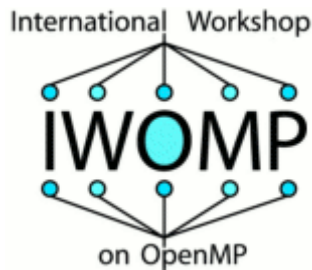


The OMPlab on Sun Systems

Ruud van der Pas



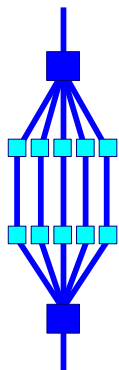
**Senior Staff Engineer
Sun Microsystems
Menlo Park, CA, USA**



**IWOMP 2007
Tsinghua University
Beijing, China
June 3-7, 2007**

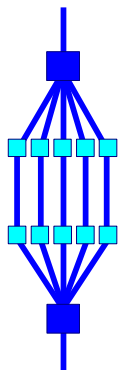
Sun Studio Compilers and Tools

2



- **Fortran (f95), C (cc) and C++ (CC) compilers**
 - **Support sequential optimization, automatic parallelization and OpenMP**
- **Sun Performance Analyzer**
 - **Languages supported: Fortran, C, C++ and Java**
 - **Parallel: POSIX Threads, AutoPar, OpenMP and MPI**
 - **OMPlab focus: Fortran, C, C++, AutoPar, OpenMP**
- **Sun Thread Analyzer**
 - **Languages supported: Fortran, C, C++**
 - **Parallel: POSIX Threads, Solaris Threads, OpenMP**
 - **OMPlab focus: Fortran, C, C++, OpenMP**
- **Not covered here: Sun Studio IDE and other tools**

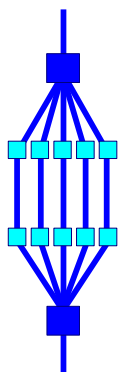
Sequential Performance



- **Recommended option to use: *-fast***
 - ***This is a compile and link option***
- **Recommended to add**
 - ***On SPARC***
 - ✓ ***-xarch=sparcvis2 -m32 (32-bit addressing)***
 - ✓ ***-xarch=sparcvis2 -m64 (64-bit addressing)***
 - ***For AMD Opteron***
 - ✓ ***-xarch=sse2a -m64 -xvector=simd, lib***
For C programs, add: -xprefetch=auto

Additional serial options to explore

4



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.	+	-
-xsfpcnst	Unsuffixd fp consts are single	n.a.	av.	n.a.	+	-
-xrestrict	Restricted pointers (or not)	n.a.	av.	av.	+	-

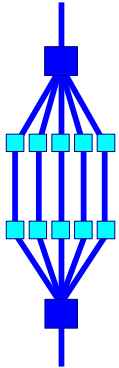
= option is available, but may not be implied, or try out non-default settings

= option is not available or applicable

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

The C compiler on AMD does not set -xprefetch=yes with -fast

Automatic 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];
```

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

Thread 1

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

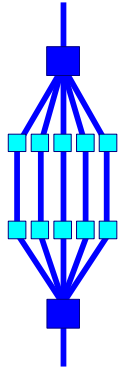
Thread 2

-xautopar option

```
for (i=n*(P-1)/P;i<P;i++)  
  a[i] = b[i] + c[i];
```

Thread P

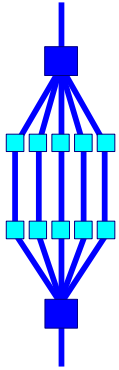
Options For Automatic Parallelization



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



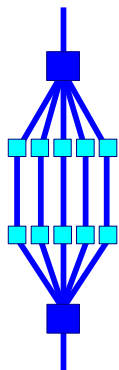
```
% cc -c -g -fast -xrestrict -xautopar -xloopinfo sub1.c
```

```
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 }
```

"sub1.c", line 4: PARALLELIZED, and serial version generated

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

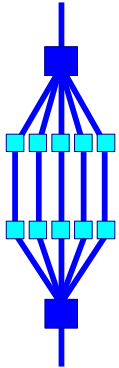
OpenMP Compiler Options



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

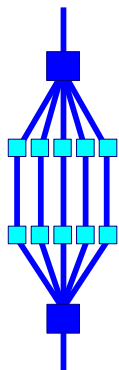
**) The compiler does not raise the optimization level if it is lower than `-xO3`*

Related Compiler Options



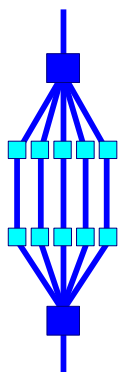
Option	Description
-xloopinfo	Display parallelization messages on screen
-stackvar	Allocate local data on the stack (Fortran only) Use this when calling functions in parallel Included with -xopenmp=parallel noopt
-vpara/-xvpara	Reports OpenMP scoping errors in case of incorrect parallelization (Fortran 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



- **Get information about the optimizations performed:**
 - *Loop transformations and parallelization (irop)*
 - *Instruction scheduling (cg, SPARC only currently)*
- **How to get these messages:**
 - *Add **-g** to the other compiler options you use*
 - *Example: % **cc -c -fast -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 -src funcA.c funcA.o***
 - *Messages are also shown in analyzer source window*

Compiler Commentary with OpenMP



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

Private variables in OpenMP construct below: *j, i*

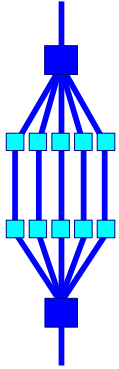
Shared variables in OpenMP construct below: *c, a, b*

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)
```

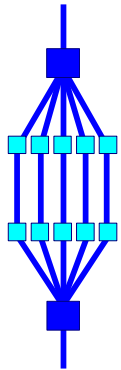
Loop below parallelized by explicit user directive

```
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. }
```



