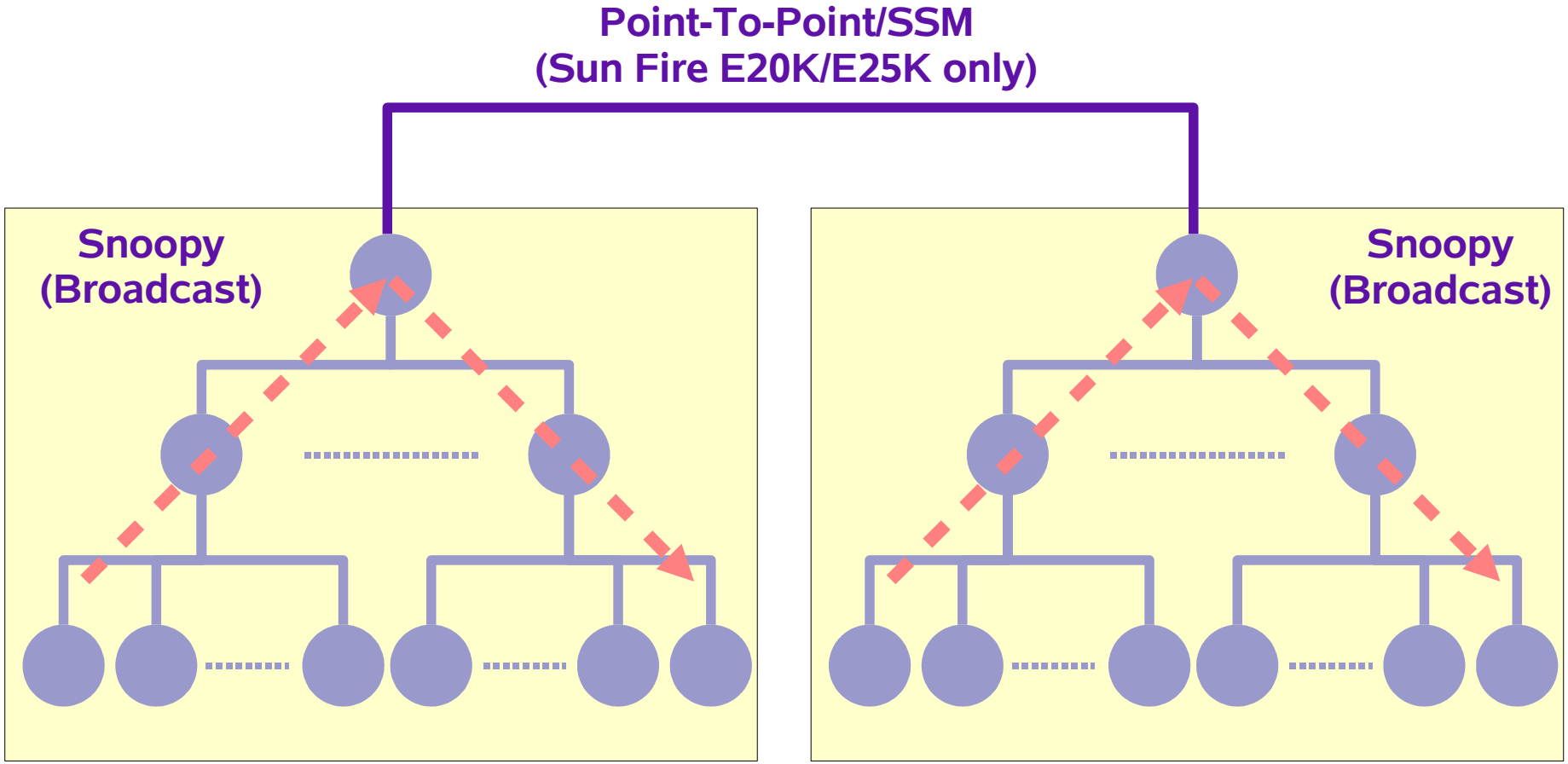




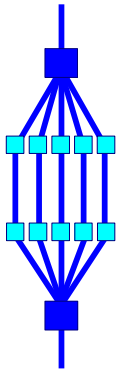
- ✔ *The SMP model is preserved throughout the product line*
- ✔ *Architectural details of the switch network depend on the Sun Fire model*
- ✔ *A hierarchical tree is used to build the interconnect*
- ✔ *Smaller systems, have less switch layers*
- ✔ *Largest system, the Sun Fire E25K, can have up to 104 US III (Cu) processors or 72 US IV processors (144 cores)*

A hierarchical coherency tree



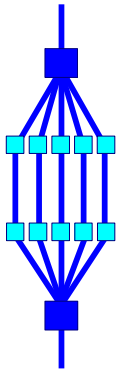


IWOMP 2005 Sun lab system

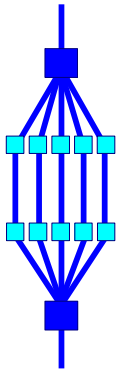


- *Sun Fire SMP - Kindly made available by the Technical University of Denmark (DTU) in Lyngby, Denmark*
- *Twelve UltraSPARC IV processors*
  - *Two cores/processor*
    - ✔ *Each core runs @ 1350 MHz*
  - *Total: 24 cores*
  - *Memory: 48 GByte*
- *Software Environment*
  - *Solaris 9*
  - *Sun Studio 10 compilers*
- *Directions how to log into the system will be handed out separately*

How to access the DTU system

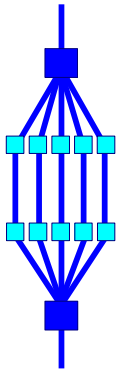


- Use the ThinLinc client on the PC/workstation/laptop
  - Download from http://www.thinlinc.com if not installed yet
- Start the ThinLinc client:
  - Server name: thinlinc.hpc.dtu.dk
  - User name and password will be provided
- Select your favourite Window Manager
  - Recommend: IceWM (very lightweight WM)
- Open a terminal (task bar or right mouse button)
- In this terminal window, type: $ run_on isaac xterm -ls
  - Full name is isaac.hpc.dtu.dk
- Do not change your PATH and other settings!



