Using OpenMP to Parallelize Interval Algorithms

Ruud van der Pas
Senior Staff Engineer
Technical Developer Tools
Sun Microsystems, Menlo Park, CA, USA

SCAN 2008
The University of Texas at El Paso
El Paso, TX, USA
Sep 29-Oct 3, 2008
Goals Of This Talk

*Present the OpenMP Parallel Programming Model as a possible solution to speed up interval algorithms that require a significant time to compute*

*Demonstrate that Interval Algorithms are not exempt from Data Races*
Outline

- **Interval Arithmetic in the Sun Studio Compilers**
- **The OpenMP Programming Model**
  - Includes a short demo
- **Data Races**
- **Extensive demo**
  - An interval program
    - Written in Fortran
    - Parallelized with OpenMP
  - Thread Analyzer - Detects data races (and deadlock)
- **Wrap Up**
The Sun Studio™ Compilers and Tools

For Free
Sun Studio Compilers and Tools

- **Fortran (f95), C (cc) and C++ (CC) compilers**
  - Support sequential optimization, automatic parallelization and OpenMP

- **The Sun Studio Performance Analyzer**
  - **Languages supported:** Fortran, C, C++ and Java
  - **Parallel:** AutoPar, OpenMP, POSIX/Solaris Threads, MPI

- **The Sun Studio Thread Analyzer**
  - **Languages supported:** C, C++ and Fortran
  - **Parallel:** OpenMP, POSIX/Solaris Threads

- **Sun Studio Integrated Development Environment**

- **Additional tools**
Supported Platforms

- The Sun Studio compilers and tools are supported on various AMD and Intel processors, as well as all SPARC processors
  - SPARC has the siam instruction to better support interval arithmetic
- Operating Systems supported
  - Solaris
  - Certain Linux implementations (RedHat, Suse)
- Regarding Interval Arithmetic
  - Fortran has the best and easiest support
    - Intervals are a built in, native, data type
  - C++ support is through a class library
Using OpenMP to Parallelize Interval Algorithms

http://developers.sun.com/sunstudio

Sun Studio Topics

THE RIGHT TOOLS
Make all the difference. Choose the best Sun Studio today.

Overview  Features  Documentation  Community  Support  Downloads

At a Glance  What's New  Topics  Product Tour  Heroes  Participate

http://developers.sun.com/sunstudio

Edition 9/07
Get the best OS platform for Java, Web 2.0, and C/C++/Fortran developers.
» Free DVD
» Download now
» Learn More

C/C++/Fortran 95 Compilers
The Sun C, C++, and Fortran compilers include advanced features for developing applications on Sun Solaris SPARC and x86/x64 platforms. They utilize a common optimizing backend code generator, and accept standard C, C++, and Fortran with extensions.

The Sun Studio Performance Tools
The Sun Studio performance tools are designed to help answer questions about application performance. This article discusses the kinds of performance questions that users typically ask.

Debugging
Successful program debugging is more an art than a science. dbx is an interactive, source-level, post-mortem and real-time command-line debugging

Sun Developer Network (SDN)
APIs  Downloads  Products  Support  Training  Participate

> search tips

Sun Studio

downloads
Sun Studio Support for Interval Arithmetic
Intervals in Fortran - Key Features

- **Native Interval Data Type**
- **Fortran Intrinsic Functions** (e.g. EXP, LOG, SIN, ...)
- **Interval Specific Intrinsic (Set) Functions**
  - *width, midpoint, hull, union, subset, element of, ...*
- **Order Relations** (e.g. “certainly less than”)
- **Input/Output can be handled in different ways**
- **Integer Power understands Dependence**
- **Mixed mode interval expressions**
- **Context dependent literal interval constants**
Support in C++

- Implemented as class library
- SPARC only
- Same functionality as Fortran

- No mixed mode support because of C++ language standard and not a native data type
Assume that $[a,b]$ and $[c,d]$ are intervals.