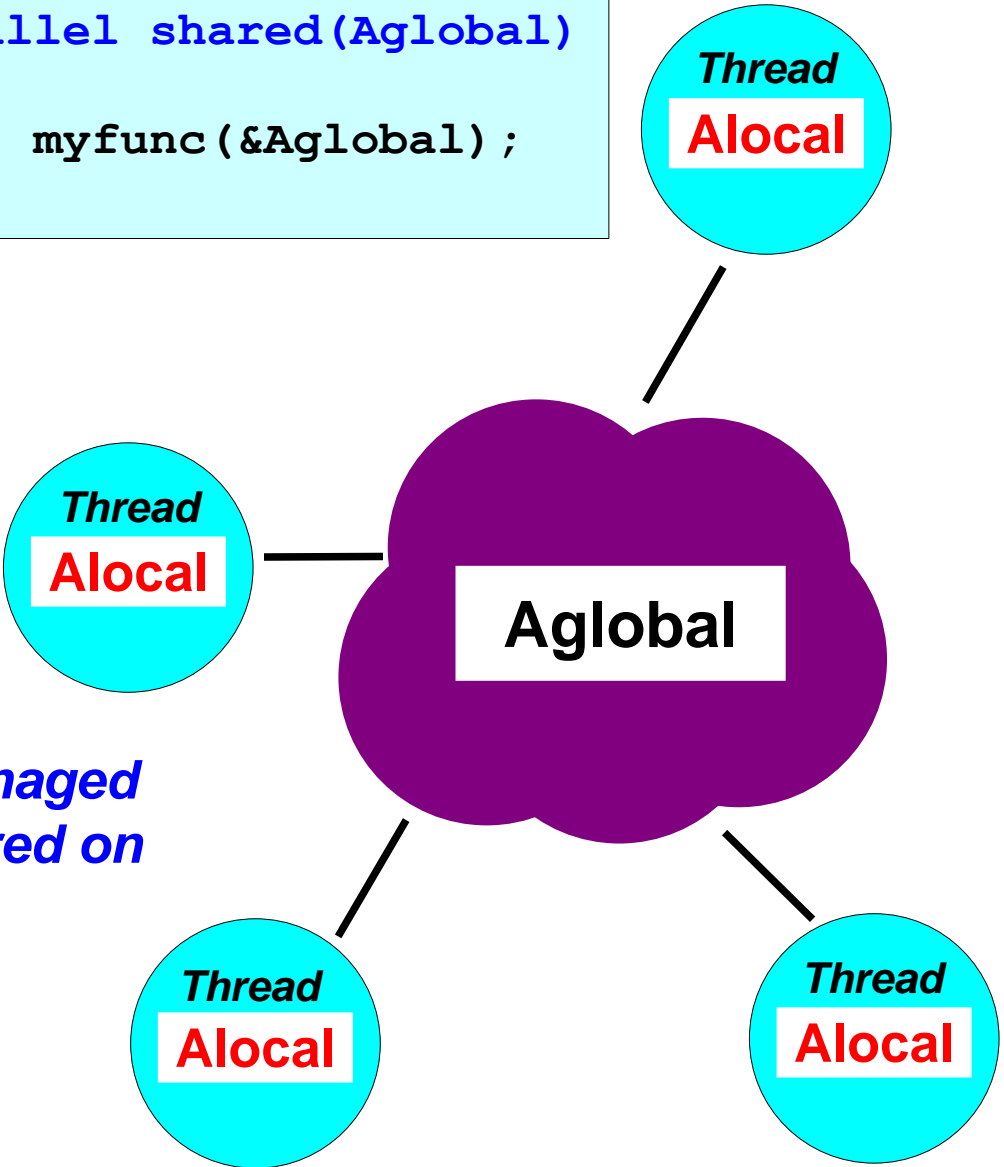
Sun-specific OpenMP Environment Variables

About the stack



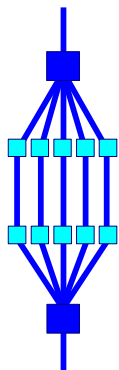
```
#omp parallel shared(Aglobal)
{
  (void) myfunc (&Aglobal) ;
}
```

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



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

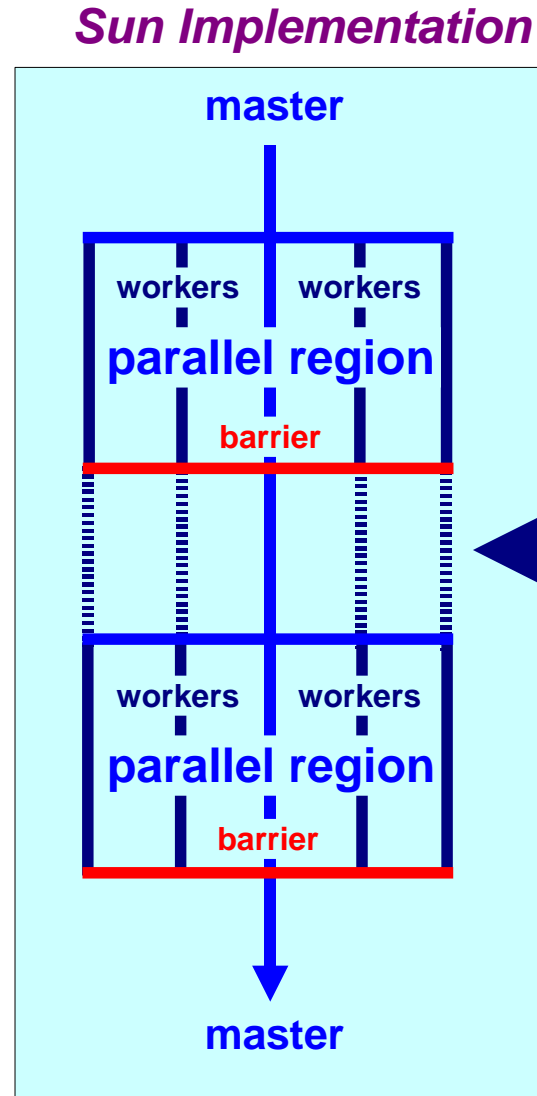
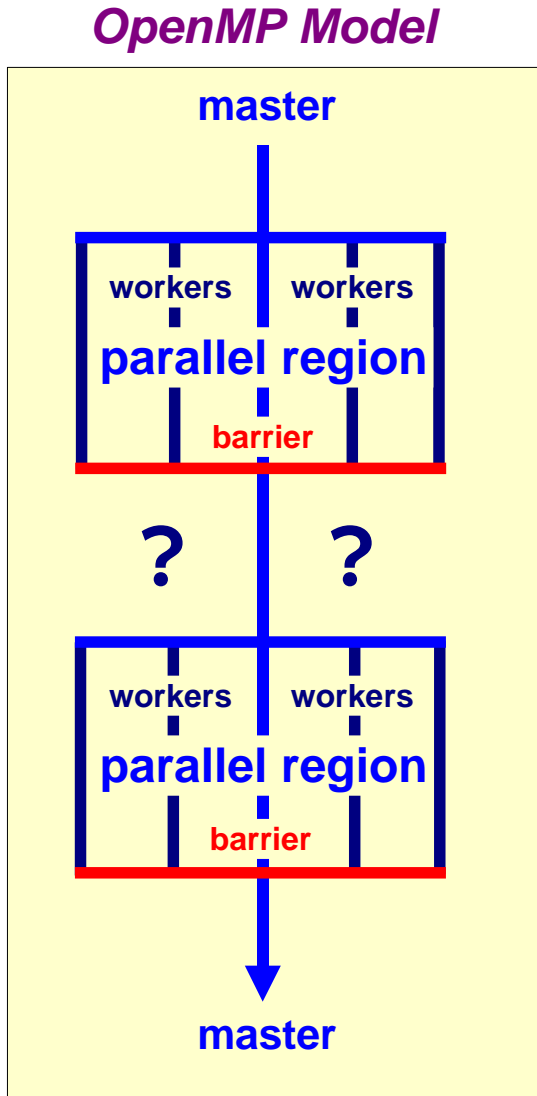
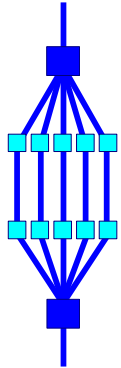
Setting the size of the stack



