# Verifying an OpenMP Parallelization with the Intel® Thread Checker

Rolf Rabenseifner, Bettina Krammer
**rabenseifner@hlrs.de, krammer@hlrs.de**

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Sep 11, 2006
Used version of Intel® Thread Checker under Linux: 3.0



Overview

## Goal

- To detect race conditions
- and other parallelization errors (e.g., missing **first**private)
- in OpenMP parallel application programs

OpenMP parallelizations should never be used in production without verification with race-condition checking tools (like Assure or Intel® Thread Checker)

**Content**

## Intel® Thread Checker has two modes

- **Thread count independent analysis mode** TCI
  - Projection technology
    executes the sequential version of the application and treats it as the specification for the OpenMP application
  - Source code instrumentation
  - Compares sequential code with maximal parallel execution
  - Can detect many different kinds of errors
  - Can detect far more errors than possible with simulation approach
  - Not applicable if parallel code can not be projected to sequential code
  - → **Used on most slides**
- **Simulation approach** BIN
  - Using binary instrumentation
  - Supported for x86 code on x86 and EM64T platforms
  - Thread-parallel execution

Intel Thread Checker [08a]



## Method

- Compile your OpenMP application with Thread Checker
- Start and execute with Thread Checker
  - → executed on 1 thread
  - → verifying all memory accesses
  - → ~300 times slower than normal execution!!!

  TCI
- Invoke analysis tool
  - → Error report
  - → with references to your source file
- Try to find the parallelization bugs in your application
- Try to correct these parallelization bugs
  - – without modifying the serial semantics of the program
- Compile and execute again
  - → **until all errors are resolved**

## Processing

**Source program**
my_prog.c or my_prog.f

↓

compile with options
–tcheck –openmp –g

↓

**Executable** my_prog

↓

execute with command
tcheck_cl ./a.out

↓

**threadchecker.thr**
and ascii **myprog.txt**

↓

Further analysis with
tcheck_cl –f …
tcheck_cl –s …

Example:

```
module load itt/tcheck-3.0                    (setup)

icc   -tcheck -openmp -g -o my_prog my_prog.c
ifort -tcheck -openmp -g -o my_prog my_prog.f
```

TCI

```
tcheck_cl -w 90 -o my_prog.txt  ./my_prog
```

Text output (130 chars line width):

```
tcheck_cl -f txt -w 130 threadchecker.thr
```

Comma separated output (e.g., as input for Excel):

```
tcheck_cl -f csv threadchecker.thr
```

Call stack of one event id (e.g. 7):

```
tcheck_cl -s 7 threadchecker.thr
```



TCI

## Output – Example with demo_with_bugs.c

| | ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|---|
| Exa.1 | 1 | Write -> Read | Error | 98 | omp for | Memory read of a[ ] at "demo.c":30 conflicts with a prior memory write of a[ ] at "demo.c":29 (flow dependence) | "demo.c" :29 | "demo.c" :30 |
| Exa.2 | 2 | Write -> Read | Error | 197 | omp par-allel reg. | Memory read of a[ ] at "demo.c":54 conflicts with a prior memory write of a[ ] at "demo.c":49 (flow dependence) | "demo.c" :49 | "demo.c" :54 |
| | 3 | Read -> Write | Error | 197 | omp par-allel reg. | Memory write of a[ ] at "demo.c":49 conflicts with a prior memory read of a[ ] at "demo.c":54 (anti dependence) | "demo.c" :54 | "demo.c" :49 |
| Exa.3 | 4 | Write -> Write | Error | 98 | omp for | Memory write of x at "demo.c":71 conflicts with a prior memory write of x at "demo.c":71 (output dependence) | "demo.c" :71 | "demo.c" :71 |
| | 5 | Read -> Write | Error | 98 | omp for | Memory write of x at "demo.c":71 conflicts with a prior memory read of x at "demo.c":72 (anti dependence) | "demo.c" :72 | "demo.c" :71 |
| Exa.4 | 6 | undefined in access | Caution | 1 | omp par-allel reg. | OpenMP -- the access at "demo.c":88 is undefined, the expected value was defined at "demo.c":84 in serial exec. | "demo.c" :84 | "demo.c" :88 |
| | 7 | undefined in the serial c. | Warning | 1 | "demo.c": 17 | OpenMP -- undefined in the serial code (original program) at "demo.c":91 with "demo.c":88 | "demo.c" :88 | "demo.c" :91 |
| Exa.5 | 8 | Read -> Write | Error | 98 | omp for | Memory write of sum at "demo.c":105 conflicts with a prior memory read of sum at "demo.c":105 (anti dependence) | "demo.c" :105 | "demo.c" :105 |
| | 9 | Write -> Read | Error | 98 | omp for | Memory read of sum at "demo.c":105 conflicts with a prior memory write of sum at "demo.c":105 (flow dependence) | "demo.c" :105 | "demo.c" :105 |
| | 10 | Write -> Write | Error | 98 | omp for | Memory write of sum at "demo.c":105 conflicts with a prior memory write of sum at "demo.c":105 (output dependence) | "demo.c" :105 | "demo.c" :105 |
| Exa.6 | 11 | OpenMP -- cannot be private | Caution | 98 | omp for | OpenMP -- The access at "demo.c":136 cannot be private because it expects the value previously defined at "demo.c":136 in the serial execution | "demo.c" :136 | "demo.c" :136 |
| Exa.7 | 12 | OpenMP -- cannot be private | Caution | 99 | omp for | OpenMP -- The access at "demo.c":178 cannot be private because it expects the value previously defined at "demo.c":178 in the serial execution | "demo.c" :178 | "demo.c" :178 |
| | 13 | Thread termination | Informati on | 1 | Whole Program | Thread termination at "demo_with_bugs.c":17 - includes stack allocation of 10485760 and use of 2416 bytes | "demo.c" :17 | "demo.c" :17 |