For a basic operator "op" in $\{+,-,\times,\div\}$ we can then define:

$$[a,b] \ "op" \ [c,d] \supseteq \{x "op" y | x \in [a,b] \text{ and } y \in [c,d]\}$$

Formulas for basic operations:

$$[a,b] + [c,d] = [a+c, b+d]$$
$$[a,b] - [c,d] = [a-d, b-c]$$
$$[a,b] \times [c,d] = [\min(a*c, a*d, b*c, b*d), \max(a*c, a*d, b*c, b*d)]$$
$$[a,b] \div [c,d] = [\min(a/c, a/d, b/c, b/d), \max(a/c, a/d, b/c, b/d)]$$

(if 0 is not included in $[c,d]$)
Support For Intrinsic Functions

All Fortran intrinsic functions have an interval counterpart if they either return a REAL, or accept a REAL type argument

```
% cat -n cos.f95
1  program demo
2
3  print *, 'cos (-0.5) = ', cos(-0.5D0)
4  print *, 'cos (+0.5) = ', cos(+0.5D0)
5  print *, 'cos [-0.5,+0.5] = ', cos([-0.5,+0.5])
6
7  stop
8  end

% f95 -o cos -xia cos.f95
% ./cos
  cos (-0.5) = 0.8775825618903728
  cos (+0.5) = 0.8775825618903728
  cos [-0.5,+0.5] = [0.87758256189037264,1.0]
```
Integer Powers

The Dependence Problem:

\([-1,2] \times [-1,2] = [-2,4]\]

The Sun Compiler will do the right thing:

```bash
% f95 -o pow -xia pow.f95
% ./pow

X    = [ -1.00000000,  2.00000000]
X*X  = [ -2.00000000,  4.00000000]
X**2 = [  0.00000000,  4.00000000]
```
Order Relations - What To Do?

\[ [a,b] \text{ certainly less than } [c,d] \]

\[ [a,b] \text{ possibly less than } [c,d] \]

Implementation in the Sun compiler:
One of \{C, P, S\}, followed by LT/LE/EQ/NE/GE/GT

Example: A .CLT. B
## Set-Theoretic Interval Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Math. Notation</th>
<th>Fortran</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval hull</td>
<td>$X \cup Y$</td>
<td>X .IH. Y</td>
<td>Interval</td>
</tr>
<tr>
<td>Intersection</td>
<td>$X \cap Y$</td>
<td>X .IX. Y</td>
<td>Interval</td>
</tr>
<tr>
<td>Disjoint</td>
<td>$X \cap Y = \emptyset$</td>
<td>X .DJ. Y</td>
<td>Logical</td>
</tr>
<tr>
<td>Element</td>
<td>$r \in Y$</td>
<td>R .IN. Y</td>
<td>Logical</td>
</tr>
<tr>
<td>Interior</td>
<td>$X &lt; Y$ and $\overline{X} &lt; \overline{Y}$</td>
<td>X .INT. Y</td>
<td>Logical</td>
</tr>
<tr>
<td>Proper subset</td>
<td>$X \subset Y$</td>
<td>X .PSB. Y</td>
<td>Logical</td>
</tr>
<tr>
<td>Proper superset</td>
<td>$X \supset Y$</td>
<td>X .PSP. Y</td>
<td>Logical</td>
</tr>
<tr>
<td>Subset</td>
<td>$X \subseteq Y$</td>
<td>X .SB. Y</td>
<td>Logical</td>
</tr>
<tr>
<td>Superset</td>
<td>$X \supseteq Y$</td>
<td>X .SP. Y</td>
<td>Logical</td>
</tr>
</tbody>
</table>
## Interval Specific Intrinsics

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Name</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infimum</td>
<td>$\inf([a,b]) = a$</td>
<td>INF</td>
<td>REAL</td>
</tr>
<tr>
<td>Supremum</td>
<td>$\sup([a,b]) = b$</td>
<td>SUP</td>
<td>REAL</td>
</tr>
<tr>
<td>Width</td>
<td>$w([a,b]) = b-a$</td>
<td>WID</td>
<td>REAL</td>
</tr>
<tr>
<td>Midpoint</td>
<td>$(a+b) / 2$</td>
<td>MID</td>
<td>REAL</td>
</tr>
<tr>
<td>Magnitude</td>
<td>$\max(</td>
<td>a</td>
<td>,</td>
</tr>
<tr>
<td>Mignitude</td>
<td>$\min(</td>
<td>a</td>
<td>,</td>
</tr>
<tr>
<td>Empty Test</td>
<td>TRUE if empty</td>
<td>ISEMP</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>Number Of Digits</td>
<td>Max. digits</td>
<td>NDIGITS</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>

*) Returns 0 if 0 $\in [a,b]$
Additional Features

- A closed interval system in which all expressions (including singularities and indeterminate forms) are defined
  - Examples: $1/0$, $x^y$ with $x=y=0$, operations involving $+\infty$ and/or $-\infty$
- Domain constraints on intrinsic functions are gracefully handled
  - Example: $\text{SQRT}( [ -1 , +1 ] ) = [ 0 , 1 ]$
- Input / Output can be handled in different ways
- Context dependent literal interval constants
- Mixed mode expressions
Example Code