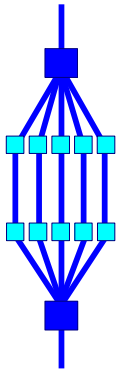
The Sun Studio Compilers

Sun Studio 10 - Some features



- *Hardware support:*
  - *SPARC, AMD and Intel*
- *Compilers:*
  - *Fortran (f95), C (cc) and C++ (CC)*
- *Parallelization*
  - *Automatic parallelization (-xautopar option)*
  - *OpenMP V2.0 (-xopenmp option)*
  - *Combination of the above (use both options)*
- *Compiler Commentary*
  - *Add -g to compile options*
  - *Use the er_src command to analyze object file*

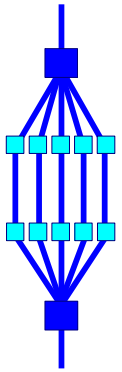
Serial compiler options



In general, one obtains very good performance out of the Sun compilers by just using these 2 options on the compile and link line:

```
-fast -xarch=v8plusb    (32-bit addressing)
-fast -xarch=v9b        (64-bit addressing)
```

Additional serial options to explore

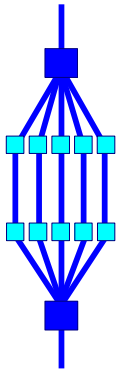


Option	Description	f95	cc	CC	Comp	Link
-xinline	Controls inlining	av.	av.	av.	+	-
-xipo	Interprocedural analysis	av.	av.	av.	+	+
-xprofile	Profile feedback	av.	av.	av.	+	+
-xprefetch	Prefetch on/off	av.	av.	av.	+	-
-xprefetch_level	Controls prefetch algorithm	av.	av.	av.	+	-
-xprefetch_auto_type	Prefetch for indirect addressing	av.	av.	av.	+	-
-stackvar	Local data on stack	av.	n.a.	n.a.	+	-
-xvector	Vectorization of intrinsics	av.	av.	av.	+	+
-xalias	Aliasing of variables	av.	n.a.	n.a.	+	-
-xalias_level	Aliasing of data types	n.a.	av.	av.	+	-
-xsfpconst	Unsuffixed fp consts are single	n.a.	av.	n.a.	+	-
-xrestrict	Restricted pointers (or not)	n.a.	av.	av.	+	-

= option is available, but may not be implied, or try out non-default settings

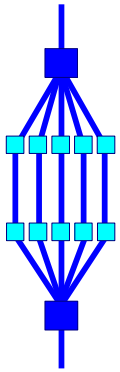
= option is not available or applicable

Note: -vector is implied with -fast on SPARC, but not on AMD



Automatic Parallelization

Loop based parallelization



- **Loop based parallelization:**
  - *Different iterations of the loop are executed in parallel*
- **Same binary can be run using any number of threads**

```
for (i=0; i<n; i++)
  a[i] = b[i] + c[i];
```

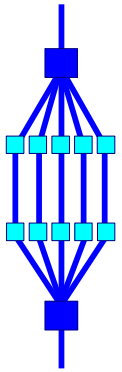
Thread 0

```
for (i=0; i<n/2; i++)
  a[i] = b[i] + c[i];
```

```
for (i=n/2; i<n; i++)
  a[i] = b[i] + c[i];
```

Thread 1

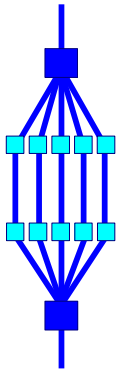
Automatic parallelization options



Option	Description
-xautopar	Automatic parallelization (Fortran, C and C++ compiler) Requires -xO3 or higher (-xautopar implies -xdepend)
-xreduction	Parallelize reduction operations Recommended to use -fsimple=2 as well
-xloopinfo	Show parallelization messages on screen

Use environment variable OMP_NUM_THREADS to set the number of threads (default value is 1)

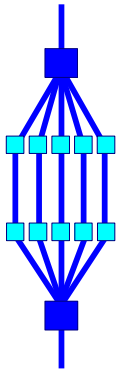
Loop versioning example



```
% cc -c -g -fast -xrestrict -xautopar -xloopinfo sub1.c
```

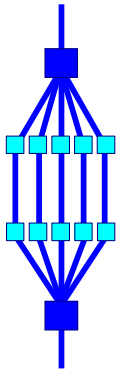
```
     1 void sub1(int n, double a, double *x, double *y)
     2 {
     3    int i;
     4    for (i=0; i<n; i++)
     5      x[i] += a*y[i];
     6 }

"sub1c.c", line 4: PARALLELIZED, and serial version generated
```

