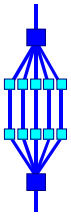


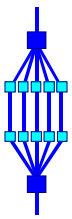
Sun Tools for OpenMP Programming

Ruud van der Pas

Senior Staff Engineer
Compilers, Libraries and
Performance Technologies
Sun Microsystems, Menlo Park, CA, USA

PARCO 2007 Tutorial
RWTH Aachen University
September 3, 2007

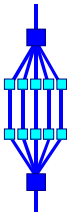
Outline



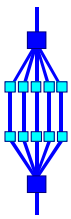
- ***Automatic Parallelization***
- ***Sun Compiler Support for OpenMP***
  - *Compiler options*
  - *Compiler Commentary*
  - *Sun specific environment variables*
  - *Autoscoping*
- ***OpenMP Support in the Sun Performance Analyzer***
- ***The Sun Studio Thread Analyzer***
- ***Appendix***
  - *Compiler options for serial performance*
  - *Pointers to more information*

The new Sun Studio 12 IDE supports OpenMP as well; it is not covered here

Sun Studio Compilers and Tools

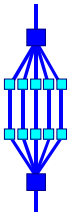


- ***Fortran (f95), C (cc) and C++ (CC) compilers***
  - ***Support sequential optimization, automatic parallelization and OpenMP***
- ***Sun Performance Analyzer***
  - ***Languages supported:*** ***Fortran, C, C++ and Java***
  - ***Parallel:*** ***POSIX Threads, AutoPar, OpenMP and MPI***
  - ***OMPlab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



Automatic Parallelization

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];
```

-xautopar option

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

Thread 1

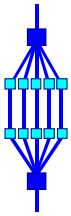
```
for (i=n/P;i<2*n/P;i++)
  a[i] = b[i] + c[i];
```

Thread 2

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

Thread P

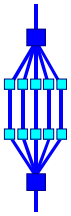
Options For Automatic Parallelization



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)

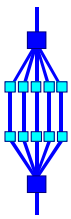
Loop Versioning Example



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

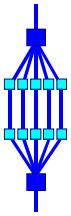
```
     1 void sub1(int n, double a, double *x, double *y)
     2 {
     3    int i;
     4    for (i=0; i<n; i++)
     5      x[i] += a*y[i];
     6 }

"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*



Sun Compiler Support for OpenMP

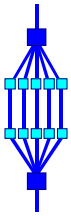
OpenMP Compiler Options



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)

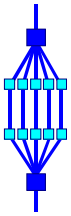
**) The compiler does not raise the optimization level if it is lower than -xO3*

Related Compiler Options



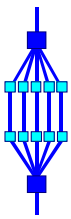
Option	Description
-xloopinfo	Display parallelization messages on screen
-stackvar	Allocate local data on the stack (Fortran only) Use this when calling functions in parallel Included with -xopenmp=parallel | noopt
-vpara/-xvpara	Reports OpenMP scoping errors in case of incorrect parallelization (Fortran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

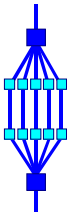


- *Get information about the optimizations performed:*
  - *Loop transformations and parallelization (iropt)*
  - *Instruction scheduling (cg, SPARC only currently)*
- *How to get these messages:*
  - *Add* ***-g*** *to the other compiler options you use*
  - *Example:* ***% cc -c -fast -g funcA.c***
- *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

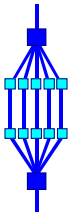


```
% cc -c -g -fast -xopenmp mxv.c
% er_src -scc parallel -src mxv.c mxv.o

   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. }
```



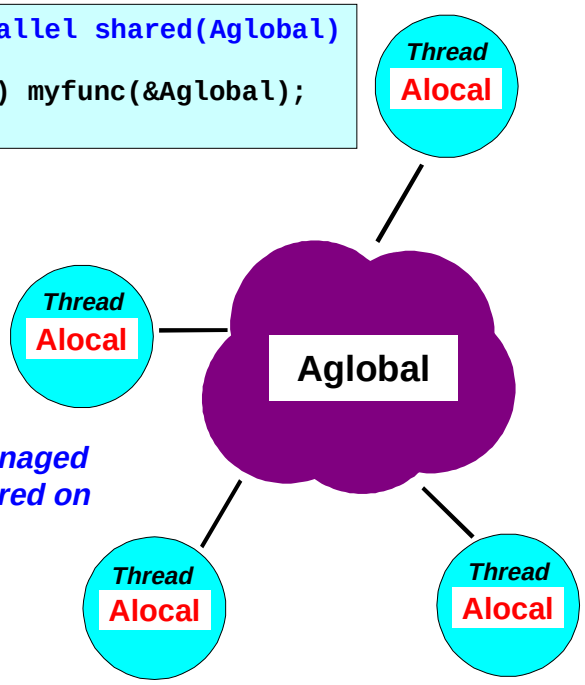
Sun-specific OpenMP Environment Variables

