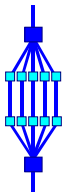


OMPlab On SUN Systems

**Ruud van der Pas
Nawal Copty**

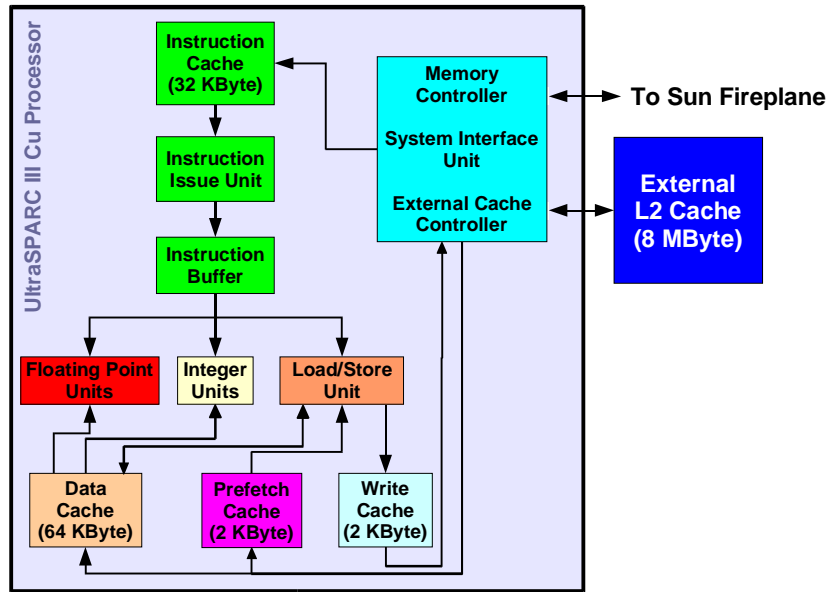
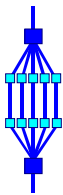
**Scalable Systems Group
Sun Microsystems**

**EWOMP 2004
KTH Royal Institute of Technology
Stockholm, Sweden
October 20-21, 2004**

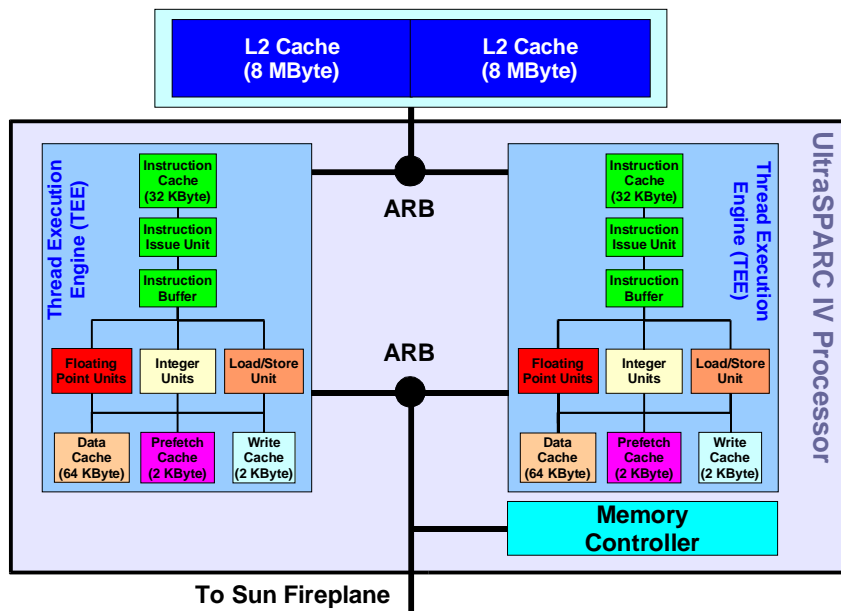
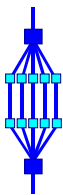