- *The compiler will generate two versions, unless the loop has constant bounds or if the compiler can derive the loop lengths from the context*
- *The serial version will be executed if there is not enough work to be done in the loop*



OpenMP on Sun systems

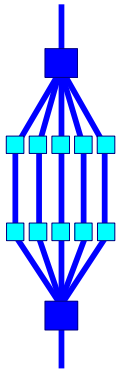
OpenMP compiler options



Option	Description
-xopenmp	Equivalent to -xopenmp=parallel
-xopenmp=parallel	Enables recognition of OpenMP pragmas Requires at least optimization level -xO3
-xopenmp=noopt	Enables recognition of OpenMP pragmas The program is parallelized accordingly, but no optimization is done *
-xopenmp=none	Disables recognition of OpenMP pragmas (default)

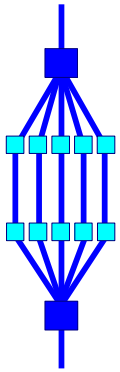
**) The compiler does not raise the optimization level if it is lower than -xO3*

Related compiler options



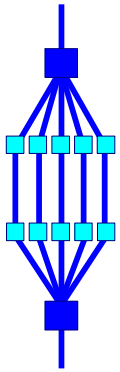
Option	Description
-xloopinfo	Display parallelization messages on screen
-stackvar	Allocate local data on the stack (Fortran only) Use this when calling functions in parallel Included with -xopenmp=parallel | noopt
-vpara/-xvpara	Reports OpenMP scoping errors in case of incorrect parallelization (Fortran and C compiler only) Also reports OpenMP scoping errors and race conditions statically detected by the compiler
-XlistMP	Reports warnings about possible errors in OpenMP parallelization (Fortran only)

Compiler commentary



```
% cc -c -g -fast -xopenmp mxv.c
% er_src -cc parallel mxv.o

   Private variables in OpenMP construct below: j, i
   Shared variables in OpenMP construct below: c, a, b
   Firstprivate variables in OpenMP construct below:  n, m

     6. #pragma omp parallel for default(none) \
     7.         private(i,j) firstprivate(m,n) shared(a,b,c)

   Loop below parallelized by explicit user directive
     8.    for (i=0; i<m; i++)
     9.    {
    10.      a[i] = 0.0;
    11.      for (j=0; j<n; j++)
    12.        a[i] += b[i*n+j]*c[j];
    13.    }
    14. }
```

OpenMP environment variables



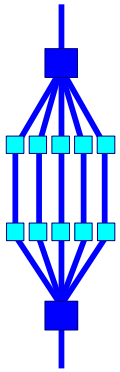
OpenMP environment variable	Default for Sun OpenMP
OMP_NUM_THREADS n	1
OMP_SCHEDULE "schedule,[chunk]"	static, "N/P" (1)
OMP_DYNAMIC { TRUE | FALSE }	TRUE (2)
OMP_NESTED { TRUE | FALSE }	FALSE (3)

(1) The chunk size approximately equals the number of iterations (N) divided by the number of threads (P)

(2) The number of threads will be limited to the number of on-line processors in the system. This can be changed by setting OMP_DYNAMIC to FALSE.

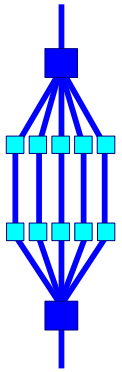
(3) Multi-threaded execution of inner parallel regions in nested parallel regions is supported as of Sun Studio 10

Note: The names are in uppercase, the values are case insensitive



Sun-specific OpenMP Environment Variables

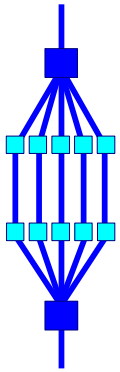
Run-time warnings



SUNW_MP_WARN TRUE | FALSE	*Control printing of warnings*

- *The OpenMP run-time library will not print warning messages by default*
- *Strongly recommended to set this environment variable to TRUE to activate the warnings*
- *This will help you diagnose run-time problems*
  - *Also reports (some) non-conforming program errors*
- *Note there is a slight performance penalty associated with setting this environment variable to TRUE*
  - *Cost depends on the operation - Explicit locking will be more expensive for example*

Example SUNW_MP_WARN/1



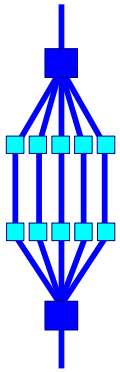
Using more threads than processors:

```
# SUNW_MP_WARN=TRUE; export SUNW_MP_WARN
# OMP_NUM_THREADS=3; export OMP_NUM_THREADS
# ./omp.exe
WARNING (libmtsk): Dynamic adjustment of threads is enabled. The
number of threads is adjusted to 2.
Thread ID 0 updates i = 0
Thread ID 0 updates i = 1
Thread ID 0 updates i = 2
Thread ID 1 updates i = 3
Thread ID 1 updates i = 4
Thread ID 1 updates i = 5
```

Now we get 3 threads →

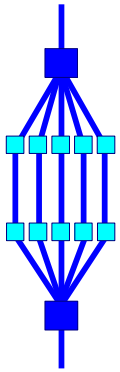
```
# OMP_DYNAMIC=FALSE; export OMP_DYNAMIC
# ./omp.exe
Thread ID 0 updates i = 0
Thread ID 0 updates i = 1
Thread ID 1 updates i = 2
Thread ID 1 updates i = 3
Thread ID 2 updates i = 4
Thread ID 2 updates i = 5
```