Seven Examples



## Example 1 – source code

```
25    a[0] = 0;
26  # pragma omp parallel for
27    for (i=1; i<N; i++)
28    {
29      a[i] = 2.0*i*(i-1);
30      b[i] = a[i] - a[i-1];
31    } /* end of omp parallel for */
```

What‘s wrong?

How to solve?



TCI

## Example 1 – analysis

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 1 | **Write -> Read** data-race | Error | 98 | omp for | Memory **read** of **a[ ]** at "demo_with_bugs.c":**30** conflicts with a prior memory **write** of **a[ ]** at "demo_with_bugs.c":**29** (flow dependence) | "demo_with_bugs.c":29 | "demo_with_bugs.c":30 |

```
25    a[0] = 0;
26  # pragma omp parallel for
27    for (i=1; i<N; i++)
28    {
29      a[i] = 2.0*i*(i-1);
30      b[i] = a[i] - a[i-1];
31    } /* end of omp parallel for */
```

Solution

In the printed version, the solutions can be found in the appendix ;

## Example 2 – source code

```
43    a[0] = 0;
44  # pragma omp parallel
45    {
46  #   pragma omp for nowait
47      for (i=1; i<N; i++)
48      {
49        a[i] = 3.0*i*(i+1);
50      } /* end of omp for nowait */
51  #   pragma omp for
52      for (i=1; i<N; i++)
53      {
54        b[i] = a[i] - a[i-1];
55      } /* end of omp for */
56    } /* end of omp parallel */
```

What's wrong?

How to solve?



TCI

## Example 2 – analysis

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 2 | **Write -> Read** data-race | Error | 197 | omp parallel region | Memory **read** of **a[ ]** at "demo_with_bugs.c":**54** conflicts with a prior memory **write** of **a[ ]** at "demo_with_bugs.c":**49** (flow dependence) | "demo_with_bugs.c":49 | "demo_with_bugs.c":**54** |
| 3 | **Read -> Write** data-race | Error | 197 | omp parallel region | Memory **write** of **a[ ]** at "demo_with_bugs.c":**49** conflicts with a prior memory **read** of **a[ ]** at "demo_with_bugs.c":**54** (anti dependence) | "demo_with_bugs.c":**54** | "demo_with_bugs.c":**49** |

```
43    a[0] = 0;
44  # pragma omp parallel
45    {
46  #   pragma omp for nowait
47      for (i=1; i<N; i++)
48      {
49        a[i] = 3.0*i*(i+1);
50      } /* end of omp for nowait */
51  #   pragma omp for
52      for (i=1; i<N; i++)
53      {
54        b[i] = a[i] - a[i-1];
55      } /* end of omp for */
56    } /* end of omp parallel */
```

see appendix

## Example 3 – source code

```
68  # pragma omp parallel for
69    for (i=1; i<N; i++)
70    {
71      x = sqrt(b[i]) - 1;
72      a[i] = x*x + 2*x + 1;
73    } /* end of omp parallel for */
```

What's wrong?

How to solve?