About the stack



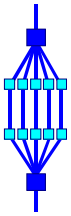
```
#omp parallel shared(Aglobal)
{
   (void) myfunc(&Aglobal);
}
```

```
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

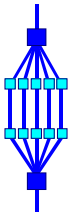


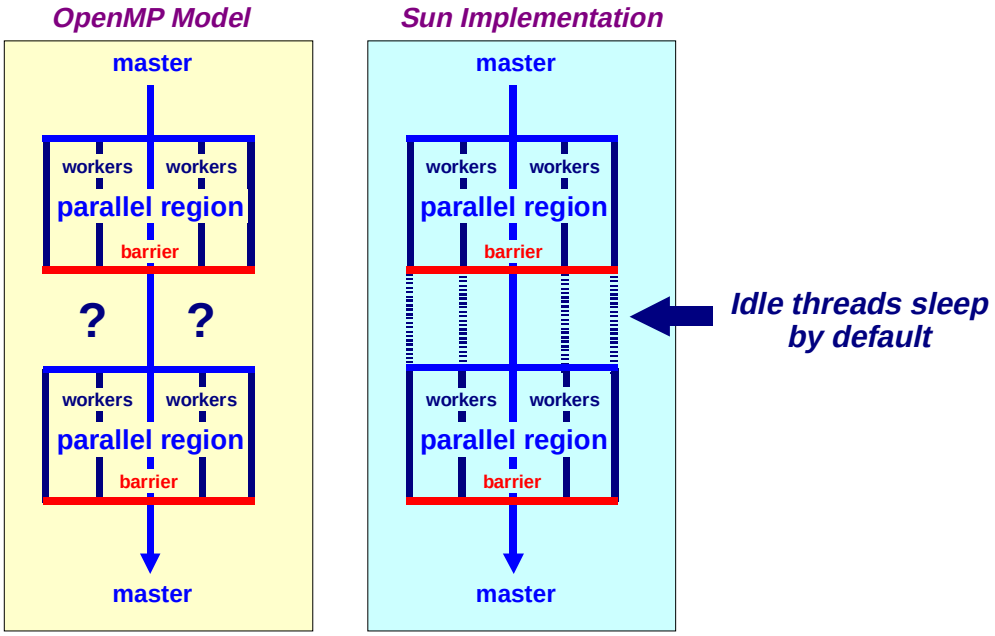
Set thread stack size in n Byte, KB, MB, or GB
Default is KByte

`STACKSIZE n [B,K,M,G]`

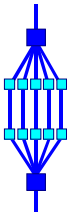
- *Each thread has its own private stack space*
- *If a thread runs out of this stack space, your program crashes with a segmentation violation*
- *Use the Unix "limit/ulimit" command to increase the MAIN ("initial" thread) stack size*
- *Use the **STACKSIZE** environment variable to increase the stack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





The behavior of idle threads

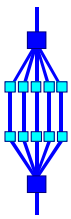


Environment variable to control the behavior:

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

- ***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

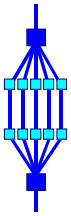
Run-time warnings



```
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



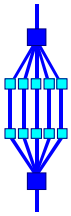
Using more threads than processors:

```
# SUNW_MP_WARN=TRUE; export SUNW_MP_WARN
# OMP_NUM_THREADS=3; export OMP_NUM_THREADS
# ./omp.exe
WARNING (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



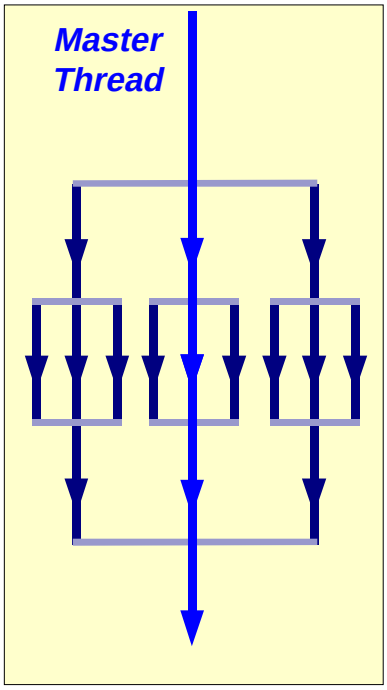
```
20      void foo()
21      {
22        #pragma omp barrier
23           whatever();
24      }
25
26      void bar(int n)
27      {
28        printf("In bar: n = %d\n",n);
29        #pragma omp parallel for
30        for (int i=0; i<n; i++)
31           foo();
32      }
33
34      void whatever()
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

Nested Parallelism



3-way parallel

9-way parallel

3-way parallel

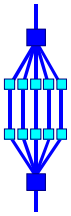
Outer parallel region

Nested parallel region

Outer parallel region

Note: nesting level can be arbitrarily deep

Nested Parallelism on Sun



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

```
SUNW_MP_MAX_POOL_THREADS    <n>
```

Default is 1023

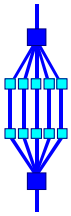
Control the maximum depth of nested active parallel regions

```
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

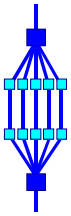


Control binding of threads to "processors"

```
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
```

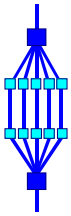
Fragment of psrinfo output

Logical ID

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 starts at processor 0)

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

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

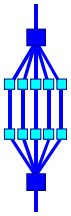
Bind threads to processor 5, 6, 7,, ..., 0, 1, 2

```
% setenv SUNW_MP_PROCBIND 5
```

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

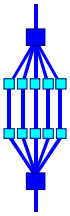
```
% setenv SUNW_MP_PROCBIND “0 24 1 25 2 26”
```

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



Autoscoping

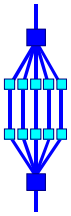
Autoscoping example



Autoscoping 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
```

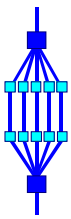
Autoscoping results



```
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

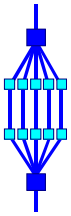
Autoscoping in C



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

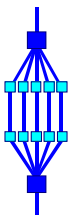
```
Source OpenMP region below has tag R1
Variables autoscoped as SHARED in R1: a, b, n
Private variables in R1: i
Shared variables in R1: n, a, b
 4. #pragma omp parallel for default(__auto)