Example SUNW_MP_WARN/2



```
20      void foo()
21      {
22        #pragma omp barrier
23           whatever();
24      }
25
26      void bar(int n)
27      {
28        printf("In bar: n = %d\n",n);
29        #pragma omp parallel for
30        for (int i=0; i<n; i++)
31           foo();
32      }
33
34      void whatever()
35      {
36        int TID = omp_get_thread_num();
37        printf("Thread %d does do nothing\n",TID);
38      }
```

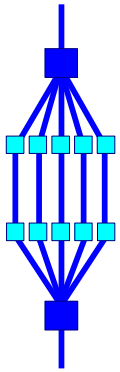
Example SUNW_MP_WARN/3

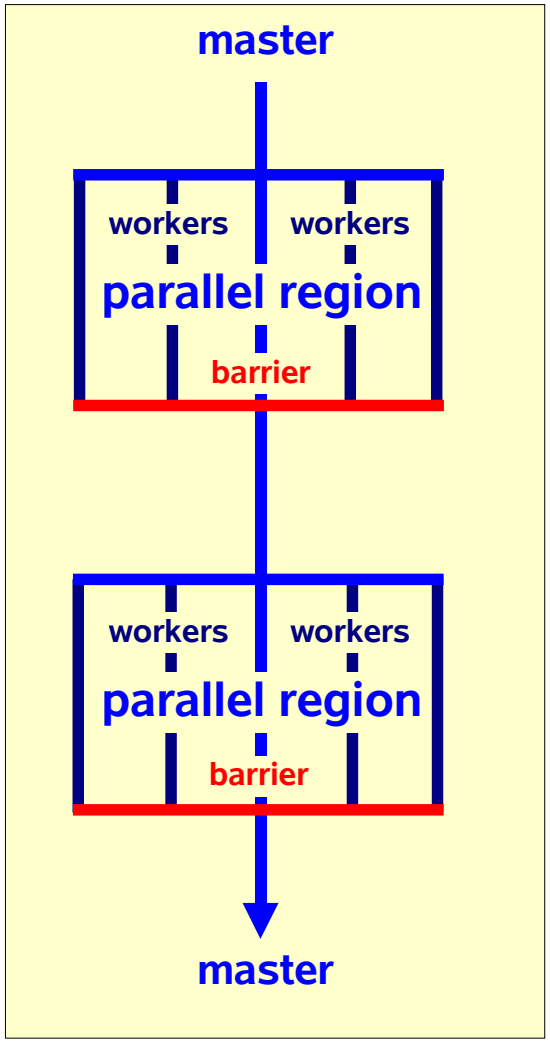


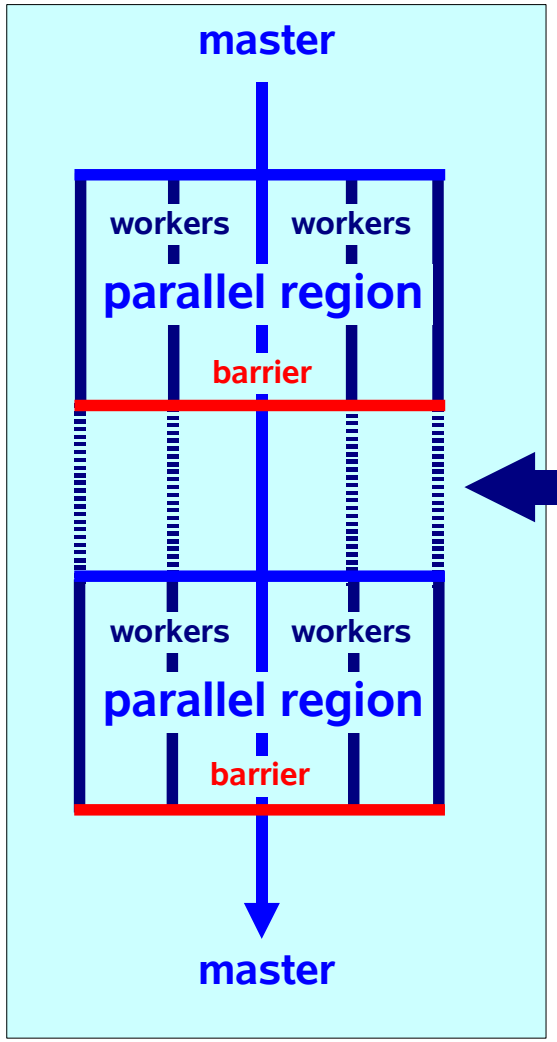
```
% cc -fast -xopenmp -xloopinfo -xvpara main.c
"main.c", line 30: PARALLELIZED, user pragma used
% setenv OMP_NUM_THREADS 4
% setenv SUNW_MP_WARN TRUE
% ./a.out
In bar: n = 5
WARNING (libmtsk): at main.c:22. Barrier is not permitted in
dynamic extent of for / DO.
Thread 0 does do nothing
Thread 3 does do nothing
Thread 2 does do nothing
Thread 1 does do nothing
WARNING (libmtsk): Threads at barrier from different
directives.
    Thread at barrier from main.c:22.
    Thread at barrier from main.c:29.
    Possible Reasons:
    Worksharing constructs not encountered by all threads in
    the team in the same order.
    Incorrect placement of barrier directives.
Thread 0 does do nothing
```