Hardware Architecture

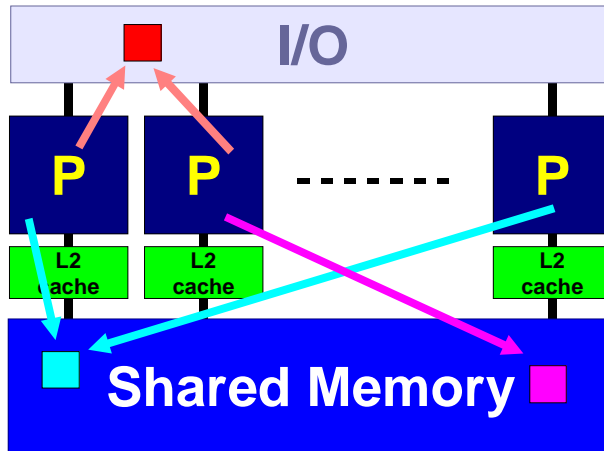
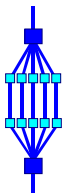
US III Cu - Block Diagram



US IV - Block Diagram

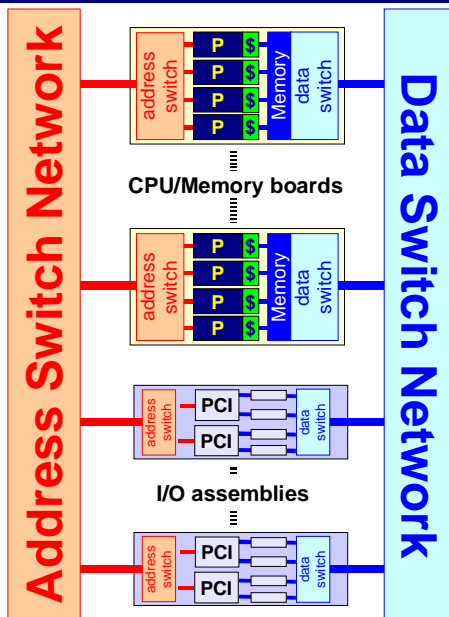
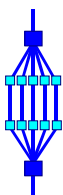


Shared Memory Architecture



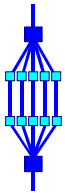
- ✓ Easy to use and administer
- ✓ Efficient use of resources
- ✓ Scales as needs grow

The Simplified Big Picture

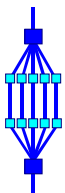
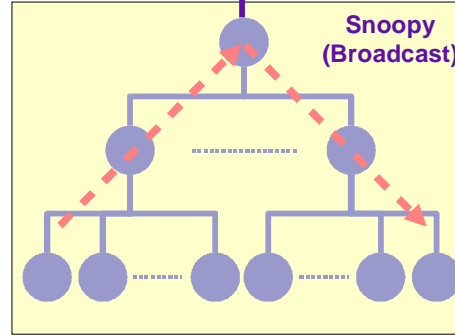
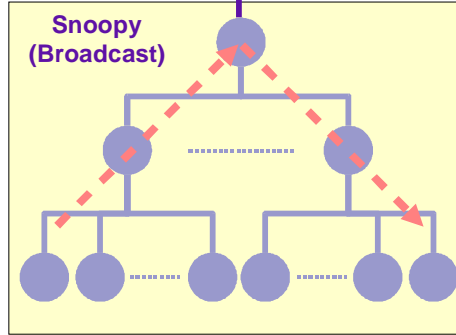


- ✓ The SMP model is preserved throughout product line
- ✓ Architectural details of the switch network depend on Sun Fire model
- ✓ A hierarchical tree is used to build the interconnect
- ✓ Smaller systems, have less switch layers
- ✓ Largest system, the Sun Fire E25K, can have up to 104 US III (Cu)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

A Hierarchical Tree

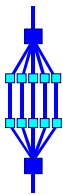


Point-To-Point/SSM
(Sun Fire 12K2/E20K and 15K/E25K)



Serial Performance

Minimal Effort



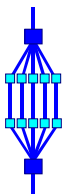
In general, one obtains very good performance out of the Sun compilers by just using 3 options on the compile and link line:

For the UltraSPARC III Cu processor:

```
-fast -xchip=ultra3cu -xarch=v8plusb (32-bit addressing)
-fast -xchip=ultra3cu -xarch=v9b      (64-bit addressing)
```

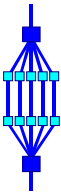
- ◆ The **-fast** option is a macro that expands to a series of options
- ◆ Purpose of **-fast** is to give you very good performance with just one single option
- ◆ The **-fast** option works fine for many applications, but does make some assumptions. When in doubt whether this is acceptable, one is advised to check the documentation about the details.

Beyond -fast (Studio 9)



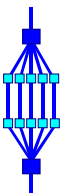
Option	Description	f95	cc	CC	Comp	Link
-xinline	Controls inlining	yes	yes	yes	+	-
-xipo	Interprocedural analysis	yes	yes	yes	+	+
-xprofile	Profile feedback	yes	yes	yes	+	+
-xprefetch	Prefetch on/off	yes	yes	yes	+	-
-xprefetch_level	Controls prefetch algorithm	yes	yes	yes	+	-
-stackvar	Local data on stack	yes	no	no	+	-
-xvector	Vectorization of intrinsics	default	yes	yes	+	+
-xalias	Aliasing of variables	yes	no	no	+	-
-xalias_level	Aliasing of data types	no	yes	yes	+	-
-xsfpconst	Unsuffixd fp consts are single	no	yes	no	+	-
-xrestrict	Restricted pointers (or not)	no	yes	yes	+	-

The Sun Performance Library

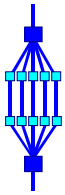


- **Amongst other functionality, this library contains:**
 - **Linear Algebra routines - BLAS levels 1-3, LAPACK**
 - **Fourier Transforms - FFTPACK, VFFTPACK**
- **Supports both 32-bit and 64-bit addressing modes**
- **Tuned for serial and shared memory performance**
- **Add `-xlic_lib=sunperf` on the link line to use the library:**
 - **Be aware of the `LD_LIBRARY_PATH` environment variable**
 - ✓ **Don't set it yourself explicitly, unless you know what you're doing**
 - **Use the `(p)ldd` command to check which version you get**

Compiler Commentary



- **Get information about the optimizations performed:**
 - **Loop transformations and parallelization (`iropt`)**
 - **Instruction scheduling (`cg`)**
- **How to get these messages:**
 - **Add `-g` to the other compiler options you use**
 - **Example: `% cc -c -fast -xarch=v8plus -g funcA.c`**
- **Two ways to display the compiler messages:**
 - **Use the `er_src` command to display the messages on the screen**
 - ✓ **Example: `% er_src funcA.o`**
 - **Shown in analyzer source window**



```
1. void mxv_col(int m, int n, double *a, double *b, double *c)
```

```
2. {
3.     int i, j;
```

```
4.
5. Loop below fused with loop on line 10
```

```
6.     for (i=0; i<m; i++)
7.         a[i] = b[i*n]*c[0];
```

```
8. Loop below interchanged with loop on line 10
```

```
9.     for (j=1; j<n; j++)
10.    {
```

```
11. Loop below interchanged with loop on line 8
```

```
12. Loop below fused with loop on line 5
```

```
13.     for (i=0; i<m; i++)
```

```
14. Loop below pipelined with steady-state cycle count = 2
15. before unrolling
```

```
16. Loop below unrolled 8 times
```

```
17. Loop below has 2 loads, 0 stores, 4 prefetches, 1 FPadds,
18. 1 FPMuls, and 0 FPdivs per iteration
```

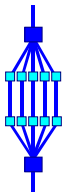
```
19.         a[i] += b[i*n+j]*c[j];
```

```
20.     }
```

```
21. }
```

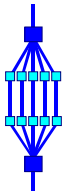
IRopt messages

CG messages



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
- The order in which the iterations are executed is:

- *Undetermined*
- *Different from run to run*

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

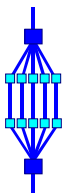
Thread 0

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

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

Thread 1

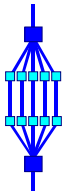
Automatic Parallelization Options



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 which loops are parallelized and which are not

- ◆ *Use the environment variable **OMP_NUM_THREADS** to set the number of threads*
- ◆ *For efficiency reasons, we recommend to only compile the most time consuming parts of the program for automatic parallelization*
- ◆ *Use the Sun Performance Analyzer to find those parts*

Loop Versioning



□ For every automatically parallelized loop, the compiler will generate two versions*:

- A serial version

- ✓ Executed serially if there is not enough work in the loop

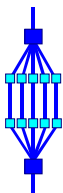
- A parallel version

- ✓ Executes in parallel if there is enough work

□ A barrier is executed after every parallelized loop

*) Not done if the loop has constant bounds or if the compiler can derive the loop lengths from the context

Loop Versioning Example



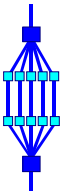
```
% cc -c -g -fast -xrestrict -xautopar -xloopinfo subl.c
```

```
1 void subl(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 }
```

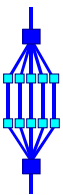
"sublc.c", line 4: PARALLELIZED, and serial version generated

◆ In this case, the compiler generates two versions

◆ The serial version will be executed if there is not enough work to be done in the loop



OpenMP on SUN systems

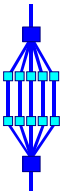


OpenMP Compiler Options

Option	Description
<code>-xopenmp=parallel</code>	Enables recognition of OpenMP pragmas Requires at least <code>-xO3</code>
<code>-xopenmp=noopt</code>	Enables recognition of OpenMP pragmas Can not set an optimization level
<code>-xopenmp=none</code>	Disables recognition of OpenMP pragmas

Note: `-xopenmp` is equivalent to `-xopenmp=parallel`

Example OpenMP Options



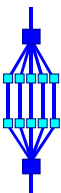
```
% cc -c -g -xopenmp=noopt -xloopinfo parloop.c
"parloop.c", line 18: PARALLELIZED, user pragma used

% cc -c -g -xopenmp=noopt -xloopinfo -xO2 parloop.c
cc: Optimization level must be >= 3, or not specified at all,
with -xopenmp=noopt

% cc -c -g -xopenmp=noopt -xloopinfo -xO3 parloop.c
"parloop.c", line 18: PARALLELIZED, user pragma used

% cc -c -g -xopenmp -xloopinfo -xO3 parloop.c
"parloop.c", line 18: PARALLELIZED, user pragma used
```

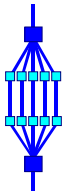
OpenMP Example



```
1 void mxv_row(int m,int n,double *a,double *b,double *c)
2 {
3   int i, j;
4   double sum;
5
6   #pragma omp parallel for default(none) \
7       private(i,j,sum) shared(m,n,a,b,c)
8   for (i=0; i<m; i++)
9   {
10      sum = 0.0;
11      for (j=0; j<n; j++)
12          sum += b[i*n+j]*c[j];
13      a[i] = sum;
14  }
15 }
```

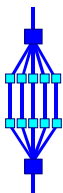
```
% cc -c -fast -xrestrict -xopenmp -xloopinfo mxv_row.c
"mxv_row.c", line 8: PARALLELIZED, user pragma used
"mxv_row.c", line 11: not parallelized
```

Related Compiler Options



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

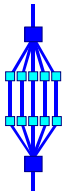
OpenMP Environment Variables



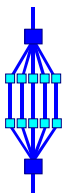
OpenMP environment variable	Default Value	Sun OpenMP
<code>OMP_NUM_THREADS <u>n</u></code>	Implementation dependent	1
<code>OMP_SCHEDULE "<u>schedule</u>,<u>[chunk]</u>"</code>	Implementation dependent (<code>chunk=1</code> , except for static)	static "N/P"
<code>OMP_DYNAMIC { TRUE FALSE }</code>	Implementation dependent	TRUE*
<code>OMP_NESTED { TRUE FALSE }</code>	Implementation dependent	FALSE