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



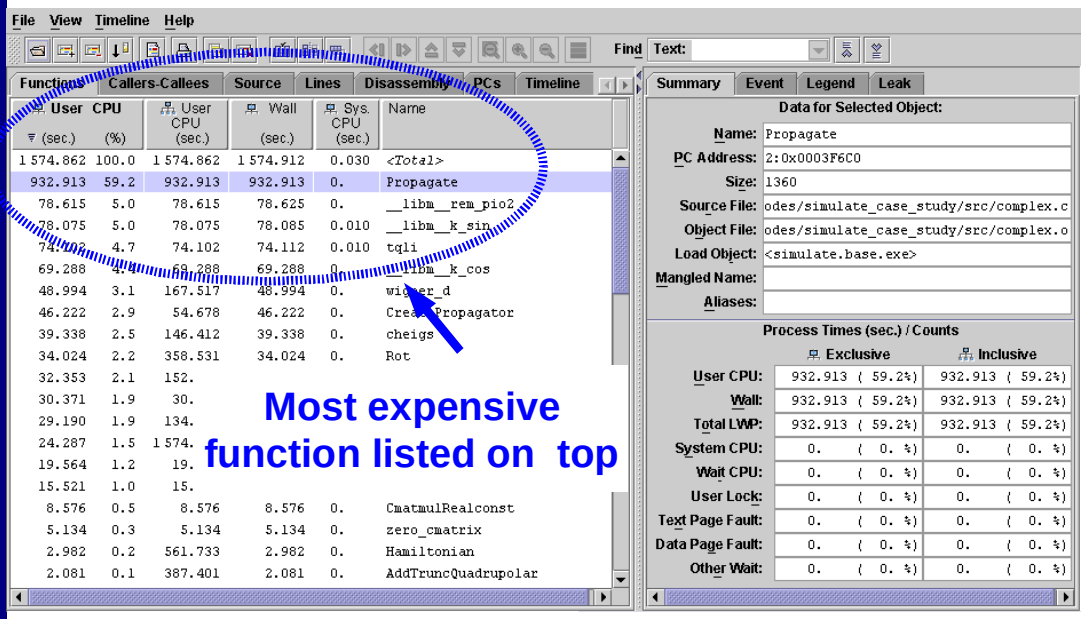
OpenMP Support in the Sun Performance Analyzer

How To Use The Sun Analyzer ?

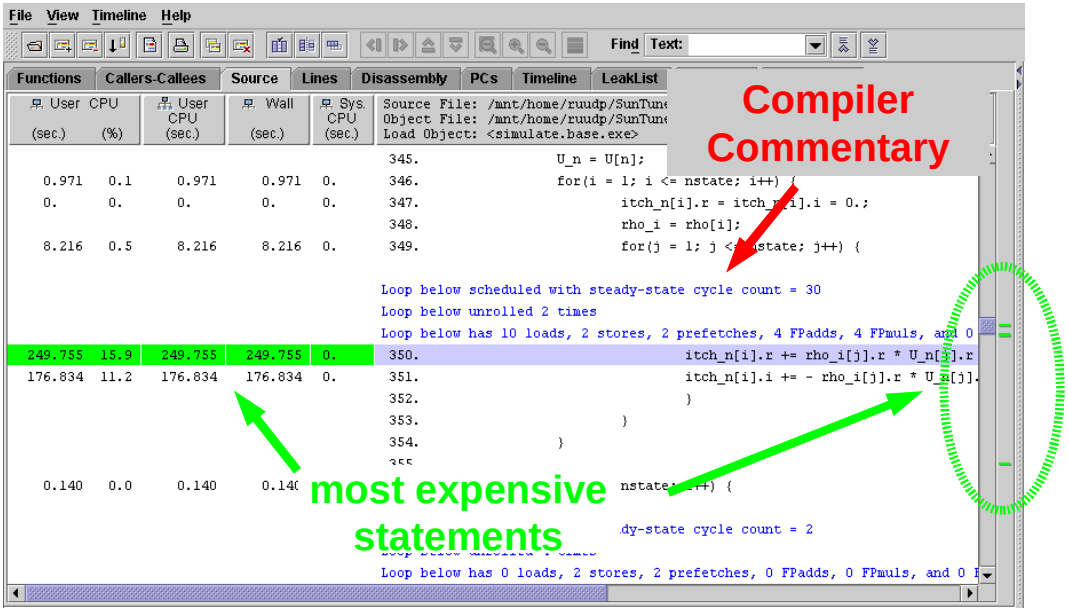


- ***For the most complete information:*** ***recompile with -g***
- ***Two step approach:***
  - ***Step 1 - Collect the data:***
    - % ***collect command - CLI to collect the data***
    - % ***analyzer command - GUI to collect the data***
  - ***Step 2 - Display and analyze the data***
    - % ***analyzer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