Set thread stack size in n Byte, KB, MB, or GB
STACKSIZE n [B,K,M,G] *Default is KByte*

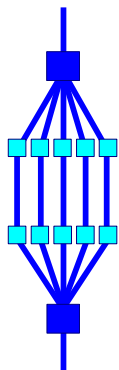
- ☞ ***Each thread has its own private stack space***
- ☞ ***If a thread runs out of this stack space, your program crashes with a segmentation violation***
- ☞ ***Use the Unix "limit/ulimit" command to increase the MAIN ("initial" thread) stack size***
- ☞ ***Use the **STACKSIZE** environment variable 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***

Implementing the Fork-Join Model



*Idle threads sleep
by default*

The behavior of idle threads



Environment variable to control the behavior:

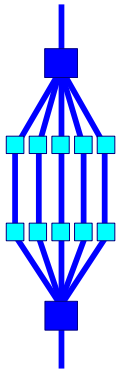
```
SUNW_MP_THR_IDLE  
[ spin | sleep [ ('n's) , ('n'ms) , ('n'us) ] ]
```

- ◆ Default is to have idle threads go to sleep after a spinning for a short while
- ◆ **Spin: threads 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 milli/micro seconds), then go into sleep mode**
 - **Examples: setenv SUNW_MP_THR_IDLE "sleep(5 ms)"**
setenv SUNW_MP_THR_IDLE spin

Run-time warnings

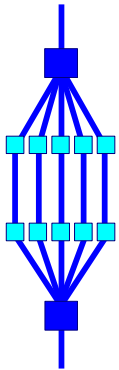
`SUNW_MP_WARN` `TRUE` | `FALSE`

Control printing of warnings



- ☞ *The OpenMP run-time library does not print warning messages by default*
- ☞ *Strongly recommended to set this environment variable to TRUE to activate the warnings*
- ☞ *This helps to 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 is 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 (libmtnsk): 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
```

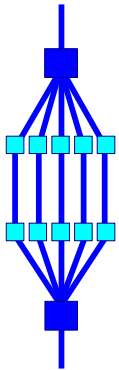
```
# OMP_DYNAMIC=FALSE; export OMP_DYNAMIC  
# ./omp.exe
```

```
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
```

Now we get 3 threads

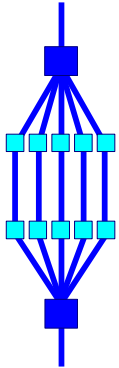


Example SUNW_MP_WARN/2



```
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     }
```

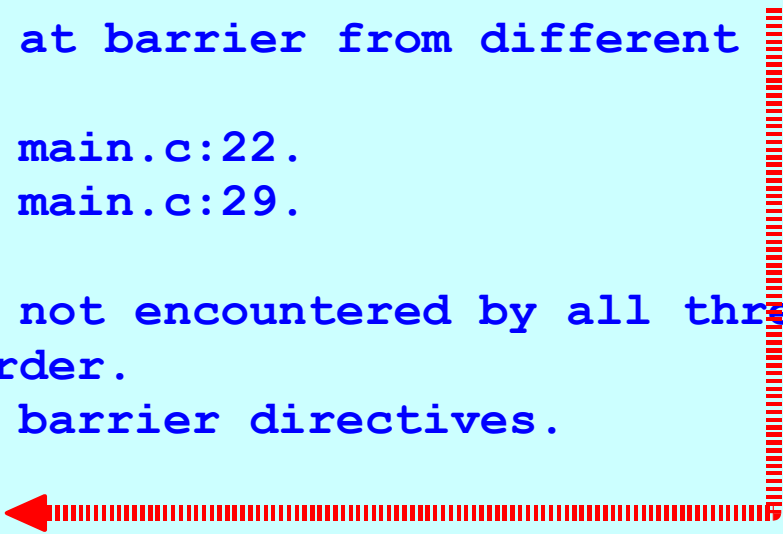
Example SUNW_MP_WARN/3



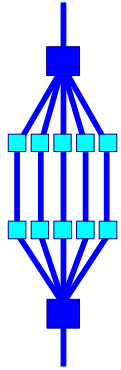
```
% cc -fast -xopenmp -xloopinfo -xvpara main.c
"main.c", line 30: PARALLELIZED, user pragma used
% setenv OMP_NUM_THREADS 4
% setenv SUNW_MP_WARN TRUE
% ./a.out
In bar: n = 5
WARNING (libmtnsk): at main.c:22. Barrier is not permitted in
dynamic extent of for / DO.
Thread 0 does do nothing
Thread 3 does do nothing
Thread 2 does do nothing
Thread 1 does do nothing
WARNING (libmtnsk): 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



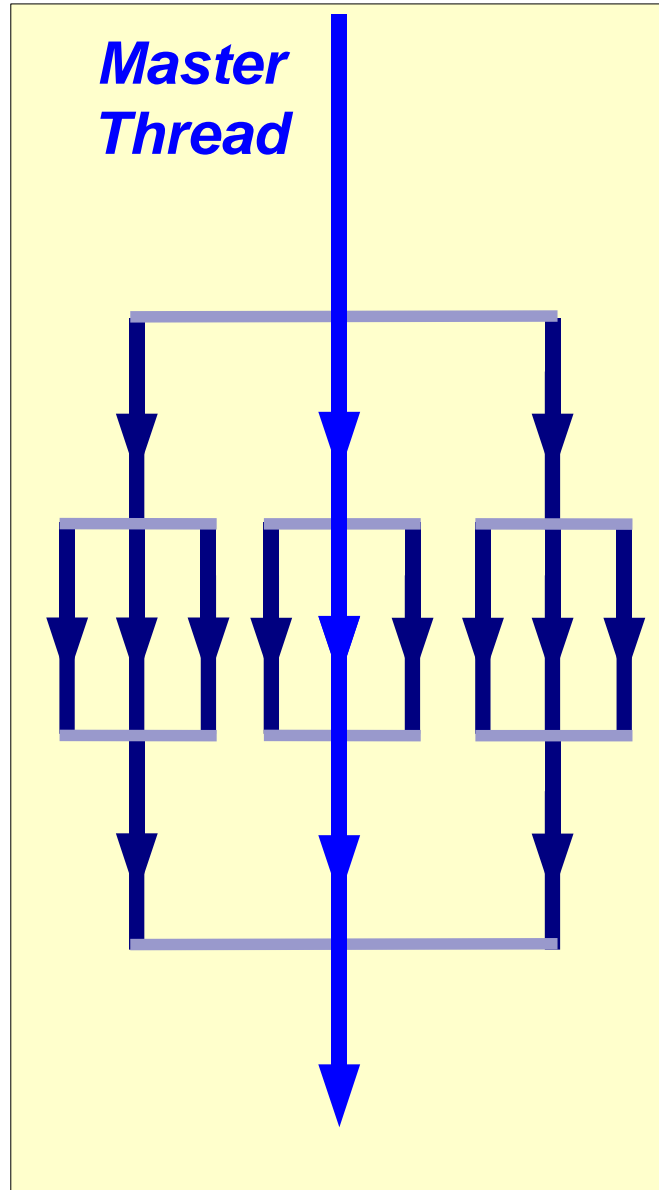
Nested Parallelism



3-way parallel

9-way parallel

3-way parallel



Master Thread

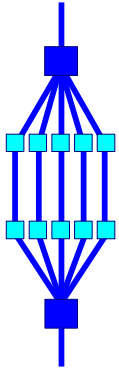
Outer parallel region

Nested parallel region

Outer parallel region

Note: nesting level can be arbitrarily deep

Nested Parallelism on Sun



Control the number of threads in the pool of slave threads used to execute the program