Application hangs

The fork-join model implemented

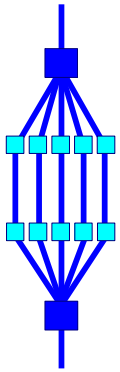






Idle threads sleep by default

The behaviour of idle threads

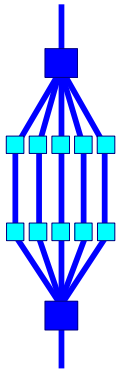


Environment variable to control the behaviour:

```
SUNW_MP_THR_IDLE
[ spin | sleep | sleep ('n's) | sleep ('n'ms) ]
```

- *Default is to have idle threads go to sleep*
- *Spin: threads will keep the CPU busy (but don't do useful work)*
- *Sleep: threads are put to sleep; awakened when new work arrives*
- *Sleep ('time'): spin for 'n' seconds (or 'n' ms), then go into sleep mode*
  - *Example:* ***setenv SUNW_MP_THR_IDLE "sleep(5 ms)"***

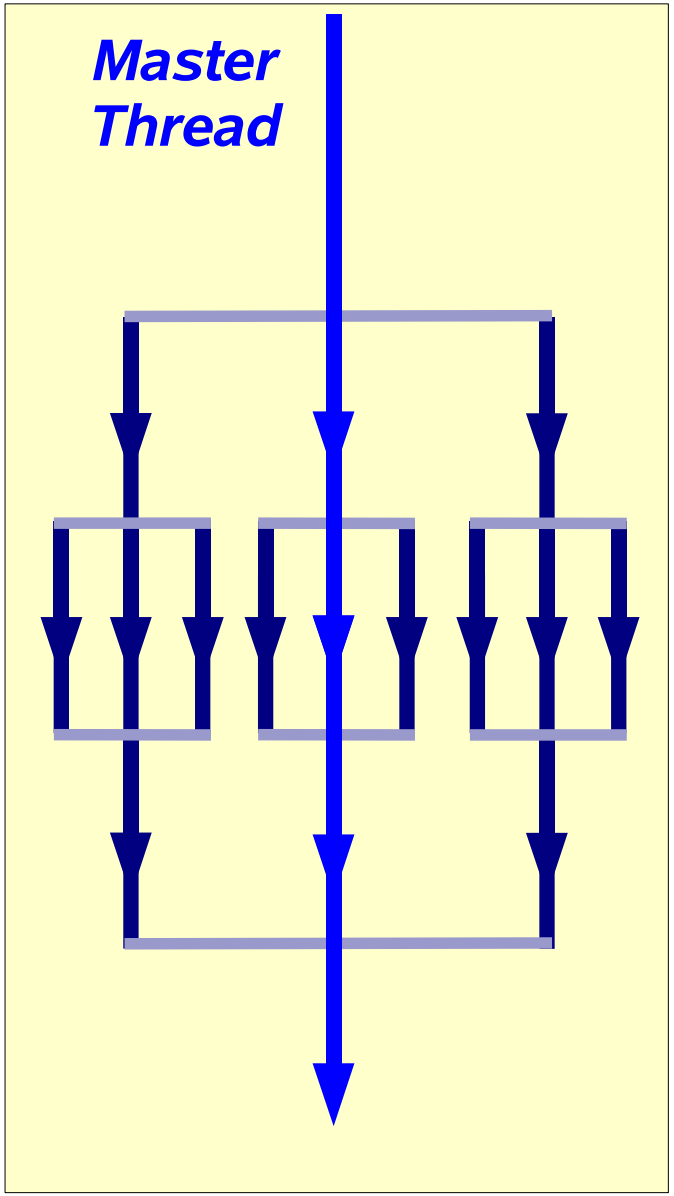
Nested parallelism



3-way parallel

9-way parallel

3-way parallel



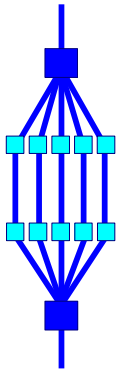
Outer parallel region

Nested parallel region

Outer parallel region

Note: nesting level can be arbitrarily deep

Nested parallelism on Sun



Control maximum number of threads for nested parallelism

```
SUNW_MP_MAX_POOL_THREADS      <n>
```

Default is 1023

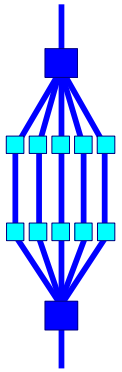
Control maximum nesting level

```
SUNW_MP_MAX_NESTED_LEVELS     <n>
```

Default is 2

Note: need to set environment variable OMP_NESTED to TRUE for this to take effect

Processor binding

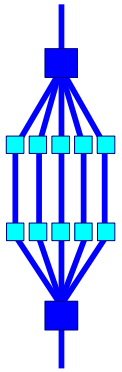


Control binding of threads to "processors"

```
SUNW_MP_PROCBIND  TRUE | FALSE
SUNW_MP_PROCBIND  Logical ID, or Range of logical IDs,
                  or list of logical IDs (separated by
                  spaces)
```