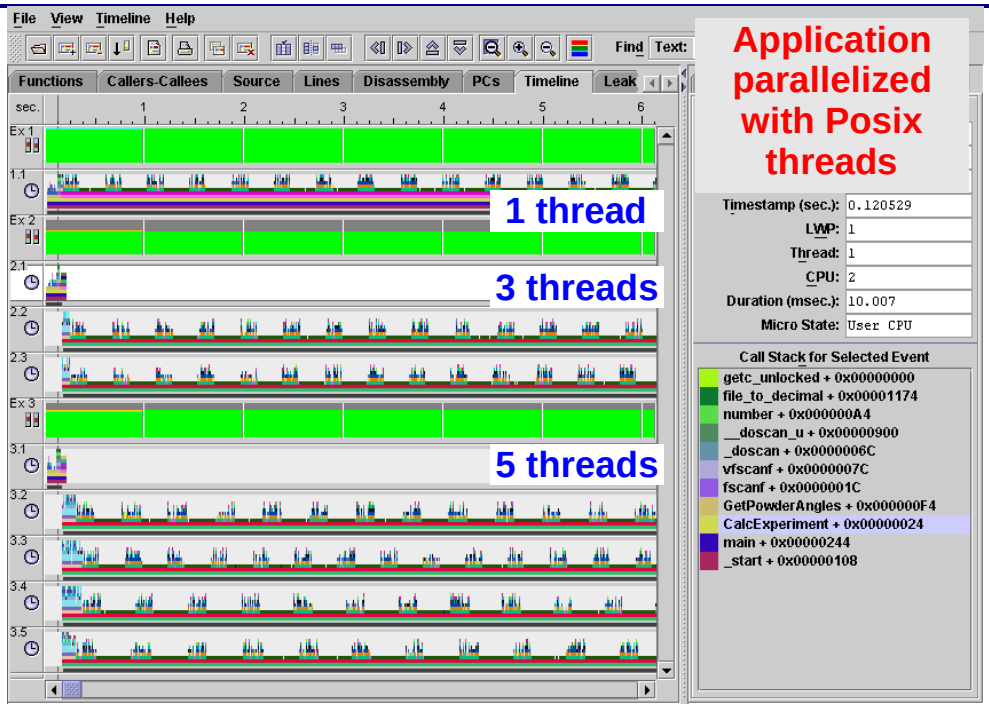
Main Window



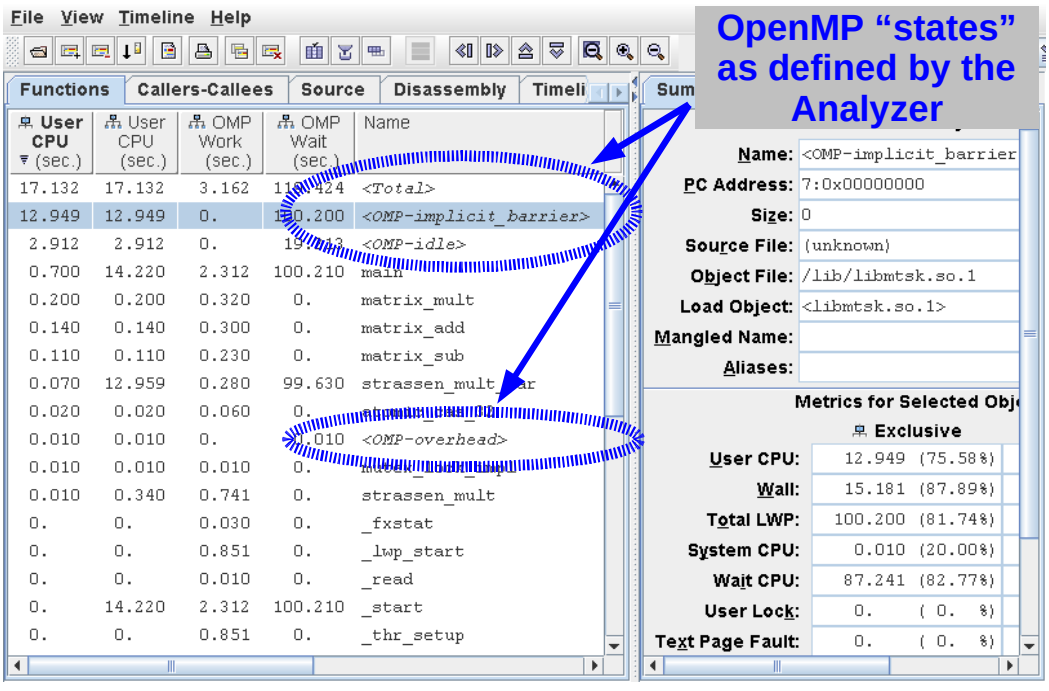
Source Window



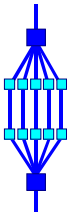
Timeline - Parallel Applications

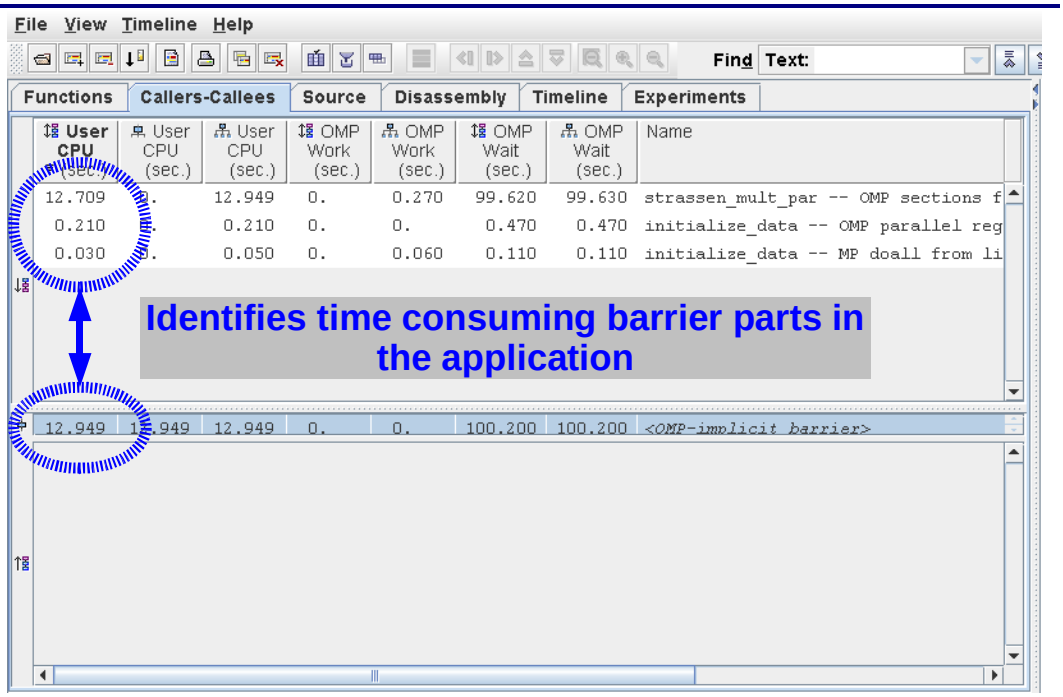


Analyzer Support - OpenMP States

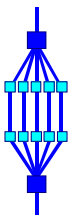


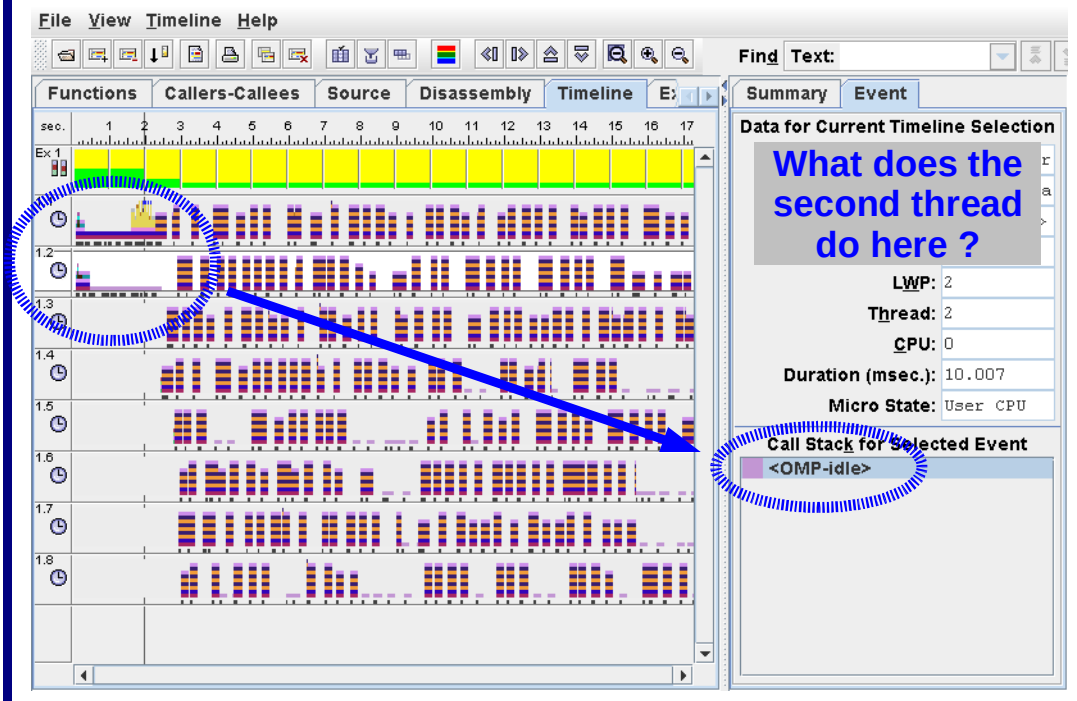
Analyzer - Callers/Callees

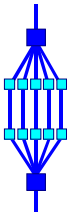




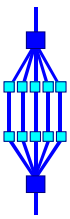
Analyzer - Timeline OpenMP States





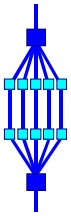


Sun Performance Analyzer Demo



The Sun Studio Thread Analyzer

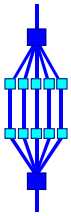
An example of a data race/1



```
#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



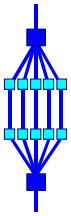
```
#pragma omp parallel default(none) private(i,k,s) \
        shared(n,m,a,b,c,d,dr)
{
   #pragma omp for
   for (i=0; i<m; i++)
   {
```

```
% 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 --*/
```

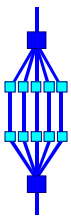
Here is the data race !

Sun Studio Thread Analyzer



- *New tool from Sun - Available in Sun Studio 12*
- *Detects threading errors in a multi-threaded program*
  - *Data race and Deadlock detection*
- *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*