`SUNW_MP_MAX_POOL_THREADS <n>`

Default is 1023

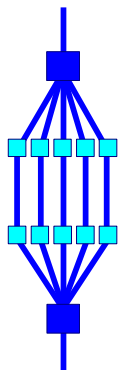
Control the maximum depth of nested active parallel regions

`SUNW_MP_MAX_NESTED_LEVELS <n>`

Default is 4

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

Processor binding

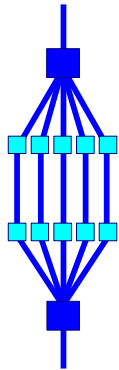


Control binding of threads to “processors”

```
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 is 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

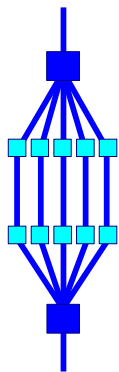
*Fragment of
psrinfo output*

↑
Logical ID

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

*Fragment of
prtdiag output*

Examples SUNW_MP_PROCBIND



Activate binding of threads to processors

```
% setenv SUNW_MP_PROCBIND TRUE
```

(binding starts at processor 0)

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

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

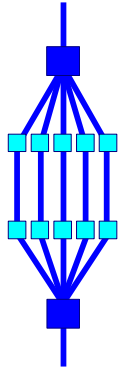
Bind threads to processor 5, 6, 7,,, 0, 1, 2

```
% 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"
```

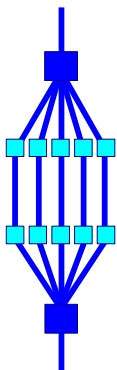
Notes: *This is the logical, not physical, numbering
The third example binds from 5 up to the last processor
and then continues at 0*



Autoscopying

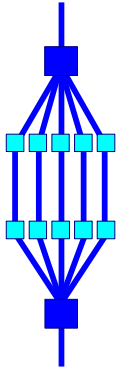
Autoscopying example

Autoscopying is a unique feature available in the Sun Studio compilers 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
```

Autoscopying 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.
```

Variable 'i' re-scoped



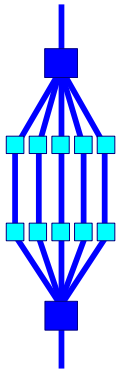
Private variables in OpenMP construct below: i