- *Processor binding, when used along with static scheduling, benefits applications that exhibit a certain data reuse pattern where data accessed by a thread in a parallel region will be in the local cache from a previous invocation of a parallel region*
- *One can use the **psrinfo** and **prtdiag** (in /usr/sbin) commands to find out how processors are configured*
- *Note that the binding is to the logical processor ID, not the physical ID (order is dictated by output of **psrinfo**)*
- *In case of syntax error, an error message is emitted and execution of the program is terminated.*

Configuration information



```
0   <=    0   on-line   since 10/30/2004 13:43:44
1   <=    1   on-line   since 10/30/2004 13:45:49
2   <=    2   on-line   since 10/30/2004 13:45:49
3   <=    3   on-line   since 10/30/2004 13:45:49
 ...            ..................
21  <=   21   on-line   since 10/30/2004 13:45:49
22  <=   22   on-line   since 10/30/2004 13:45:49
23  <=   23   on-line   since 10/30/2004 13:45:49
24  <=   512  on-line   since 10/30/2004 13:45:49
25  <=   513  on-line   since 10/30/2004 13:45:49
26  <=   514  on-line   since 10/30/2004 13:45:49
 ...            ..................
                535  on-line   since 10/30/2004 13:45:49
```

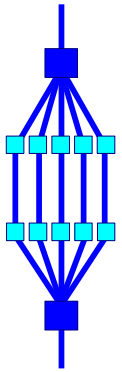
Fragment of psrinfo output

Logical ID

Fragment of prtdiag output

FRU Name	CPU ID	Run MHz	E$ MB	CPU Impl.	CPU Mask
/N0/SB0/P0	0,512	1200	16.0	US-IV	2.3
/N0/SB0/P1	1,513	1200	16.0	US-IV	2.3
/N0/SB0/P2	2,514	1200	16.0	US-IV	2.3
/N0/SB0/P3	3,515	1200	16.0	US-IV	2.3
/N0/SB1/P0	4,516	1200	16.0	US-IV	2.3
	.....				
/N0/SB5/P2	22,534	1200	16.0	US-IV	2.3
/N0/SB5/P3	23,535	1200	16.0	US-IV	2.3

Examples SUNW_MP_PROCBIND



Activate binding of threads to processors

```
% setenv SUNW_MP_PROCBIND TRUE
```

(binding will start at processor 0)

Bind threads to processor 5, 6, 7, …., 10 and 11

```
% setenv SUNW_MP_PROCBIND 5-11
```

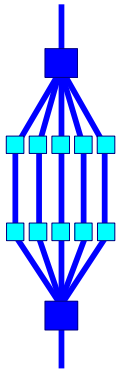
Bind threads to processor 5, 6, 7,, ……, 0, 1, 2

```
% setenv SUNW_MP_PROCBIND 5
```

Bind threads to processor 0, 24, 1, 25, 2 and 26

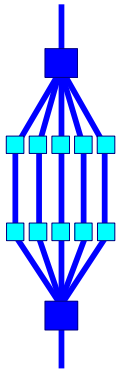
```
% setenv SUNW_MP_PROCBIND "0 24 1 25 2 26"
```

Note: this is the logical, not physical, numbering



Autoscoping

Autoscoping example (Fortran only)

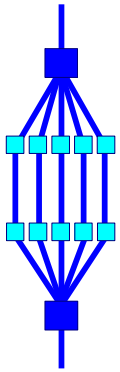


*Autoscoping is a unique feature available in the Sun Fortran compiler only**

```
!$OMP PARALLEL DEFAULT (__AUTO)

!$OMP SINGLE
      T = N*N
!$OMP END SINGLE

!$OMP DO
      DO I = 1, N
         A(I) = T + I
      END DO
!$OMP END DO

!$OMP END PARALLEL
```

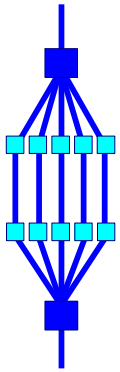
*) C/C++ will be supported in a future release

Autoscoping results



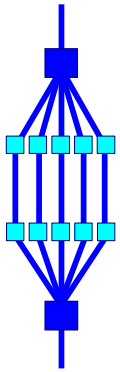
```
Shared variables in OpenMP construct below: a, i, t, n
Variables autoscoped as SHARED in OpenMP construct below: i, t, n, a

 10. !$OMP PARALLEL DEFAULT (__AUTO)
 11.
 12. !$OMP SINGLE
 13.       T = N*N
 14. !$OMP END SINGLE
 15.

Private variables in OpenMP construct below: i
 16. !$OMP DO

Loop below parallelized by explicit user directive
 17.       DO I = 1, N
     <Function: _$d1A16.auto_>
 18.          A(I) = T + I
 19.       END DO
 20. !$OMP END DO
 21.
 22. !$OMP END PARALLEL
```