Intel Thread Checker [08a]



TCI

## Example 3 – analysis

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 4 | **Write -> Write** data-race | Error | 98 | omp for | Memory **write** of **x** at "demo_with_bugs.c":**71** conflicts with a prior memory **write** of **x** at "demo_with_bugs.c":**71** (output | "demo_with_bugs.c":**71** | "demo_with_bugs.c":**71** |
| 5 | **Read -> Write** data-race | Error | 98 | omp for | Memory write of **x** at "demo_with_bugs.c":**71** conflicts with a prior memory **read** of **x** at "demo_with_bugs.c":**72** (anti dependence) | "demo_with_bugs.c":**72** | "demo_with_bugs.c":**71** |

```
68  # pragma omp parallel for
69    for (i=1; i<N; i++)
70    {
71      x = sqrt(b[i]) - 1;
72      a[i] = x*x + 2*x + 1;
73    } /* end of omp parallel for */
```

see appendix

## Example 4 – source code

```
84    f = 2;
85  # pragma omp parallel for private(f,x)
86    for (i=1; i<N; i++)
87    {
88      x = f * b[i];
89      a[i] = x - 7;
90    } /* end of omp parallel for */
91    a[0] = x;
```

What‘s wrong?

How to solve?



TCI

## Example 4 – analysis

| ID | Short Description | Severity Name | Count | Context [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 6 | OpenMP -- **undefined in access** | Caution | 1 | omp parallel region | OpenMP -- the access at "demo_with_bugs.c":88 is undefined, the expected value was defined at "demo_with_bugs.c":84 in the serial execution | "demo_with_bugs.c":84 | "demo_with_bugs.c":88 |
| 7 | OpenMP -- **undefined in the serial code** (original prog.) | Warning | 1 | "demo_with_bugs.c":17 | OpenMP -- undefined in the serial code (original program) at "demo_with_bugs.c":91 with "demo_with_bugs.c":88 | "demo_with_bugs.c":88 | "demo_with_bugs.c":91 |

```
84    f = 2;
85  # pragma omp parallel for private(f,x)
86    for (i=1; i<N; i++)
87    {
88      x = f * b[i];
89      a[i] = x - 7;
90    } /* end of omp parallel for */
91    a[0] = x;
```

see appendix

## Example 5 – source code

```
101     sum = 0;
102   # pragma omp parallel for
103     for (i=1; i<N; i++)
104     {
105       sum = sum + b[i];
106     } /* end of omp parallel for */
```

What‘s wrong?

How to solve?



TCI

## Example 5 – analysis

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 8 | Read -> Write data-race | Error | 98 | omp for | Memory **write** of **sum** at "demo_with_bugs.c":**105** conflicts with a prior memory **read** of **sum** at "demo_with_bugs.c":**105** (anti dependence) | "demo_with_bugs.c":105 | "demo_with_bugs.c":105 |
| 9 | Write -> Read data-race | Error | 98 | omp for | Memory **read** of **sum** at "demo_with_bugs.c":**105** conflicts with a prior memory **write** of **sum** at "demo_with_bugs.c":**105** (flow dependence) | "demo_with_bugs.c":105 | "demo_with_bugs.c":105 |
| 10 | Write -> Write data-race | Error | 98 | omp for | Memory **write** of **sum** at "demo_with_bugs.c":**105** conflicts with a prior memory **write** of **sum** at "demo_with_bugs.c":**105** (output | "demo_with_bugs.c":105 | "demo_with_bugs.c":105 |

```
101     sum = 0;
102   # pragma omp parallel for
103     for (i=1; i<N; i++)
104     {
105       sum = sum + b[i];
106     } /* end of omp parallel for */
```

see appendix

TCI

## Example 6 – nothing wrong – unexpected "Caution"

```
129     sum = 0;
130   # pragma omp parallel private(psum)
131     {
132       psum = 0;
133   #   pragma omp for
134         for (i=1; i<N; i++)
135         {
136           psum = psum + b[i];
137         } /* end of omp for */
138   #   pragma omp critical
139       {
140         sum = sum + psum;
141       } /* end of omp critical */
142     } /* end of omp parallel */
```

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 11 | OpenMP - - **cannot be private** | Cautio n | 98 | omp for | OpenMP -- The access at "demo_with_bugs.c":136 cannot be private because it expects the value previously defined at "demo_with_bugs.c":136 in the serial execution | "demo_w ith_bugs. c":136 | "demo_w ith_bugs. c":136 |



## Example 7 – source code

From OpenMP Introduction: Parallelization trick to achieve reproducible & efficient reduction results if OMP_NUM_THREADS is fixed

```
      omp_set_dynamic(0);
165   b[0] = 0;
166   sum = 0;
167 # pragma omp parallel private(psum, num_threads)
168   {
169 #   ifdef _OPENMP
170       num_threads=omp_get_num_threads();
171 #   else
172       num_threads=1;
173 #   endif
174     psum = 0;
175 #   pragma omp for schedule(static,(N-1)/num_threads+1)
176       for (i=0; i<N; i++)
177       {
178         psum = psum + b[i];
179       } /* end of omp for */
180 #   pragma omp for ordered schedule(static,1)
181       for (i=0;i<num_threads;i++)
182       {
183 #       pragma omp ordered
184           sum = sum + psum;
185       } /* end of omp for */
186   } /* end of omp parallel */
```

Schedule is fixed

Reduction ordering is fixed

Nothing is wrong. But there are unexpected warnings!