```
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
```

Autoscopying in C



```
% cc -fast -g -c -xopenmp -xloopinfo -xvpara auto.c  
% er_src -scc parallel -src auto.c auto.o
```

Source OpenMP region below has tag R1

Variables autoscoped as SHARED in R1: a, b, n

Private variables in R1: i

Shared variables in R1: n, a, b

4. #pragma omp parallel for default(__auto)

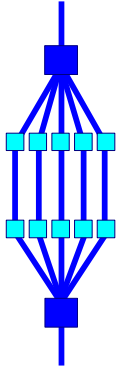
L1 parallelized by explicit user directive

5. for (int i=0; i<n; i++)

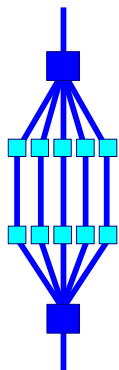
6. a[i] += b[i];

*Note the OpenMP support in
the Compiler Commentary*

Performance Analysis

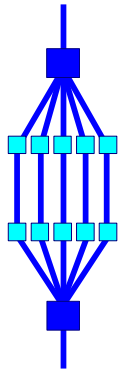


How To Use The Sun Analyzer ?



- ❑ *For the most complete information: recompile with -g*
- ❑ *Two step approach:*
 - *Step 1 - Collect the data:*
 - % collect command - CLI to collect the data*
 - % analyzer command - GUI to collect the data*
 - *Step 2 - Display and analyze the data*
 - % analyzer command - GUI to display the data*
 - % er_print command - CLI to display the data*

Main Window



File View Timeline Help

Find Text:

Functions Callers-Callees Source Lines Disassembly PCs Timeline

User	CPU	User CPU	Wall	Sys. CPU	Name
(sec.)	(%)	(sec.)	(sec.)	(sec.)	
1 574.862	100.0	1 574.862	1 574.912	0.030	<Total>
932.913	59.2	932.913	932.913	0.	Propagate
78.615	5.0	78.615	78.625	0.	__libm_rem_pio2
78.075	5.0	78.075	78.085	0.010	__libm_k_sin
74.102	4.7	74.102	74.112	0.010	tqli
69.288	4.4	69.288	69.288	0.	__libm_k_cos
48.994	3.1	167.517	48.994	0.	wigner_d
46.222	2.9	54.678	46.222	0.	Create Propagator
39.338	2.5	146.412	39.338	0.	cheigs
34.024	2.2	358.531	34.024	0.	Rot
32.353	2.1	152.			
30.371	1.9	30.			
29.190	1.9	134.			
24.287	1.5	1 574.			
19.564	1.2	19.			
15.521	1.0	15.			
8.576	0.5	8.576	8.576	0.	CmatmulRealconst
5.134	0.3	5.134	5.134	0.	zero_cmatrix
2.982	0.2	561.733	2.982	0.	Hamiltonian
2.081	0.1	387.401	2.081	0.	AddTruncQuadrupolar

Most expensive function listed on top

Summary Event Legend Leak

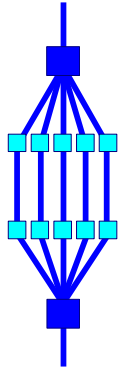
Data for Selected Object:

Name: Propagate
 PC Address: 2:0x0003F6C0
 Size: 1360
 Source File: odes/simulate_case_study/src/complex.c
 Object File: odes/simulate_case_study/src/complex.o
 Load Object: <simulate.base.exe>
 Mangled Name:
 Aliases:

Process Times (sec.) / Counts

	Exclusive	Inclusive
User CPU:	932.913 (59.2%)	932.913 (59.2%)
Wall:	932.913 (59.2%)	932.913 (59.2%)
Total LWP:	932.913 (59.2%)	932.913 (59.2%)
System CPU:	0. (0.%)	0. (0.%)
Wait CPU:	0. (0.%)	0. (0.%)
User Lock:	0. (0.%)	0. (0.%)
Text Page Fault:	0. (0.%)	0. (0.%)
Data Page Fault:	0. (0.%)	0. (0.%)
Other Wait:	0. (0.%)	0. (0.%)

Source Window



File View Timeline Help

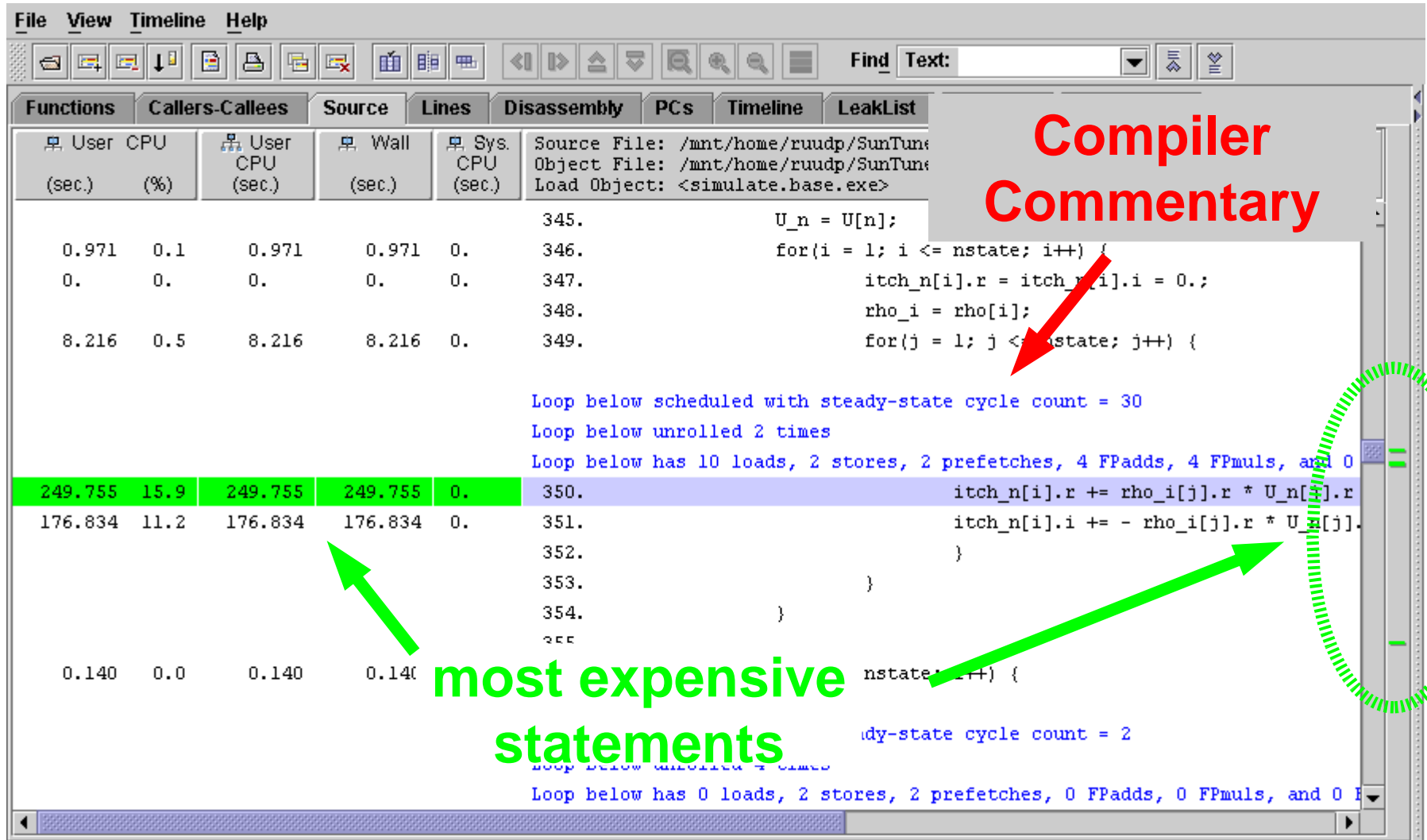
Find Text:

Functions Callers-Callees Source Lines Disassembly PCs Timeline LeakList

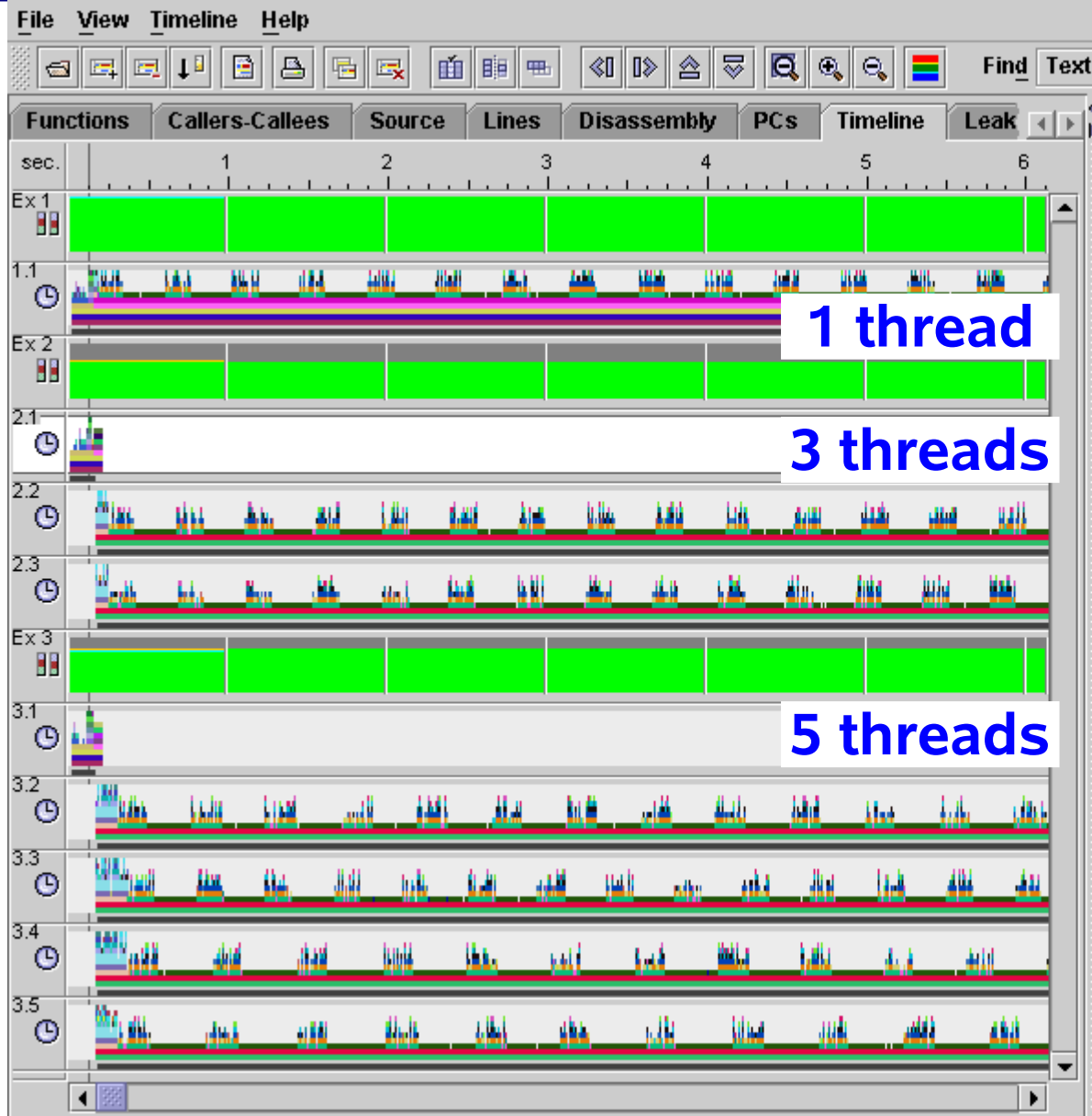
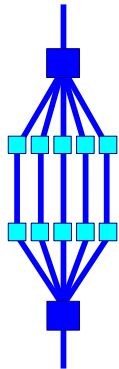
User CPU (sec.)	CPU (%)	User CPU (sec.)	Wall (sec.)	Sys. CPU (sec.)	Source File:	Disassembly
0.971	0.1	0.971	0.971	0.	/mnt/home/ruudp/SunTune	U_n = U[n];
0.	0.	0.	0.	0.	/mnt/home/ruudp/SunTune	for(i = 1; i <= nstate; i++) {
8.216	0.5	8.216	8.216	0.	<simulate.base.exe>	itck_n[i].r = itck_n[i].i = 0.;
						rho_i = rho[i];
						for(j = 1; j <= nstate; j++) {
						Loop below scheduled with steady-state cycle count = 30
						Loop below unrolled 2 times
						Loop below has 10 loads, 2 stores, 2 prefetches, 4 FPadds, 4 FPMuls, and 0