Program Demo
logical :: not_done = .true.
interval(kind=8) :: ai, bi
write(*,*) 'Please give values for A and B'
do while ( not_done )
   read(*,*),ai,bi
   write(*,9010) '+',ai,'+',bi,ai+bi
   write(*,9010) '-',ai,'-',bi,ai-bi
   write(*,9010) '*',ai,'*',bi,ai*bi
   write(*,9010) '/',ai,'/',bi,ai/bi
end do
9000 continue
stop
9010 format(1X,'A',1X,(A),1X,'B =',VF17.4,1X,(A), &
           1X,VF17.4,' = ',VF17.4)
end
Example Closed Interval System

% f95 -xia math.f95
% ./a.out

Please give values for A and B

A + B = [-1.0000, 3.0000] + [ 1.0000, 2.0000] = [ 0.0000, 5.0000]
A - B = [-1.0000, 3.0000] - [ 1.0000, 2.0000] = [-3.0000, 2.0000]
A * B = [-1.0000, 3.0000] * [ 1.0000, 2.0000] = [-2.0000, 6.0000]
A / B = [-1.0000, 3.0000] / [ 1.0000, 2.0000] = [-1.0000, 3.0000]

A + B = [ 1.0000, 2.0000] + [-1.0000, 3.0000] = [ 0.0000, 5.0000]
A - B = [ 1.0000, 2.0000] - [-1.0000, 3.0000] = [-2.0000, 3.0000]
A * B = [ 1.0000, 2.0000] * [-1.0000, 3.0000] = [-2.0000, 6.0000]
A / B = [ 1.0000, 2.0000] / [-1.0000, 3.0000] = [ -Inf, Inf]
Documentation on Interval Arithmetic support

Sun Studio: Numerical Computation

http://developers.sun.com/sunstudio/overview/topics/numerics_index.html

Latest Documentation: Sun Studio 12

Reference Manuals

- **Numerical Computation Guide**
  A complete application programmer's handbook to understanding the data structures and operations made available by hardware, system software, and software libraries that together implement IEEE Standard 754. IEEE Standard 754 makes it easier to write numerical applications. It is a solid, well-thought-out basis for computer arithmetic that advances the art of numerical programming. (November, 2005)

- **Fortran 95 Interval Arithmetic Programming Reference**
  Documents the intrinsic INTERVAL data types in the Sun Fortran 95 compiler (f95). (November, 2005)

- **C++ Interval Arithmetic Programming Reference**
  Documents the C++ interface to the C++ interval arithmetic library provided with the Sun C++ compilers. (November, 2005)

Math Library Release Notes
<table>
<thead>
<tr>
<th>Compiler</th>
<th>Name/Description</th>
<th>Documentation</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>C++ Interval Arithmetic Examples</td>
<td>C++ Interval Arithmetic code example support documents:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A listing of all the code examples in the C++ Interval Programming Reference</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 95</td>
<td>General Fortran 95 Interval Arithmetic Examples</td>
<td>Provided in the README file included in the tar file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A tar file containing the Fortran 95 interval arithmetic examples included in the examples directory of the installed product</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran 95</td>
<td>Fortran 95 Interval Arithmetic Examples</td>
<td>Fortran 95 Interval Arithmetic code example support documents:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A listing of all the code examples in the Fortran 95 Interval Arithmetic Programming Reference.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://developers.sun.com/sunstudio/documentation/codesamples/index.jsp
Pointers To More Information

- **Documentation**
  - Fortran Interval Arithmetic Programming Reference
    - [http://docs.sun.com/app/docs/doc/819-5271](http://docs.sun.com/app/docs/doc/819-5271)
  - C++ Interval Arithmetic Programming Reference
    - [http://docs.sun.com/app/docs/doc/819-5272](http://docs.sun.com/app/docs/doc/819-5272)

- More information, plus code examples, can be downloaded from [http://developers.sun.com/sunstudio](http://developers.sun.com/sunstudio)

- Another useful web site (on numerical computations):
  - [http://developers.sun.com/sunstudio/overview/topics/numerics_index.html](http://developers.sun.com/sunstudio/overview/topics/numerics_index.html)
Summary Interval Support

- The Sun Fortran and C++ compilers support Interval Arithmetic
- The regular Basic Arithmetic Operations, intrinsic functions and logical operations have been extended to intervals
- In addition to this, several quality of implementation features are supported:
  - Closed interval system, domain constraints on intrinsic functions, input/output, ontext dependent literal interval constants, etc.
- We believe that this provides for a production quality interval compiler
About Parallelization
Why Parallelization?