TCI

## Example 7 – analysis

- Three problems
  - Compile time error ➡ see (A) on next slides
  - Runtime error ➡ see (B) on next slides
  - Analysis error: a **caution** although the **code is correct** → may be ignored (same as in Example 6)

```
174      psum = 0;
175 #    pragma omp for schedule(static,(N-1)/num_threads+1)
176        for (i=0; i<N; i++)
177        {
178          psum = psum + b[i];
179        } /* end of omp for */
```

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 12 | OpenMP -- cannot be private | Caution | 99 | omp for | OpenMP -- The access at "demo_with_bugs.c":178 cannot be private because it expects the value previously defined at "demo_with_bugs.c":178 in the serial execution | "demo_with_bugs.c":178 | "demo_with_bugs.c":178 |

Intel Thread Checker [08a]



## Example 7 – analysis – Compile time error (A)

- Compile time error:
  - **Demo_with_bugs.c(167): warning #1378: Variable "num_threads" in OpenMP schedule clause should appear on shared list**
    **# pragma omp parallel private(psum, num_threads)**

```
167 # pragma omp parallel private(psum, num_threads)
168   {
169 #   ifdef _OPENMP
170       num_threads=omp_get_num_threads();
171 #   else
172       num_threads=1;
173 #   endif
      ...
175 #   pragma omp for schedule(static,(N-1)/num_threads+1)
176       for ...
```

- Same value in `num_threads` on all threads, but compiler doesn't know!

## Example 7 – analysis – Compile time error (A) cont'd

- Work-around:
  **num_threads** as **automatic variable**, defined in the parallelized block

```
167 # pragma omp parallel private(psum, num_threads)
168   {int num_threads;
169 #   ifdef _OPENMP
170       num_threads=omp_get_num_threads();
171 #   else
172       num_threads=1;
173 #   endif
      ...
175 #   pragma omp for schedule(static,(N-1)/num_threads+1)
176       for ...
```

- Don't use **shared(num_threads)**
  - because assignment
    **num_threads = omp_get_num_threads();**
    would cause a write-write race condition
  - ➔ cache-line false sharing



## Example 7 – analysis – Compile time error (A) cont'd

- Solution:
  - Calculate **num_threads** with an own parallel region
  - before numerical parallel region (lines 167-186)

```
      int num_threads;
      omp_set_dynamic(0);
    # pragma omp parallel
      {
    #   pragma omp single
        {
    #     ifdef _OPENMP
            num_threads=omp_get_num_threads();
    #     else
            num_threads=1;
    #     endif
        } /* end of omp single */
      } /* end of omp parallel */
...
167 # pragma omp parallel private(psum) shared(num_threads)
168   {
169-173 /* calculation of num_treads is removed */
174     psum = 0;
175 #   pragma omp for schedule(static,(N-1)/num_threads+1)
176       for (i=0; i<N; i++)
177 ...
```

## Example 7 – analysis – Run time error (B)

- Run time error
  - `omp_get_num_threads()` returns 2
  - but only 1 thread is used!
  - sum = sum + psum executed twice → **wrong result**
  - Feature (not a bug) of Intel® Thread Checker

```
180 # pragma omp for ordered \
            schedule(static,1)
181     for (i=0;i<num_threads;i++)
182     {
183 #     pragma omp ordered
184         sum = sum + psum;
185     } /* end of omp for */
```

- Solution:
  - Calculate `num_threads` at the beginning and without OpenMP functions

```
  int num_threads;
  omp_set_dynamic(0);
  num_threads=0;
# pragma omp parallel
  {
#   pragma omp critical
    {
      num_threads++;
    } /* end of omp critical */
  } /* end of omp parallel */
```



## Example 7 – solution to A and B

```
  int num_threads;
  omp_set_dynamic(0);
  num_threads=0;
# pragma omp parallel
  {
#   pragma omp critical
    { num_threads++;
    } /* end of omp critical */
  } /* end of omp parallel */

  b[0] = 0;
  sum = 0;
# pragma omp parallel private(psum)
  { psum = 0;
#   pragma omp for schedule(static,(N-1)/num_threads+1)
      for (i=0; i<N; i++)
      {
        psum = psum + b[i];
      } /* end of omp for */
#   pragma omp for ordered schedule(static,1)
      for (i=0;i<num_threads;i++)
      {
#       pragma omp ordered
          sum = sum + psum;
      } /* end of omp for */
  } /* end of omp parallel */
```