249.755	15.9	249.755	249.755	0.	350.	itck_n[i].r += rho_i[j].r * U_n[j].r
176.834	11.2	176.834	176.834	0.	351.	itck_n[i].i += - rho_i[j].r * U_n[j].r
					352.	}
					353.	}
					354.	}
					...	
0.140	0.0	0.140	0.140	0.140	nstate; i++) {	
						steady-state cycle count = 2
						Loop below unrolled 2 times
						Loop below has 0 loads, 2 stores, 2 prefetches, 0 FPadds, 0 FPMuls, and 0

Compiler Commentary

most expensive statements



Timeline - Parallel Applications



**Application
parallelized
with Posix
threads**

Timestamp (sec.): 0.120529

LWP: 1

Thread: 1

CPU: 2

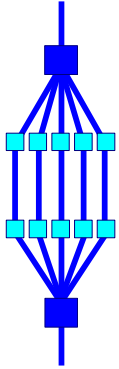
Duration (msec.): 10.007

Micro State: User CPU

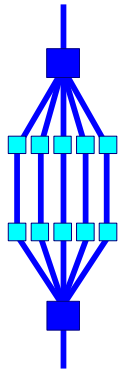
Call Stack for Selected Event

- getc_unlocked + 0x00000000
- file_to_decimal + 0x00001174
- number + 0x000000A4
- __doscan_u + 0x00000090
- __doscan + 0x0000006C
- vfscanf + 0x0000007C
- fscanf + 0x0000001C
- GetPowderAngles + 0x000000F4
- CalcExperiment + 0x00000024
- main + 0x00000244
- _start + 0x00000108

OpenMP Support in the Sun Performance Analyzer



Analyzer Support - OpenMP States



File View Timeline Help

Functions Callers-Callees Source Disassembly Timeli Sum

User CPU (sec.)	User CPU (sec.)	OMP Work (sec.)	OMP Wait (sec.)	Name
17.132	17.132	3.162	118.424	<Total>
12.949	12.949	0.	100.200	<OMP-implicit_barrier>
2.912	2.912	0.	19.713	<OMP-idle>
0.700	14.220	2.312	100.210	main
0.200	0.200	0.320	0.	matrix_mult
0.140	0.140	0.300	0.	matrix_add
0.110	0.110	0.230	0.	matrix_sub
0.070	12.959	0.280	99.630	strassen_mult_bar
0.020	0.020	0.060	0.	strassen_mult
0.010	0.010	0.	0.010	<OMP-overhead>
0.010	0.010	0.010	0.	mutex_lock_imp
0.010	0.340	0.741	0.	strassen_mult
0.	0.	0.030	0.	_fxstat
0.	0.	0.851	0.	_lwp_start
0.	0.	0.010	0.	_read
0.	14.220	2.312	100.210	_start
0.	0.	0.851	0.	_thr_setup

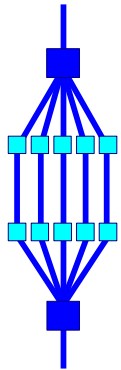
OpenMP "states" as defined by the Analyzer

Name: <OMP-implicit_barrier>
PC Address: 7:0x00000000
Size: 0
Source File: (unknown)
Object File: /lib/libmtnsk.so.1
Load Object: <libmtnsk.so.1>
Mangled Name:
Aliases:

Metrics for Selected Object

	Exclusive
User CPU:	12.949 (75.58%)
Wall:	15.181 (87.89%)
Total LWP:	100.200 (81.74%)
System CPU:	0.010 (20.00%)
Wait CPU:	87.241 (82.77%)
User Lock:	0. (0.%)
Text Page Fault:	0. (0.%)

Analyzer - Callers/Callees



File View Timeline Help

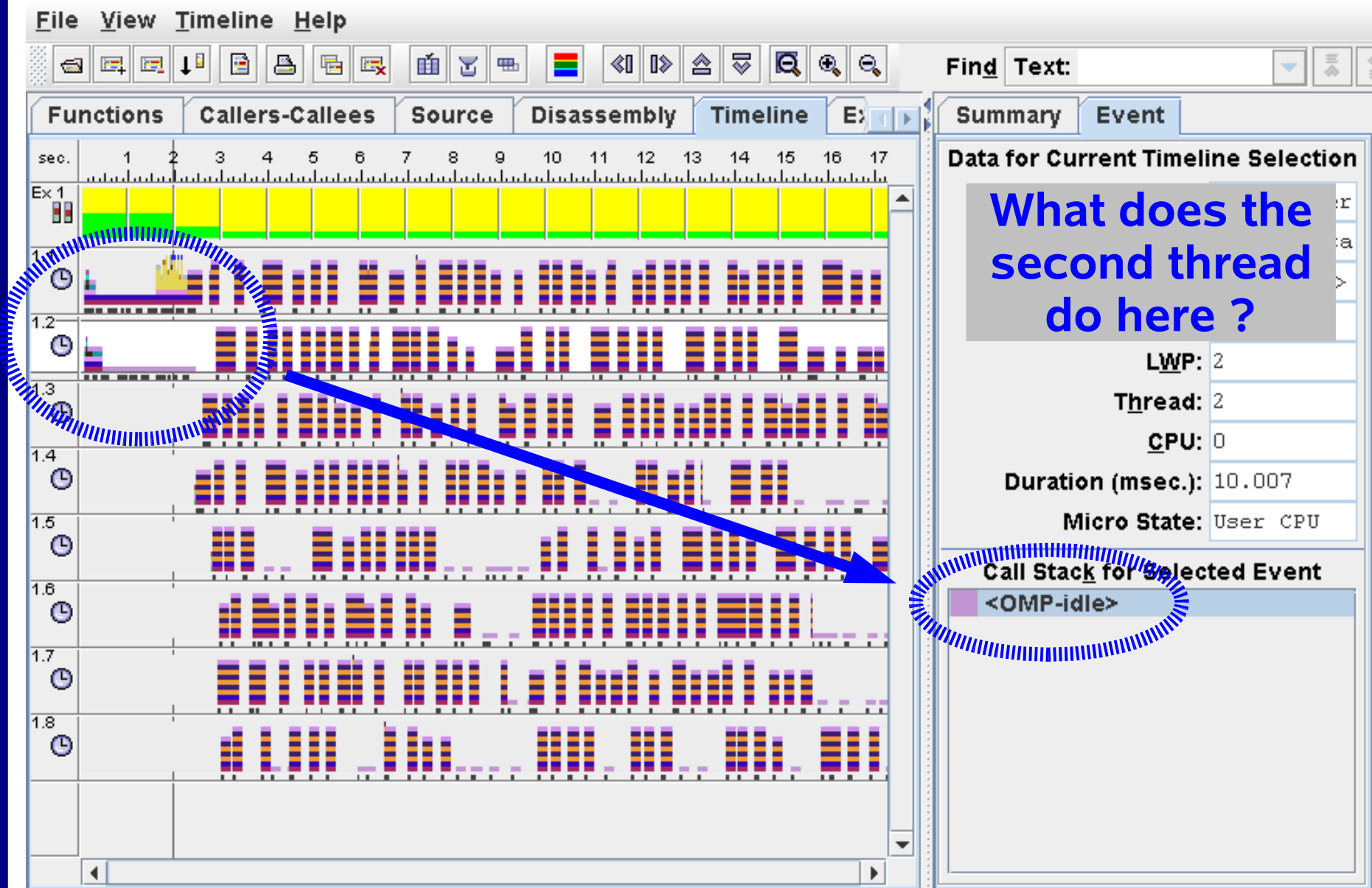
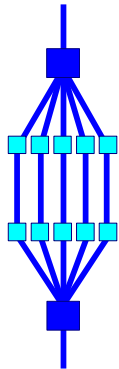
Find Text:

Functions Callers-Callees Source Disassembly Timeline Experiments

User CPU (sec.)	User CPU (sec.)	User CPU (sec.)	OMP Work (sec.)	OMP Work (sec.)	OMP Wait (sec.)	OMP Wait (sec.)	Name
12.709	0.	12.949	0.	0.270	99.620	99.630	strassen_mult_par -- OMP sections f
0.210	0.	0.210	0.	0.	0.470	0.470	initialize_data -- OMP parallel reg
0.030	0.	0.050	0.	0.060	0.110	0.110	initialize_data -- MP doall from li
12.949	12.949	12.949	0.	0.	100.200	100.200	<OMP-implicit barrier>

Identifies time consuming barrier parts in the application

Analyzer - Timeline OpenMP States



File View Timeline Help

Find Text: []

Summary Event

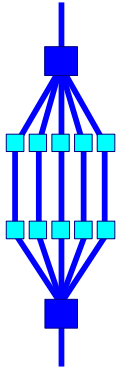
Data for Current Timeline Selection

What does the second thread do here ?

LWP: 2
Thread: 2
CPU: 0
Duration (msec.): 10.007
Micro State: User CPU

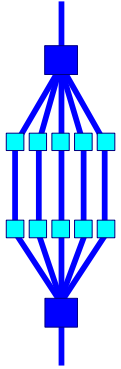
Call Stack for Selected Event

<OMP-idle>

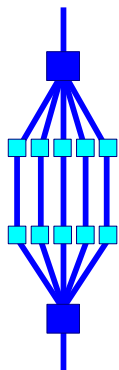


Sun Performance Analyzer Demo

The Sun Thread Analyzer



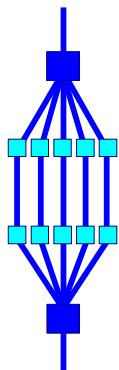
An example of a data race/1