**) 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.*

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



Sun-specific OpenMP Environment Variables



The Behaviour Of Idle Threads

Environment variable to control the behaviour:

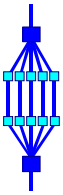
```
SUNW_MP_THR_IDLE
[ spin | sleep | sleep (ns) | sleep (nms) ]
```

- ◆ Default is to "spin" i.e. block the CPU (will change in next release)
- ◆ Sleep: thread is put to sleep; awakened when new work arrives
- ◆ Sleep ('time'): spin for 'time' s (or ms), then go into sleep mode

When to set SUNW_MP_THR_IDLE to sleep ?

- ◆ If the number of threads exceeds the number of processors
- ◆ Parallel regions constitute a small portion of the total execution time of the program (i.e. long serial portions in the code)
- ◆ To use the CPU time effectively (throughput versus single job performance)

Sun-specific Environment Variables



SUNW_MP_WARN TRUE | FALSE

Control printing of warnings

- ☞ **WARNING:** The MT run-time library will not print warning messages by default
- ☞ Set this environment variable to TRUE to activate the warnings

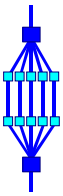
Control binding of threads to "processors"

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
- ☞ Will be ignored in case of syntax error; if warnings are enabled, a message will be issued

Examples SUNW_MP_PROCBIND



Activate binding of threads to processors

```
% setenv SUNW_MP_PROCBIND TRUE
```

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

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

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

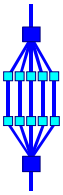
```
% setenv SUNW_MP_PROCBIND "0 24 1 25 2 26"
```

Bind threads to processor 5, 0, 1, 2,

```
% setenv SUNW_MP_PROCBIND 5
```

- ▶ One can use the `/usr/sbin/psrinfo` and `prtdiag` commands to find out how processors are configured
- ▶ The `prtdiag` command can be found in `/usr/platform`
- ▶ For example in `/usr/platform/sun4u/sbin/prtdiag`

Configuration Information

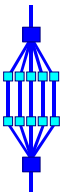


FRU Name	Port ID	Run MHz	E\$ MB	CPU Impl.	CPU Mask
/NO/SB0/P0	0	900	8.0	US-III+	2.1
/NO/SB0/P1	1	900	8.0	US-III+	2.1
/NO/SB0/P2	2	900	8.0	US-III+	2.1
/NO/SB0/P3	3	900	8.0	US-III+	2.1
/NO/SB1/P0	4	900	8.0	US-III+	2.2
/NO/SB1/P1	5	900	8.0	US-III+	2.2
/NO/SB1/P2	6	900	8.0	US-III+	2.2
/NO/SB1/P3	7	900	8.0	US-III+	2.2
/NO/SB2/P0	8	900	8.0	US-III+	2.1
/NO/SB2/P1	9	900	8.0	US-III+	2.1
/NO/SB2/P2	10	900	8.0	US-III+	2.1
/NO/SB2/P3	11	900	8.0	US-III+	2.1