TCI

## Stack info

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 13 | Thread terminatio n | Inform ation | 1 | Whole Progra m 1 | Thread termination at "demo_with_bugs.c":17 - includes stack allocation of 10485760 and use of 2416 bytes | "demo_w ith_bugs. c":17 | "demo_w ith_bugs. c":17 |



Restrictions in TCI mode

## Restrictions on OpenMP library routines in TCI mode

TCI



- **Get functions**

| | returned values outside par.regions | in parallel regions |
|---|---|---|
| omp_get_thread_num() | 0 | 0 |
| omp_get_num_threads() | 1 | 2<br>this value is wrong,<br>• see Example 7<br>• next slides |

TCI

## Restrictions on OpenMP library routines

- **Get functions**

| | returned values outside par.regions | in parallel regions |
|---|---|---|
| omp_in_parallel() | 1 | 0 |
| omp_get_max_threads() | 2 (see next slides) | 2 (see next slides) |
| omp_get_num_procs() | 2 (see next slides) | 2 (see next slides) |
| omp_get_dynamic() | 0 | 0 |
| omp_get_nested() | 0 | 0 |
| omp_get_wtick() | 1.0 | 1.0 |
| omp_get_wtime() | call counter | call counter |

Intel Thread Checker [08a]



TCI

## Number of threads = 2

Reported behavior in "Thread Count Independent Mode"
(TCI = projection mode):

- omp_get_num_threads(), omp_get_max_threads(), and omp_get_num_procs() → return 2
- Internally **only one thread** is used
- But race condition **checking** is done with number of threads = number of loop iterations or sections!
  → maximally parallel !

TCI

## Number of threads = 2 – Reasons

```
  int num_threads;
  omp_set_dynamic(0);
# pragma omp parallel
  {
#   pragma omp single
    {
      num_threads=omp_get_num_threads();
    } /* end of omp single */

    // Create a domain decomposition with num_threads domains
       ...

#   pragma omp for
    for (domain_no = 0; domain_no < num_threads; domain_no++)
    {
      Process_domain( D[domain_no] );

    } /* end of omp for */

  } /* end of omp parallel */
```

This programming style needs num_threads > 1

Reason: No race detection if num_threads == 1



H L R S

TCI

## Restrictions on OpenMP library routines

- **Set functions**

| | |
|---|---|
| omp_set_num_threads() | given value is returned by subsequent calls to omp_get_max_threads(), omp_get_num_procs(), and inside of parallel regions: omp_get_num_threads(). **Real number of threads is still 1!** |
| omp_set_dynamic() | returned by subsequent omp_get_dynamic() |
| omp_set_nested() | ignored, i.e., omp_get_nested() returns always 0 |

- **Lock functions**

  e.g., deadlocks due to wrong nesting of different locks are reported

```
int main(void){
  double x=0, y=0;  int i;  omp_lock_t lock_x, lock_y;
  omp_init_lock(&lock_x);  omp_init_lock(&lock_y);
# pragma omp parallel for shared(x,y)
   for(i=1; i<=N; i++){
      if (i < 0.3*N) {
         omp_set_lock(&lock_x);  x = x + i;
          omp_set_lock(&lock_y);  y = y + i;
          omp_unset_lock(&lock_y);
         omp_unset_lock(&lock_x);
      } else {
         omp_set_lock(&lock_y);  y = y + i;
          omp_set_lock(&lock_x);  x = x + i;
          omp_unset_lock(&lock_x);
         omp_unset_lock(&lock_y);
      } /*endif*/
   } /*end for*/
}
```

TCI

## Restrictions

- Runtime check
  - Therefore error detection only in software branches that are executed
- Often more than **300 times slower** than serial execution and more than **20 times memory consumption**
  - Use small number of iterations
  - Use small data set
  - But large & complex enough that all software branches are executed



End of checking

## End of checking



- When “no errors, cautions, warnings” are reported, multi-threaded run is assured to be **free of semantical OpenMP parallelization errors** (i.e., race conditions, as shown in Examples 1-5), but only in software branches touched by the checked simulation run.
- *We are not sure whether this statement is correct and formally proven.*

TCI and Binary Mode



## Modes

- **Thread count independent analysis mode** TCI
  = Source Code Instrumentation
  → Compilation with
  icc / ifort –g –openmp –tcheck
- **Simulation approach = thread-parallel execution** BIN
  = Binary Instrumentation
  → Compilation with
  icc / ifort –g –openmp (**without** –tcheck)
- **Mixed mode**, e.g., user application with TCI + all libraries with BIN
- **Execution** always with
  tcheck_cl