```
#pragma omp parallel default(none) private(i,k,s) \  
    shared(n,m,a,b,c,d,dr)  
{  
    #pragma omp for  
    for (i=0; i<m; i++)  
    {  
        int max_val = 0;  
  
        s = 0 ;  
        for (k=0; k<i; k++)  
            s += a[k]*b[k];  
        c[i] = s;  
  
        dr = c[i];  
        c[i] = 3*s - c[i];  
        if (max_val < c[i]) max_val = c[i];  
        d[i] = c[i] - dr;  
    }  
} /*-- End of parallel region --*/
```

Where is the
data race ?

An example of a data race/2



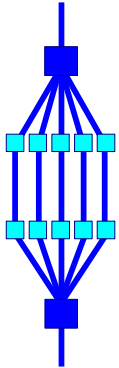
```
#pragma omp parallel default(none) private(i,k,s) \  
    shared(n,m,a,b,c,d,dr)  
{  
    #pragma omp for  
    for (i=0; i<m; i++)  
    {
```

**Here is the data
race !**

```
% cc -xopenmp -fast -xvpara -xloopinfo -c data-race.c  
"data-race.c", line 9: Warning: inappropriate scoping  
variable 'dr' may be scoped inappropriately  
as 'shared'  
. read at line 24 and write at line 21 may  
cause data race
```

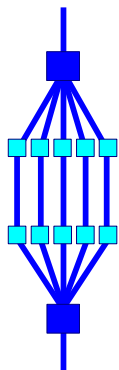
```
    dr = c[i];  
    c[i] = 3*s - c[i];  
    if (max_val < c[i]) max_val = c[i];  
    d[i] = c[i] - dr;  
}  
} /*-- End of parallel region --*/
```

Sun Studio Thread Analyzer



- ❑ *New tool from Sun*
- ❑ *Will be available in Sun Studio 12*
 - *Trial version available now under the Sun Studio Express program*
- ❑ *Detects threading errors in a multi-threaded program*
 - *Data race and Deadlock detection*

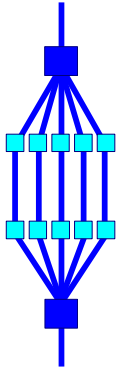
Sun Studio Thread Analyzer



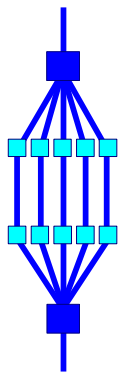
- *Parallel Programming Models supported*:*
 - *OpenMP*
 - *POSIX Threads*
 - *Solaris Threads*
- *Platforms: Solaris on SPARC, Solaris/Linux on x86/x64*
- *Languages: C, C++, Fortran*
- *API provided to inform Thread Analyzer of user-defined synchronizations*
 - *Reduce the number of false positive data races reported*

**) Legacy Sun and Cray parallel directives are supported too*

Sun Thread Analyzer Demo



Using the Thread Analyzer



1. Instrument the code