About Sun Studio Thread Analyzer



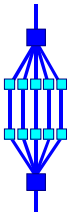
- *Getting Started:*

  http://developers.sun.com/sunstudio/downloads/ssx/tha/tha_getting_started.html

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

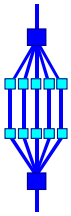
  http://developers.sun.com/sunstudio/community/forums/index.jsp

  Go to "Sun Studio Tools Forum"



Sun Studio Thread Analyzer Demo

Using the Studio 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
```

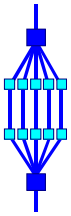
(Command-line interface)

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

(Customized Analyzer GUI)

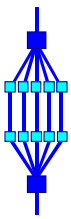
**) Executable will run slower because of instrumentation*

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 of a Data Race



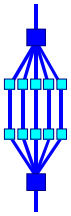
```
#pragma omp parallel shared(n)
   {
#pragma omp single
     {printf("Number of threads: %d\n",omp_get_num_threads());}

     n = omp_get_thread_num();

     printf("Hello Data Race World n = %d\n",n);
   } /*-- End of parallel region --*/
```

The output is correct:

```
Number of threads: 4
Hello Data Race World n = 3
Hello Data Race World n = 2
Hello Data Race World n = 1
Hello Data Race World n = 0
```

Let's see what the Thread Analyzer says:

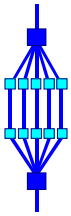
Example command line output

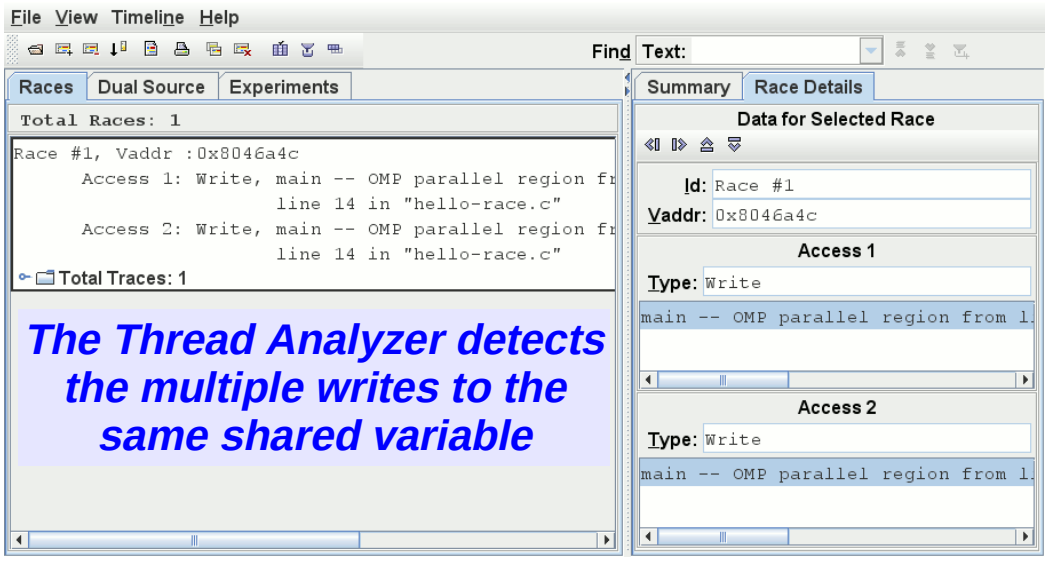


```
Total Races:  1 Experiment:  race.er

Race #1, Vaddr: 0x8046a4c
      Access 1: Write, main -- OMP parallel region from
                line 9 [_$p1A9.main] + 0x000000B9,
                       line 14 in "hello-race.c"
      Access 2: Write, main -- OMP parallel region from
                       line 9 [_$p1A9.main] + 0x000000B9,
                       line 14 in "hello-race.c"
  Total Traces: 1
```

The Thread Analyzer detects the multiple writes to the same shared variable

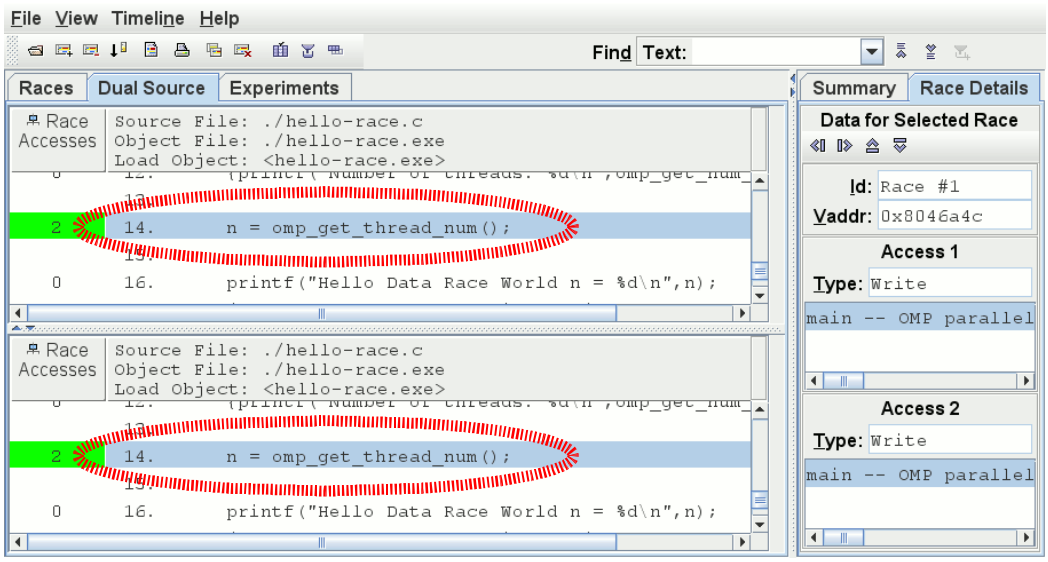
Thread Analyzer GUI - Races





The Thread Analyzer detects the multiple writes to the same shared variable

Thread Analyzer GUI - Sources

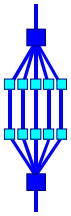


The source lines of the conflicting writes are shown in the "Dual Source" tab

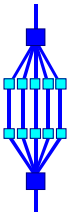
Avoiding Data Races

- ***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***

Summary

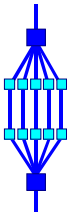


- *Do not forget to try Automatic Parallelization*
- *Sun provides a state-of-the-art OpenMP 2.5 implementation*
  - *Fortran, C and C++ are all supported*
- *The Sun Performance Analyzer is very helpful to identify performance bottlenecks*
- *The Sun Studio Thread Analyzer is a great tool to assist with the detection of data races and deadlock*



Thank You !