Variable 'i' re-scoped



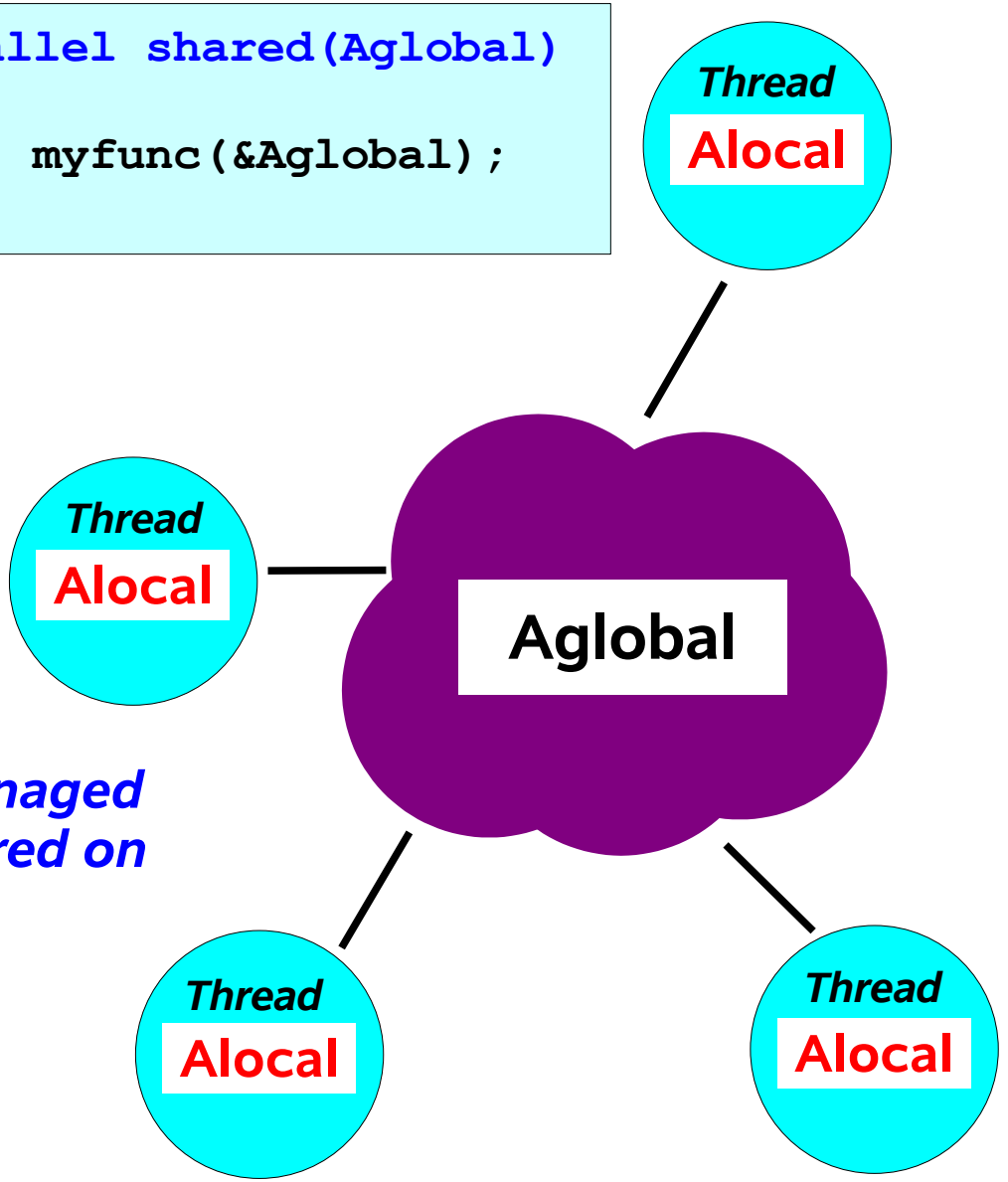
The Stack

About the stack



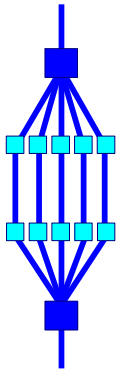
```
#omp parallel shared(Aglobal)
{
    (void) myfunc(&Aglobal);
}
```

```
void myfunc(float *Aglobal)
{
     int Alocal;

     .......
}
```



Alocal is in private memory, managed by the thread owning it, and stored on the so-called stack

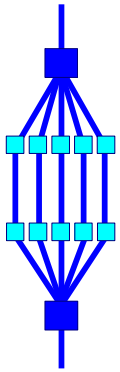
Setting the stack size



`STACKSIZE n`	*Set thread stack size in n KByte*

- *Each thread has its own private stack space*
- *If a thread runs out of this stack space, your program will crash with a segmentation violation*
- *Use the Unix "limit/ulimit" command to increase the MAIN ("initial" thread) stack size*
- *Use the **STACKSIZE** environment variable to increase the stack size for each of the worker threads*
- *Default value for STACKSIZE:*
  - ✔ *4 MByte for 32-bit addressing*
  - ✔ *8 MByte for 64-bit addressing*

Example STACKSIZE



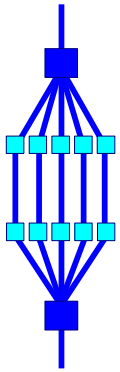
```
#define N 2000000

void myFunc(int TID, double *check);

void main()
{
   double check, a[N];
   int    TID;

#pragma omp parallel private(TID,check)
  {
    TID = omp_get_thread_num();

    myFunc(TID,&check);

  } /*-- End of parallel region --*/

}
```

Main requires about 16 MByte stack space to run

```
#define MYSTACK 1000000

void myFunc(int TID, double *check)
{
   double mystack[MYSTACK];
   int    i;

   for (i=0; i<MYSTACK; i++)
      mystack[i] = TID + 1;

   *check = mystack[MYSTACK-1];

   printf("Thread %d has initialized
           local data\n",TID);
}
```