Parallelization is another optimization technique. The goal is to reduce the execution time.

To this end, multiple processors, or cores, are used.

Using 4 cores, the execution time is 1/4 of the single core time.
What Is Parallelization?

"Something" is parallel if there is a certain level of independence in the order of operations.

In other words, it doesn't matter in what order those operations are performed.

- A sequence of machine instructions
- A collection of program statements
- An algorithm
- The problem you’re trying to solve
How To Program A Parallel Computer?

- **The more well-known parallel programming models:**
  - **A Single System ("Shared Memory")**
    - POSIX Threads (standardized, low level)
    - OpenMP (de-facto standard) \(\rightarrow\) today's focus
    - Automatic Parallelization (compiler does it for you)
  - **A Cluster Of Systems ("Distributed Memory")**
    - Sockets (standardized, low level)
    - MPI - Message Passing Interface (de-facto standard)
  - **A Cluster of Shared Memory/Multicore Systems**
    - The best and worse of both worlds
Automatic Parallelization

- Compiler analyzes loop for parallelism to exploit
- Different iterations of the loop executed in parallel
- Same binary used for any number of threads

```c
for (i=0; i<1000; i++)
a[i] = b[i] + c[i];
```

OMP_NUM_THREADS=4

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-249</td>
<td>250-499</td>
<td>500-749</td>
<td>750-999</td>
</tr>
</tbody>
</table>

Automatic Parallelization Example

```c
#define M 30
#define N 40

void mxv (int m,int n,double *a,double *b[],double *c)
{
    double sum;
    for (int i=0; i<m; i++)
    {
        sum = 0.0;
        for (int j=0; j<n; j++)
            sum += b[i][j]*c[j];
        a[i] = sum;
    }
}
```

% cc -c -fast -xrestrict -xautopar -xloopinfo mxv.c
"mxv.c", line 6: PARALLELIZED, and serial version generated
"mxv.c", line 9: not parallelized, unsafe dependence (sum)

```
for (int i=0; i<m; i++)  // parallel loop
{
    sum = 0.0;
    for (int j=0; j<n; j++)
        sum += b[i][j]*c[j];
    a[i] = sum;
}
```
The Shared Memory Model

A Single System
Using OpenMP to Parallelize Interval Algorithms

http://www.openmp.org
What is OpenMP?

- **De-facto standard API for writing shared memory parallel applications in C, C++, and Fortran**

- **Consists of:**
  - Compiler directives
  - Run time routines
  - Environment variables

- **Specification maintained by the OpenMP Architecture Review Board (ARB)**

- **Version 3.0 was released May 2008**
  - First compiler support now appearing
Advantages of OpenMP

- **Good performance and scalability**
  - *If you do it right of course*

- **De-facto and mature standard**
  - *Supported by a large number of compilers*

- **Requires little programming effort**

- **Preserves sequential version of application**

- **Supports incremental parallelization**

- **Maps naturally onto a multicore architecture:**
  - *Lightweight*
  - *Each thread efficiently executed by a hardware thread*
The OpenMP Execution Model

Fork and Join Model

Master Thread

Parallel region

Worker Threads

Synchronization

Parallel region

Worker Threads

Synchronization
Demo

Basic Parallelization with OpenMP
## Components of OpenMP 2.5

### Directives
- Parallel regions
- Work sharing
- Synchronization
- Data-sharing attributes
  - `private`
  - `firstprivate`
  - `lastprivate`
  - `shared`
  - `reduction`
- Orphaning

### Environment variables
- Number of threads
- Scheduling type
- Dynamic thread adjustment
- Nested parallelism

### Runtime environment
- Number of threads
- Thread ID
- Dynamic thread adjustment
- Nested parallelism
- Timers
- API for locking
Learning Curve - Data Scoping

- In the Shared Memory Programming Model one has to think about the use of the variables (“scoping”)
- There are two main types to distinguish
  - **Private**
    - Each thread has a local copy of the variable(s)
    - Variable is “owned” by a thread
    - Other threads will not see changes made
  - **Shared**
    - There is only one instance of the variable(s)
    - Correct updates to such a variable is under control of the developer
OpenMP Example - Matrix Times Vector

```c
#pragma omp parallel for default(none) \ private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++) {
    sum = 0.0;
    for (j=0; j<n; j++)
        sum += b[i][j]*c[j];
    a[i] = sum;
}
```

Thread 0
```
for (i=0,1,2,3,4)
  i = 0
  sum = \sum b[i=0][j]*c[j]
  a[0] = sum
  i = 1
```

Thread 1
```
for (i=5,6,7,8,9)
  i = 5
  sum = \sum b[i=5][j]*c[j]
  a[5] = sum
  i = 6
```

... etc ...