TCI / BIN

## Example 8 – Mixed mode compilation – Main program

**caller.c**

```
14 void scalar_mult_vector(float *b, float c, float *a, int n);  /* B=c*A */
15 void scalar_plus_vector(float *b, float c, float *a, int n);  /* B=c+A */

17 int main(void)
18 { float a[N], b[N], sum, sum_expected;  int i;
...
26   for (i=0; i<N; i++) { a[i]=1; b[i]=999; }
27 # pragma omp parallel
28   {
29 #   pragma omp sections
30     {
31 #     pragma omp section
32       {
33         scalar_mult_vector(a, 3, a, N);  /* A=3*A */
34       } /* end of omp section */
35 #     pragma omp section
36       {
37         scalar_plus_vector(b, 4, a, N);  /* B=4+A */
38       } /* end of omp section */
39 #     pragma omp section
40       {
41         scalar_mult_vector(a, 5, b, N);  /* A=5*B */
42       } /* end of omp section */
43     } /* end of omp sections */
44   } /* end of omp parallel */
51 }
```

TCI / BIN

## Example 8 – Mixed mode compilation – Library routines

**mylib.c**

```
 1
 2 #ifdef _OPENMP
 3 #include <omp.h>
 4 #endif
 5
 6 void scalar_mult_vector(float *b, float c, float *a, int n);  /* B=c*A */
 7 void scalar_plus_vector(float *b, float c, float *a, int n);  /* B=c+A */
 8
 9 void scalar_mult_vector(float *b, float c, float *a, int n)   /* B=c*A */
10 { int i;
11   for (i=0; i<n; i++) b[i] = c * a[i];
12 }
13
14 void scalar_plus_vector(float *b, float c, float *a, int n)   /* B=c+A */
15 { int i;
16   for (i=0; i<n; i++) b[i] = c + a[i];
17 }
```

Intel Thread Checker [08a]



H L R S

TCI / BIN

## Example 8 – Mixed mode compilation

- Mixed compilation
  - `icc -tcheck -openmp -g -c caller.c` → TCI
  - `icc -g -c mylib.c` → BIN
  - `icc -tcheck -openmp -g -o a.out caller.o mylib.o`

    not required, i.e., works with any library
- Execution and analysis in TCI (projection technology)
  - `tcheck_cl -w 90 ./a.out`
- Call stack for each entry
  - `tcheck_cl -s 1 threadchecker.thr`
  - `tcheck_cl -s 2 threadchecker.thr`
  - ...



H L R S

TCI / BIN

## Example 8 – analysis → call stack

| ID | Short Descrip-tion | Seve-rity Name | Cou nt | Con-text [Best] | Description | 1st Access [Best] | 2nd Access [Best] |
|---|---|---|---|---|---|---|---|
| 1 | **Write -> Read** data-race | Error | 100 | omp for | Memory **read** at "my_lib.c":**16** conflicts with a prior memory **write** at "my_lib.c":**11** (flow dependence) | "my_lib.c ":11 | "my_lib.c ":16 |

```
> tcheck_cl -s 1 threadchecker.thr

First access stack
#0 my_lib.c : 11   , scalar_mult_vector , caller
#1 caller.c : 33   , main , caller
#2  : 4294967295   , call_gmon_start , caller

Second access stack
#0 my_lib.c : 16   , scalar_plus_vector , caller
#1 caller.c : 37   , main , caller
#2  : 4294967295   , call_gmon_start , caller
```



TCI / BIN

## Example 8 – analysis

**Description**

Memory **read** at "my_lib.c":**16** conflicts with a prior memory **write** at "my_lib.c":**11** (flow dependence)

```
First access stack
#0 my_lib.c : 11   , scalar_mult_vector , caller
#1 caller.c : 33   , main , caller
#2  : 4294967295   , call_gmon_start , caller

Second access stack
#0 my_lib.c : 16   , scalar_plus_vector , caller
#1 caller.c : 37   , main , caller
#2  : 4294967295   , call_gmon_start , caller
```

```
caller.c
31 #     pragma omp section
32       {
33         scalar_mult_vector(a, 3, a, N);  /* A=3*A */
34       } /* end of omp section */
35 #     pragma omp section
36       {
37         scalar_plus_vector(b, 4, a, N);  /* B=4+A */
38       } /* end of omp section */
```

**Write-Read race condition**

```
my_lib.c
 9 void scalar_mult_vector(float *b, float c, float *a, int n)   /* B=c*A */
10 { int i;
11   for (i=0; i<n; i++) b[i] = c * a[i];
12 }
14 void scalar_plus_vector(float *b, float c, float *a, int n)   /* B=c+A */
15 { int i;
16   for (i=0; i<n; i++) b[i] = c + a[i];
17 }
```

## When to use …

… Source Code Instrumentation **TCI**

- Source instrumentation is required for analysis when binary instrumentation is not available.
- When you want to run the instrumented program outside of the VTune™ Performance Environment. For example, use source instrumentation for a server application.