```
% cc -xinstrument=datarace source.c
```

2. Run the resulting executable under the collect command*. At runtime, memory accesses and thread synchronizations will be monitored. Any data races found will be recorded into a log file

```
% collect -r [ race | deadlock ] a.out
```

3. Display the results:

```
% er_print [-race | -deadlock] tha.1.er
```

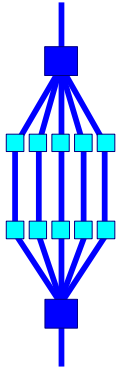
(Command-line interface)

```
% tha tha.1.er
```

(Customized Analyzer GUI)

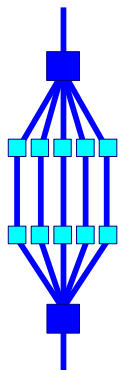
****) Executable will run slower because of instrumentation***

Support for deadlock detection



- ❑ *The Sun Studio Thread Analyzer can detect both **potential** deadlocks and **actual** deadlocks*
- ❑ *A potential deadlock is a deadlock that did not occur in a given run, but can occur in different runs of the program depending on the timings of the requests for the locks by the threads*
- ❑ *An actual deadlock is one that actually occurred in a given run of the program*
 - *An actual deadlock causes the threads involved to hang, but may or may not cause the whole process to hang*

Example command line output



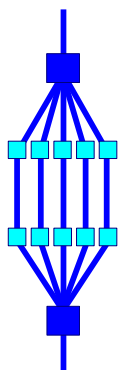
```
Total Races: 2 Experiment: race.er
Race #1, Vaddr: (Multiple Addresses)
  Access 1: Write, main -- MP doall from line 62
                    [_$d1B62.main] + 0x0000007C,
                    line 64 in "race-condition.c"
  Access 2: Read,  main -- MP doall from line 62
                    [_$d1B62.main] + 0x0000006B,
                    line 64 in "race-condition.c"

Total Traces: 1

Race #2, Vaddr: 0x8046a00
  Access 1: Read,  main -- MP doall from line 62
                    [_$d1B62.main] + 0x00000066,
                    line 64 in "race-condition.c"
  Access 2: Write, main -- MP doall from line 62
                    [_$d1B62.main] + 0x00000081,
                    line 64 in "race-condition.c"

Total Traces: 1
```

Thread Analyzer GUI - Races



File View Timeline Help



Find Text:

Races Race Source Experiments

Total Races: 2

Race #1, Vaddr : (Multiple Addresses)

- Access 1: Write, main -- MP doall from line 62 [_\$d1B62.main] line 64 in "race-condition.c"
- Access 2: Read, main -- MP doall from line 62 [_\$d1B62.main] line 64 in "race-condition.c"

Total Traces: 1

Race #2, Vaddr : 0x8046a00

- Access 1: Read, main -- MP doall from line 62 [_\$d1B62.main] line 64 in "race-condition.c"
- Access 2: Write, main -- MP doall from line 62 [_\$d1B62.main] line 64 in "race-condition.c"

Total Traces: 1

Summary Race Details

Data for Selected Race

Id: Race #1

Vaddr: (Multiple Addresses)

Access 1

Type: Write

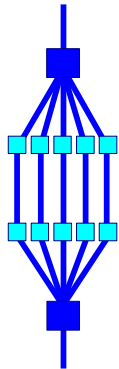
main -- MP doall from line 62 [

Access 2

Type: Read

main -- MP doall from line 62 [

Thread Analyzer GUI - Sources



File View Timeline Help

Find Text:

Races Race Source Experiments

Race Accesses	Source File: /home/ruudp/demo/races-data-dependence/rac
0	63. for (i=0; i<n-1; i++)
14	64. a[i] = a[i+1] + b[i];
	65.
	66. /*

Source File: /home/ruudp/demo/races-data-dependence/rac
Object File: ./race-condition.exe
Load Object: <race-condition.exe>

Race Accesses

Race Accesses	Source File: /home/ruudp/demo/races-data-dependence/rac
0	63. for (i=0; i<n-1; i++)
14	64. a[i] = a[i+1] + b[i];
	65.
	66. /*

Source File: /home/ruudp/demo/races-data-dependence/rac
Object File: ./race-condition.exe
Load Object: <race-condition.exe>

Summary Race Details

Data for Selected Race

Id: Race #1

Vaddr: (Multiple Addresses)

Access 1

Type: Write

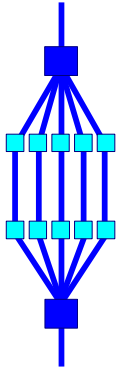
main -- MP doall from line 62 [

Access 2

Type: Read

main -- MP doall from line 62 [

About Sun Studio Thread Analyzer



□ *Free download of Sun Studio 12 Express*

<http://developers.sun.com/sunstudio/downloads/express.jsp>

□ *Thread Analyzer information can be found at:*

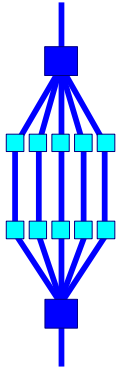
<http://developers.sun.com/prodtech/cc/downloads>

- *Check the web page for updates*
- *Read the tutorial*
- *Read the FAQ*

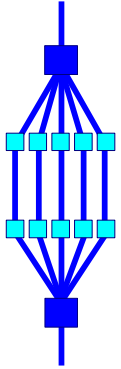
□ *Provide feedback and ask questions on the Sun Studio Tools Forum*

<http://forum.sun.com/jive/forum.jspa?forumID=309>

Avoiding Data Races

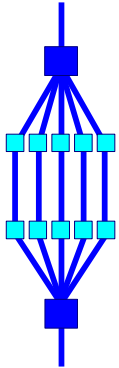


- ❑ **Rule #1** - *Avoid a simultaneous update of shared data*
- ❑ **Rule #2** - *Make sure the data sharing attributes (e.g. private, shared, etc) are correct*
 - *Consider using Sun's autoscoping to assist you*
- ❑ **Rule #3** - *Use the Sun Thread Analyzer*
- ❑ **OpenMP provides several constructs to help:**
 - **Critical Section** - *Only one thread at a time*
 - **Explicit Barrier** - *All threads wait for the last one*
 - **Atomic Construct** - *Lightweight critical section*
 - **Single Region** - *Only one thread; has implied barrier*

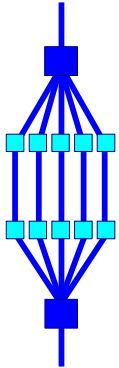


Lab code OMPlab

Sun lab code for OMPlab



- ❑ *We encourage people to work on their own code*
- ❑ *For those that prefer to start with a supplied lab code, we have prepared a simple program for you to work on*
- ❑ *Please use and read the lab description*
- ❑ *In case of questions, please ask one of the Sun people for assistance*
 - *Feel free to also explore features not covered in the hand out*



Thank You !

Ruud van der Pas
ruud.vanderpas@sun.com