Shameless Plug - “Using OpenMP”

“Using OpenMP”
Portable Shared Memory Parallel Programming

Chapman, Jost, van der Pas

MIT Press, October 2007


List price: 35 $US

All examples available soon!
(also plan to start a forum on www.openmp.org)
Why The Excitement About OpenMP 3.0?

Support for TASKS!

With this new feature, a wide range of applications can now be parallelized.
Tasking Concept in OpenMP 3.0

Developer specifies tasks in application
Run-time system executes tasks

Encountering thread adds task to pool

Threads execute tasks in the pool
Example - A Linked List

```c
my_pointer = listhead;
while(my_pointer) {
    (void) do_independent_work(my_pointer);
    my_pointer = my_pointer->next;
}
```

**Hard to do before OpenMP 3.0:**
First count number of iterations, then convert while loop to for loop
Example - A Linked List With Tasking

```c
my_pointer = listhead;

#pragma omp parallel
{
    #pragma omp single
    {
        while(my_pointer) {
            #pragma omp task firstprivate(my_pointer)
            {
                (void) do_independent_work(my_pointer);
            }
            my_pointer = my_pointer->next ;
        }
    } // End of single - implied barrier
} // End of parallel region - implied barrier
```
Data Races
About Parallelism

- **Parallelism**
- **Independence**
- **No Fixed Ordering**

"Something" that does not obey this rule, is not parallel (at that level)
#pragma omp parallel shared(n)

\{ n = omp_get_thread_num(); \}

OMP_NUM_THREADS=4

```
Thread 0
n = 0
```
```
Thread 1
n = 1
```
```
Thread 2
n = 2
```
```
Thread 3
n = 3
```

Time

```
W
```
```
W
```

```n```
```
W
```
```
W
```

```
0 1 3 4
```
```
M  M  M  M
```
What is a Data Race?

- Two different threads in a multi-threaded shared memory program
- Access the same (=shared) memory location
  - Concurrently
  - Without holding any common exclusive locks
  - At least one of the accesses is a write/store
Using OpenMP to Parallelize Interval Algorithms

A Parallel Loop

```c
for (i=0; i<8; i++)
    a[i] = a[i] + b[i];
```

The result does not depend on the order of execution

---

**Thread 1**
- a[0]=a[0]+b[0]

**Thread 2**
Not A Parallel Loop

```c
for (i=0; i<8; i++)
    a[i] = a[i+1] + b[i];
```

The result is not deterministic if executed in parallel!

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
</table>
### Numerical Results

<table>
<thead>
<tr>
<th>threads: 1 checksum</th>
<th>1953 correct value</th>
<th>1953</th>
</tr>
</thead>
<tbody>
<tr>
<td>threads: 1 checksum</td>
<td>1953 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 1 checksum</td>
<td>1953 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 2 checksum</td>
<td>1953 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 2 checksum</td>
<td>1953 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 2 checksum</td>
<td>1953 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 4 checksum</td>
<td>1905 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 4 checksum</td>
<td>1905 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 4 checksum</td>
<td>1953 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 4 checksum</td>
<td>1937 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 32 checksum</td>
<td>1525 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 32 checksum</td>
<td>1473 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 32 checksum</td>
<td>1489 correct value</td>
<td>1953</td>
</tr>
<tr>
<td>threads: 32 checksum</td>
<td>1513 correct value</td>
<td>1953</td>
</tr>
</tbody>
</table>

**Data Race In Action!**
Demo

Parallelizing An Interval Algorithm Using OpenMP
Bottom Line About Data Races

Data Races Are Easy To Put In
But
Very Hard To Find

“Finding errors in software is particularly important in computer programs that claim to be mathematically rigorous.”

R. Baker Kearfott - SCAN 2008, El Paso, TX

That is why a special tool to find data races is highly recommended to use
Wrap Up

- The Sun Studio Fortran and C++ compilers support Interval Arithmetic
  - Fortran implementation most elegant and powerful
- OpenMP provides for an easy to use, but yet very powerful, portable parallel programming model
  - Also very suitable for multicore architectures
- Despite this, parallel programming can still be tricky
- As always, good tools can make all the difference when it comes to productivity and correctness
That's It

Thank You and ..... Stay Sharp !

Ruud van der Pas
ruud.vanderpas@sun.com