… Binary Instrumentation **BIN**

- If you do not have access to an appropriate Intel® compiler.
- If you do not want to rebuild your application, for example, because it might take many hours to do so.
- If you do not have access to the source code.

From user's manual **tcheck.chm**
Section **Instrumenting Code for Intel(R) Thread Checker**



**BIN**

## Binary instrumentation and parallel execution

- Compilation: only with Intel® compilers, as usual, but with **–g** option:

```
icc   -openmp -g  -o my_prog  my_prog.c
ifort -openmp -g  -o my_prog  my_prog.f
```

- Parallel execution and analysis with **tcheck_cl**

```
export OMP_NUM_THREADS=4
tcheck_cl -w 90 -o my_prog.txt ./my_prog
```

- Problems with Intel® Thread Checker 3.0:
  - Reports many non-existing errors
  - Cannot find problems, e.g., as in Example 4
  - Cannot report variable names
  - Wrong count-values

➔ **But may help if** **TCI** **is not applicable**

## Recommendation

- Use Source Code Instrumentation = Projection Mode **TCI**
  = Thread Count Independent Mode (TCI)
  - If this mode is applicable to your application
  - TCI provides much more information
    and much more value to the users of OpenMP
  - But it can not be used with all applications

- If TCI is not applicable,
  (or after all TCI reports are resolved,)
  then use
  Binary Instrumentation = Thread Count Dependent Mode **BIN**



Wrap-up

## Thread Checker Requirements



- Intel® Thread Checker 3.0 beta
  - Linux or Windows XP Professional
  - Intel® processors
  - Intel® compiler
  - Intel® Vtune™

- On Windows XP Professional:
  - Additional GUI interface
  - ➔ my personal recommendation:
    Use text output with line-numbers

## History

- P. M. Petersen:
  **Evaluation of Programs and Parallelizing Compilers Using Dynamic Analysis Techniques.**
  PhD thesis, University of Illinois at Urbana-Champaign, January 1993.
  http://citeseer.ist.psu.edu/petersen93evaluation.html
- **Assure**
  - a KAP/pro tool
  - it was available for most shared memory platform
  - sold to Intel → basis for Intel® Thread Checker
- Paul Petersen, Sanjiv Shah:
  **OpenMP Support in the Intel® Thread Checker.**
  WOMPAT 2003, pp 1-12.
  http://sunsite.informatik.rwth-aachen.de/dblp//db/conf/wompat/wompat2003.html#PetersenS03
- **Intel® Thread Checker**
  - Product for Windows since 2005
  - Beta-test for Linux 3/2006

Intel Thread Checker [08a]



## Further information and reading

- `tcheck_cl` without arguments shows list of options
- `export TC_OPTIONS=help`
  `tcheck_cl ./my_prog`
  will return a list of additional TC_OPTIONS
  and output is mixed with run-time checking analysis, e.g.

```
[Intel(R) Thread Checker Report: OpenMP undefined access]
Exa.4: a[0] computed=  1188.0,  expected=  1188.0,  difference= 0.00000
[Intel(R) Thread Checker Report: OpenMP undefined access]
```

- GettingStarted.pdf & DiagnosticsGuide.pdf (on /opt/intel/itt/tcheck/doc)
- tcheck.chm contains the user manual (Windows?)
- Intel Threading Tools:
  **http://www.intel.com/cd/software/products/asmo-na/eng/threading/index.htm**
- At HLRS:
  **http://www.hlrs.de/organization/amt/services/tools/debugger/ittc/**

## Acknowledgements


- This lecture is based on the Assure presentation by Hans-Joachim Plum, Pallas GmbH
- The exercises *race1/2* and *conflict* were developed by Matthias Müller
- The Intel® Thread Checker 3.0 beta was installed by Dmitri Chubarov and Bettina Krammer, the Intel® Thread Checker 3.0 by Danny Sternkopf
- Thanks to James Cownie and Paul Petersen for their review comments.




Summary

## Summary

- **Intel® Thread Checker** finds the locations of race conditions, but the **programmer** must detect the reasons!
- Source code instrumentation (TCI) – executed with 1 thread – returns an important error report
- Programmer has to **eliminate** all these errors – or **must be sure** that the reported error can be ignored
- Binary instrumentation works – executed in parallel with multiple threads – should be used if TCI is not applicable (or after all TCI reports are resolved)

TCI

BIN

It is **absolutely** necessary to **verify** OpenMP parallelizations with a **race-condition detection tool**.