*Fragment of
prtdiag output*

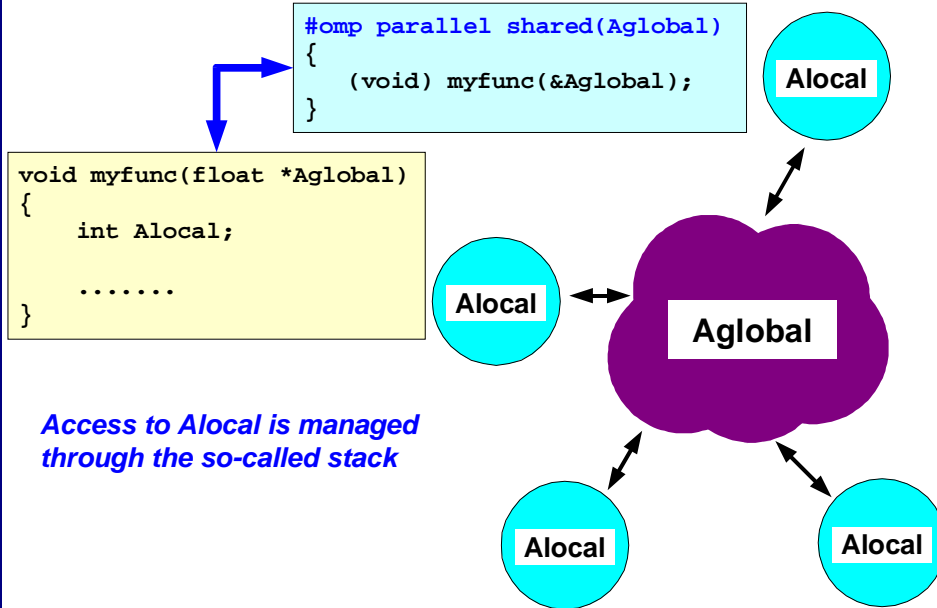
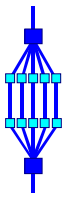
*Fragment of
psrinfo output*

0	on-line	since 12/25/2003 23:36:12
1	on-line	since 12/25/2003 23:38:01
2	on-line	since 12/25/2003 23:38:01
3	on-line	since 12/25/2003 23:38:01
4	on-line	since 12/25/2003 23:38:01
5	on-line	since 12/25/2003 23:38:01
6	on-line	since 12/25/2003 23:38:01
7	on-line	since 12/25/2003 23:38:01
8	on-line	since 12/25/2003 23:38:01

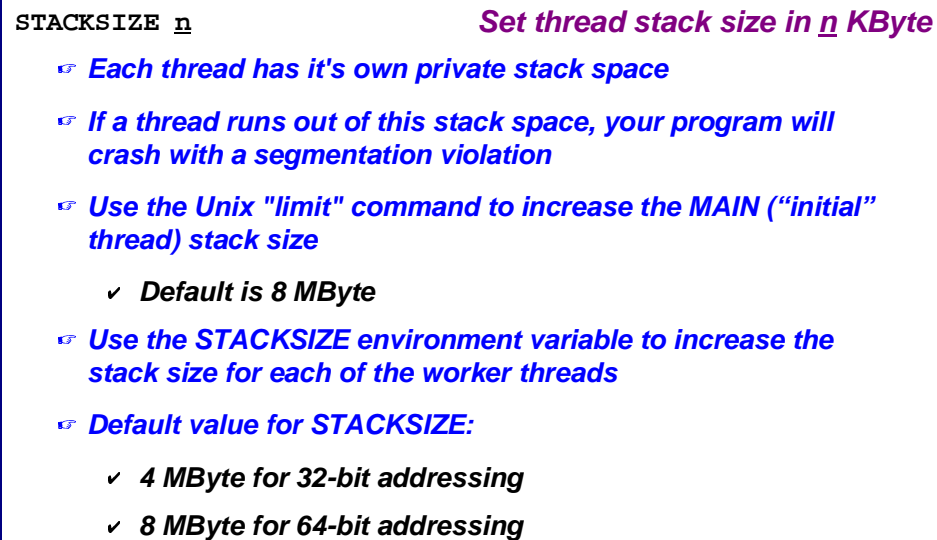
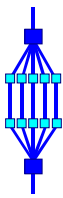


The Stack

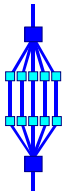
About The Stack



Setting The Stack Size



Example STACKSIZE



```

program main()
  integer, parameter:: n=2000000
  integer, parameter:: p=2
  real(kind=8)       :: a(n), check(p)

!$omp parallel default(none) &
!$omp private(i) shared (check)
!$omp do
  do i = 1, p
    call suba(i,check(i))
  end do
!$omp end do
!$omp end parallel
  print *, 'check=', check

  stop
end