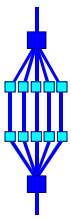
Ruud van der Pas
ruud.vanderpas@sun.com



Appendix A

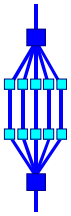
Compiler Options for Serial Performance

Serial 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

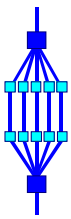


Option	Description	f95	cc	CC	Comp	Link
-xinline	Controls inlining	av.	av.	av.	+	-
-xipo	Interprocedural analysis	av.	av.	av.	+	+
-xprofile	Profile feedback	av.	av.	av.	+	+
-xprefetch	Prefetch on/off	av.	av.	av.	+	-
-xprefetch_level	Controls prefetch algorithm	av.	av.	av.	+	-
-xprefetch_auto_type	Prefetch 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

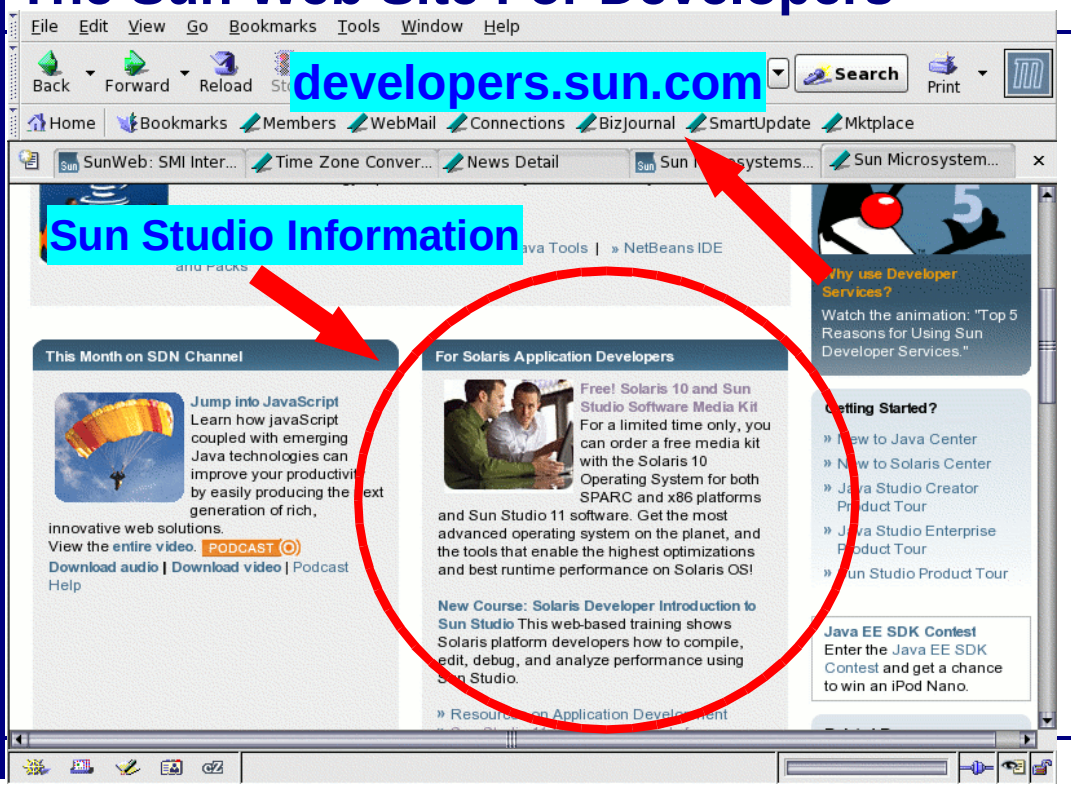
Note: -xvector is implied with -fast on SPARC, but not on AMD
The C compiler on AMD does not set -xprefetch=yes with -fast