Function requires about ~8 MByte stack space to run

Runtime behaviour



```
% setenv OMP_NUM_THREADS 1
% limit stack 10k
% ./stack.exe
Segmentation Fault (core dumped)
% limit stack 16m
% ./stack.exe
Thread 0 has initialized local data
```

Not enough stack space for master thread

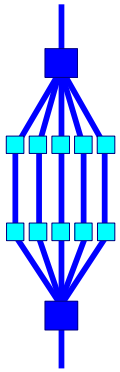
Now runs fine on 1 thread

```
% setenv OMP_NUM_THREADS 2
% ./stack.exe
Segmentation Fault (core dumped)
% setenv STACKSIZE 8192
% setenv OMP_NUM_THREADS 1
% ./stack.exe
Thread 0 has initialized local data
% setenv OMP_NUM_THREADS 2
% ./stack.exe
Thread 0 has initialized local data
Thread 1 has initialized local data
% setenv OMP_NUM_THREADS 4
% ./stack.exe
Thread 0 has initialized local data
Thread 2 has initialized local data
Thread 3 has initialized local data
Thread 1 has initialized local data
```

But crashes on 2 ....

Increase thread stacksize and all is well again

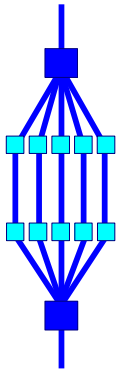
Default stack traceback



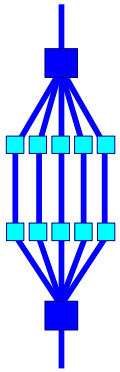
```
% ./stack.exe
Segmentation Fault (core dumped)
% pstack core
core 'core' of 10043:   ./stack.exe
-----------------  lwp# 2 / thread# 2  --------------------
 00010850 myFunc   (1, fe3ffda0, 0, 1, 0, 0) + 10
 0001082c _$p1A19.main (0, fe793380, 80, 10820, feb68260, 0) + c
 feb6834c run_job_invoke_mfunc_once (fe793380, 0, ffbff9a8, 1, 0, 0) + ac
 feb686b4 run_my_job (fe793380, 0, ffbff9a8, 2, 1, 27395000) + 20
 feb736a4 slave_startup_function (feb97290, fe7933d0, fe7933a8, 1, 2,
          feb97284) + 7dc
 feb457b4 _lwp_start (0, 0, 0, 0, 0, 0)
-----------------  lwp# 1 / thread# 1  --------------------
 000108ac myFunc   (f4238, ffbff698, 0, ffbff698, 1438, ff4685f0) + 6c
 0001082c _$p1A19.main (0, fe782100, 80, 10820, feb68260, 0) + c
 feb6834c run_job_invoke_mfunc_once (fe782100, 0, ffbff9a8, 1, ffbff768,
          ffbff879) + ac
 feb67914 __mt_MasterFunction_rtc_ (107a0, fe782180, 0, 13, fe782334, 0) +
          51c
 0001080c main     (1, 13, 702, 107a0, 10400, 10820) + 4c
 00010788 _start   (0, 0, 0, 0, 0, 0) + 108
```

pstack is a very useful Solaris command !

Compiler support: -xcheck=stkovf

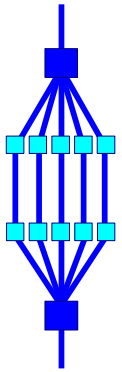


```
% cc -o stack_stkovf.exe -fast -g -xopenmp -xcheck=stkovf *.c
% ./stack_stkovf.exe
Segmentation Fault (core dumped)
% pstack core
core 'core' of 10077:   ./stack_stkovf.exe

-----------------  lwp# 2 / thread# 2  --------------------
 feb45bb4 _stack_grow (1, fe3ffda0, 0, 1, 0, 0) + 48
 00010890 _$p1A19.main (0, fe793380, 80, 10880, feb68260, 0) + 10
 feb6834c run_job_invoke_mfunc_once (fe793380, 0, ffbff988, 1, 0, 0) + ac
 feb686b4 run_my_job (fe793380, 0, ffbff988, 2, 1, 27395000) + 20
 feb736a4 slave_startup_function (feb97290, fe7933d0, fe7933a8, 1, 2,
          feb97284) + 7dc
 feb457b4 _lwp_start (0, 0, 0, 0, 0, 0)
-----------------  lwp# 1 / thread# 1  --------------------
 00010904 myFunc   (f4238, ffbff678, 0, ffbff678, 1340, ff467e10) + 64
 00010890 _$p1A19.main (0, fe782100, 80, 10880, feb68260, 0) + 10
 feb6834c run_job_invoke_mfunc_once (fe782100, 0, ffbff988, 1, ffbff748,
          ffbff859) + ac
 feb67914 __mt_MasterFunction_rtc_ (10800, fe782180, 0, 13, fe782334, 0) +
          51c
 00010870 main     (1, 13, 702, 10800, 10800, 10880) + 50
 000107e8 _start   (0, 0, 0, 0, 0, 0) + 108
```



The Sun Performance Analyzer

Main Features



- Supports serial Fortran, C, C++ and Java programs
- Also supports:
  - Automatic Parallelization
  - OpenMP
  - MPI
  - Posix Threads
  - Java Threading Model
- Does not require a re-compile
  - Recommended to use -g for maximum information
- Very powerful tool, but yet easy to use

Demo Time