```

*Main requires about 16
MByte stack space to run*

```

subroutine suba(i,check)
  integer      :: i
  real(kind=8) :: check
  integer, parameter:: n=1000000
  real(kind=8)  :: mystack(n)

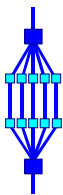
  do i = 1, n
    mystack(i) = i
  end do
  check = mystack(n)

  return
end

```

*Subroutine requires about ~8
MByte stack space to run*

Runtime Behaviour



```

% f95 -fast -g -xopenmp=parallel -xloopinfo main.f95
"main.f95", line 9: PARALLELIZED, user pragma used

```

```

% setenv OMP_NUM_THREADS 1
% limit stack 10k
% ./a.out
Segmentation Fault

```

*Not enough stack space
for master thread*

```

% limit stack 16m

```

```

% ./a.out
check= 500000.0 500000.0

```

Now runs fine on 1 thread

```

% setenv OMP_NUM_THREADS 2
% ./a.out
Segmentation Fault

```

But crashes on 2

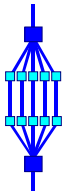
```

% setenv STACKSIZE 8000
% setenv OMP_NUM_THREADS 1
% ./a.out
check= 500000.0 500000.0
% setenv OMP_NUM_THREADS 2
% ./a.out
check= 500000.0 500000.0

```

*Increase thread stacksize
and all is well again*

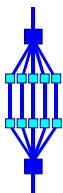
Default Behaviour



```
% f95 -g -fast -xopenmp=parallel main.f95
% setenv OMP_NUM_THREADS 2; unsetenv STACKSIZE
% dbx a.out
<... lines deleted ...>
(dbx) run
Running: a.out (process id 9585)
t@5 (l@6) signal SEGV (no mapping at the fault address) in suba (optimized)
      at line 26 in file "main.f95"
      26      mystack(i) = i
      dbx: read of 4 bytes at address fd0605f0 failed -- Error 0
(dbx) where
current thread: t@5
=>[1] suba(i = ???, check = ???) (optimized), at 0x13ed8 (line ~26) in "main.f95"
[2]  _$dlA8.MAIN_() (optimized), at 0x13f7c (line ~10) in "main.f95"
[3]  __mt_runLoop_int_(0xfd801a18, 0x6ff28, 0x1d424, 0x2, 0x13f60, 0xfd801bd4),
      at 0x1d684
[4]  __mt_run_my_job_(0x6ff28, 0xfd801a18, 0x40, 0x0, 0x0, 0x0), at 0x14b24
[5]  __mt_WorkSharing_(0xfd801b90, 0x13f60, 0xfd801bd4, 0xffbef960, 0x0, 0x0), at
      0x148c0
[6]  _$p1B6.MAIN_() (optimized), at 0x14008 (line ~8) in "main.f95"
[7]  __mt_run_my_job_(0x6ff28, 0xffbef7d8, 0x0, 0x0, 0xffbef960, 0x0), at 0x14c70
[8]  __mt_SlaveFunction_(0x6ff28, 0xfeef690, 0xffbef7d8, 0x68800, 0x64800,
      0x68800), at 0x17e70
(dbx) quit
```

For example only !