Appendix B

Pointers To More Information

The Sun Web Site For Developers



Sun Studio Compilers and Tools

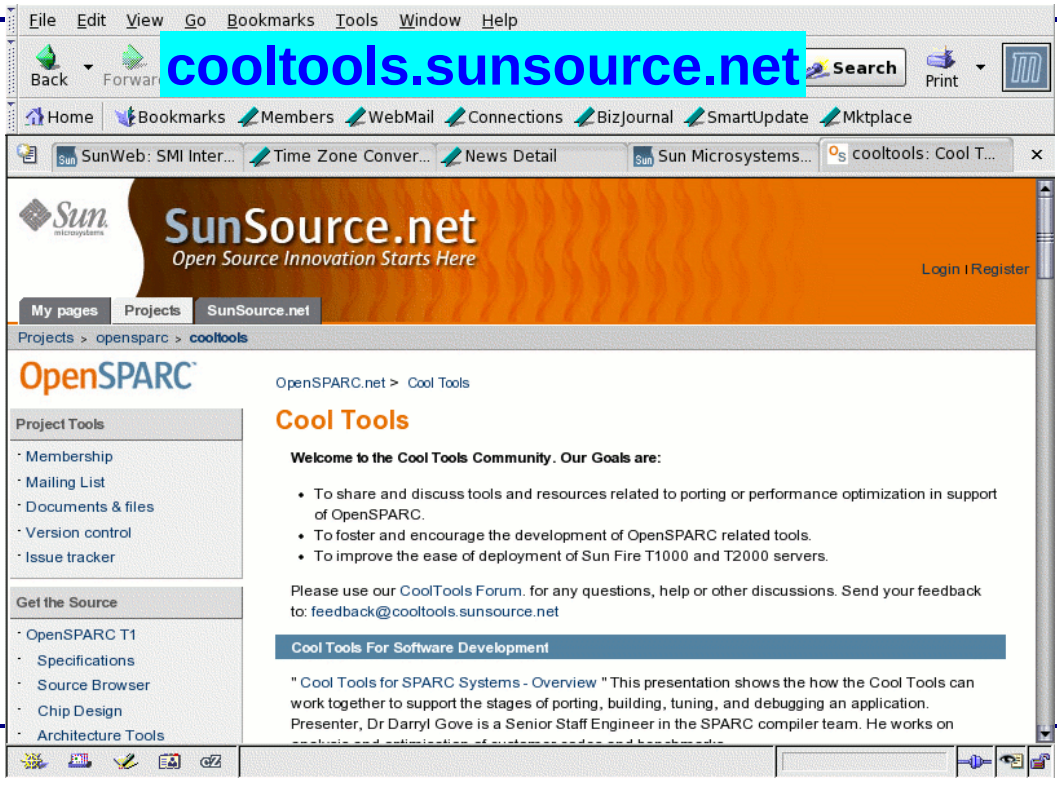
OpenSPARC

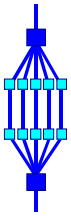
www.opensparc.net

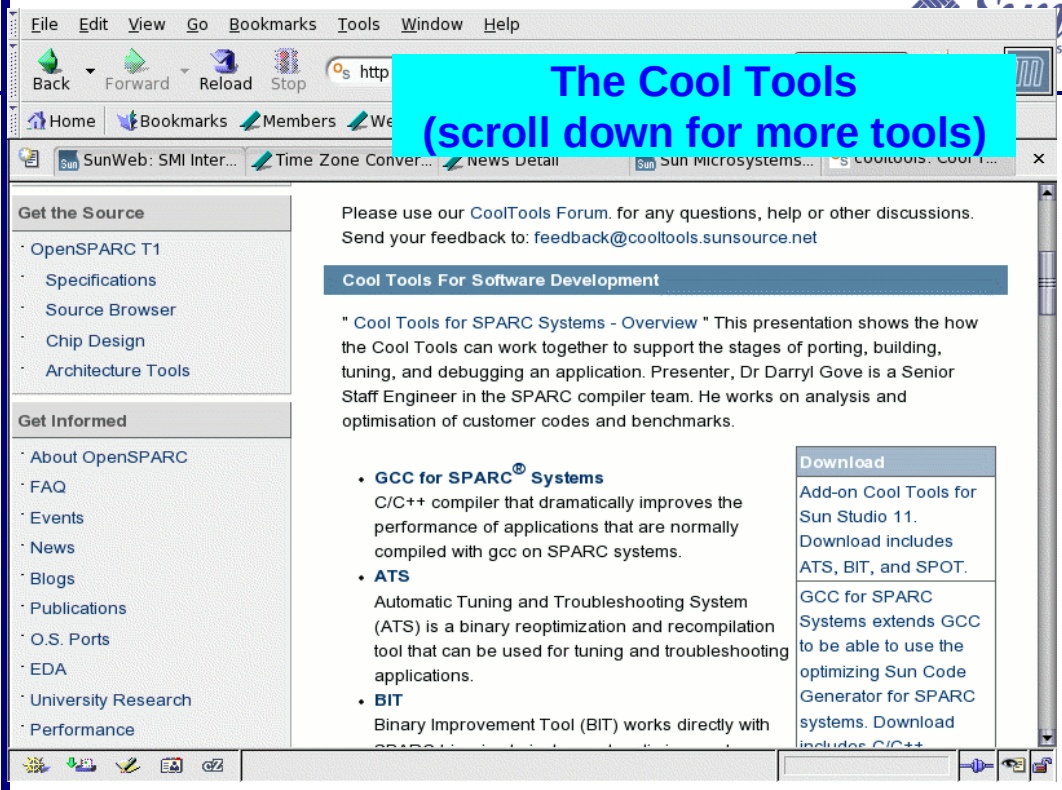
Cool Tools

Cool Tools from Sun

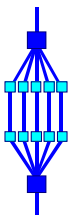
cooltools.sunsource.net



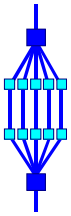




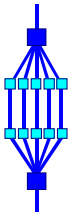
Cool Tools from Sun



- ***GCC for SPARC® systems - C/C++ compiler that dramatically improves the performance of applications that are normally compiled with gcc on SPARC systems***
- ***ATS - The Automatic Tuning and Troubleshooting System (ATS) is a binary reoptimization and recompilation tool that can be used for tuning and troubleshooting applications***
- ***SPOT - Performance optimization tool that produces a report on the performance of an application. The report contains detailed information about various common conditions that impact performance***
- ***BIT - Binary Improvement Tool (BIT) that works directly with SPARC binaries to instrument, optimize, and analyze them for performance or code coverage***
- ***..... for more check out http://opensparc.net***



AMD Developer Web Site



Pointers To More Information/1

Sun Studio Compilers and Tools
http:/developers.sun.com/sunstudio

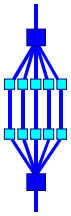
Developer Portal
http:/developers.sun.com

UltraSPARC microprocessor information and manuals
http://www.sun.com/processors

OpenSPARC web page
http://www.opensparc.net

Cool Tools for SPARC
http://cooltools.sunsource.net

Pointers To More Information/2



Using the Sun Compilers:
http://developers.sun.com/prodtech/cc/articles/options.html

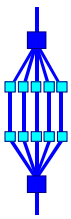
Overview of OpenMP
http://developers.sun.com/sunstudio/articles/omp-intro.html
http://developers.sun.com/sunstudio/articles/studio_openmp.html

Improving performance by using profile feedback and mapfiles
http://developers.sun.com/sunstudio/articles/codelayout.html

Use the Sun MediaLib library to improve performance
http://www.sun.com/processors/vis/mlib.html

Use the Binary Optimizer to further optimize the binary (SPARC)
http://developers.sun.com/sunstudio/articles/binopt.html

Pointers To More Information/3



Introduction on the memory hierarchy:
http://www.sun.com/solutions/blueprints/1102/817-0742.pdf

More information on Solaris and 64-bit:
The "Solaris 64-bit Developer's Guide", part no 806-6543-10
(can be downloaded from http://docs.sun.com)

The AMD64 ABI in the Sun Studio Compilers:
http://developers.sun.com/sunstudio/articles/about_amd64_abi.html

AMD related information
http://www.amd.com

AMD developer documentation
http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_739_9003,00.html