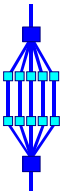
Compiler Support: -xcheck=stkovf



```
% f95 -fast -g -xopenmp -xcheck=stkovf main.f95
% setenv OMP_NUM_THREADS 2; unsetenv STACKSIZE
% dbx a.out
<... lines deleted ...>
(dbx) run
Running: a.out (process id 9590)
t@5 (l@6) signal SEGV (access to address exceeded protections) in
__stack_grow_probing_impl at 0x13ebc
0x00013ebc: __stack_grow_probing_impl+0x001c:  ldsb    [%o3], %g0
Current function is suba (optimized)
      19      subroutine suba(i,check)
(dbx) where
current thread: t@5
[1]  __stack_grow_probing_impl(0xfd0605d0, 0x13ea0, 0x64c00, ...)
=>[2] suba(i = ???, check = ???) (optimized), at 0x14158 (line ~19) in "main.f95"
[3]  _$dlA8.MAIN_() (optimized), at 0x14220 (line ~10) in "main.f95"
[4]  __mt_runLoop_int_(0xfd801a18, 0x702a0, 0x1d6cc, 0x2, 0x14200, ...)
[5]  __mt_run_my_job_(0x702a0, 0xfd801a18, 0x40, 0x0, 0x0, 0x0), at 0x14dc8
[6]  __mt_WorkSharing_(0xfd801b90, 0x14200, 0xfd801bd4, 0xffbef960, ...)
[7]  _$p1B6.MAIN_() (optimized), at 0x142ac (line ~8) in "main.f95"
[8]  __mt_run_my_job_(0x702a0, 0xffbef7d8, 0x0, 0x0, 0xffbef960, 0x0
[9]  __mt_SlaveFunction_(0x702a0, 0xfeef690, 0xffbef7d8, 0x68c00, ...)
(dbx) quit
```

For example only !

Tip: Use pstack On Core File



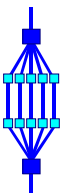
```
% pstack core
```

```
core 'core' of 9579: ./a.out
----- lwp# 5 / thread# 4 -----
00013ebc __stack_grow_probing_impl (fd0018ac, ffbeffa40, 0, 0, 0, 0) + 1c
00014220 _$dlA8.MAIN_ (fd001bd4, 3, 2, ffbeffa30, 10, fd001bd4) + 20
0001d92c __mt_runLoop_int_ (fd001a18, 702a0, 1d6cc, 2, 14200, fd001bd4) + 228
00014dc8 __mt_run_my_job_ (702a0, fd001a18, 40, 0, 0, 0) + 84
00014b64 __mt_WorkSharing_ (fd001b90, 14200, fd001bd4, ffbeffa10, 0, 0) + 260
000142ac $_p1B6.MAIN_ (ffbeffa10, 14000, 2, 1, ffbeffa34, 0) + 6c
00014f14 __mt_run_my_job_ (702a0, ffbef888, 0, 0, ffbeffa10, 0) + 1d0
----- lwp# 1 / thread# 1 -----

00013ebc __stack_grow_probing_impl (ffbef47c, ffbeffa38, 0, 0, 0, 0) + 1c
00014220 _$dlA8.MAIN_ (ffbef7a4, 2, 1, ffbeffa30, 8, ffbef7ac) + 20
0001d92c __mt_runLoop_int_ (ffbef5e8, 6fc38, 1d6cc, 1, 14200, ffbef7a4) + 228
00014dc8 __mt_run_my_job_ (6fc38, ffbef5e8, e0, a0, 0, a0) + 84
00014b64 __mt_WorkSharing_ (ffbef760, 14200, ffbef7a4, ffbeffa10, 0, 0) + 260
000142ac $_p1B6.MAIN_ (ffbeffa10, 14000, 2, 1, ffbeffa34, 0) + 6c
00014f14 __mt_run_my_job_ (6fc38, ffbef888, 0, a0, ffbeffa10, 0) + 1d0
00014748 __mt_MasterFunction_ (0, 0, 68c00, 0, 0, 0) + 38c
000140ac MAIN_ (64c00, fe0b28e0, 14000, 102, ffbeffa34, ffbeffa30) + 4c
----- lwp# 2 / thread# 2 -----

<... rest of output deleted ...>
```

What's Next ?



□ Session II

- **Sun Performance Analyzer demo**

- ✓ **Way cool!**

- **OpenMP autoscopying demo**

- ✓ **Way cool!**

□ OMPlab participants on Sun systems

- **We have prepared a simple lab for you to work on**
- **To play with automatic parallelization and OpenMP**
- **SF15K systems at UU/IT:**
 - ✓ **UltraSPARC III Cu @ 900 MHz**
 - ✓ **Two domains: 36 CPUs and 12 CPUs**