Currently[1)] we see on the market only one OpenMP race-condition detection tool that can be used under Linux and Windows:

This is the Intel® Thread Checker

[1)] June 30, 2006

Practical

## Intel® Thread Checker – Practical (on cacau.hww.de)



- **module load itt/tcheck-3.0** (setup, only once)
- **cd ~/OpenMP/#nr/pitfalls/**
- Compiling the application together with the Intel® Thread Checker:
  **icc –tcheck –openmp –g –o** *my_prog my_prog***.c**
  or
  **ifort –tcheck –openmp –g –o** *my_prog my_prog***.f**
- Executing & analyzing the application together with the Intel® Thread Checker on one processor, detecting all race conditions:
  **tcheck_cl –w 90 –o** *my_prog***.txt** **./***my_prog*
- with *myprog* = **conflict**, **race1**, and **race2** (and as additional exercise: demo_with_bugs.c)
- Tasks:
  – find the reasons in all 3 examples
  – correct the source (without modifying the numerical semantics)
  – verify again with Intel® Thread Checker
  – until **"tcheck_cl"** does not report any further error, caution, warning, …



Appendix

## Appendix



Example 1:
- Two separate loops → will cause bad cache reuse!

or
- Re-computing of `a[i-1]`, i.e., `b[i] = a[i] - 2.0*(i-1)*(i-2);`

Example 2: Remove "nowait"

Example 3: Add "private(x)"

Example 4: Use "firstprivate(f) lastprivate(x)" instead of "private(f,x)"

Example 5: Add "reduction(+:sum)"

## References

- United States Patent 6,286,130 (September 4, 2001)
  David K. Poulsen, Paul M. Petersen, Sanjiv M. Shah:
  *Software implemented method for automatically validating the correctness of parallel computer programs*
  http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=6286130.PN.&OS=PN/6286130&RS=PN/6286130
  - **This patent describes implementation and goals of the projection mode**

**"We claim:**
**1. A method for detecting individual errors in a parallel computer program by translating a parallel computer program into a sequential computer program, said method comprising the steps of:**

- identifying a parallel computer program having at least one parallelism specification;
- generating a corresponding sequential computer program to the parallel computer program by ignoring said at least one parallelism specification contained in the parallel computer program;
- adding to said corresponding sequential computer program at least one first instruction, to generate at least one first trace event, said at least one first instruction relating to said corresponding sequential computer program, and at least one second instruction, to generate at least one second trace event, said at least one second instruction based upon the ignored at least one parallelism specification
- logically partitioning the sequential computer program into at least one disjoint group based upon the at least one second trace event, said at least one disjoint group comprising at least one of the at least one first trace events; and
- executing only said sequential computer program a single time, and analyzing said at least one disjoint group of said at least one first trace event based on types of second trace events used to partition said at least one first trace event **to detect and report each precise semantic inconsistency** between said parallel computer program and said corresponding sequential computer program, thereby detecting one or more semantic inconsistencies associated with a plurality of different executions of the parallel computer program.„

**Highlighted** by the author of this slide.

## References

- Utpal Banerjee, Brian Bliss, Zhiqiang Ma, Paul Petersen (Intel):
  *Unraveling Data Race Detection in the Intel Thread Checker*
  Proceedings, First Workshop on Software Tools for Multi-Core Systems (STMCS'06), March 26, 2006, http://www.isi.edu/~kintali/stmcs06/
  (at The 4th Annual International Symposium on Code Generation and Optimization (CGO) http://www.cgo.org/cgo2006/)
  - **This paper describes the theory used in the simulation approach.**
  - **References to other products (see next slide)**

From Section 5.1:
„Happens-before requires logging the history of accesses to every shared memory variable. Instead of a complete access history, we only keep the most recently read and most recently written accesses. The tool is able to catch most data races, **but it doesn't guarantee to catch all data races in a single run**. We believe that, in practice, catching as many races of a program run as possible in a detailed manner is better than catching all races of a program run in a brief way."

**Highlighted** by the author of this slide.

## Other Products

- Sun Studio Express: Data Race Detection Tool (DRDT)
  - Automatic OpenMP source code instrumentation
  - Analysis of data races in a parallel execution
  - **http://developers.sun.com/prodtech/cc/downloads/drdt/getting_started.html**
  - **http://developers.sun.com/prodtech/cc/downloads/drdt/using.html (Tutorial)**
- Jprobe Threadanalyzer from Quest Software
  - Detects data races in Java programs
  - **www.quest.com/jprobe**
- Visual Threads from HP
  - Detects data races in POSIX threaded programs
  - **h18000.www1.hp.com/products/software/visualthreads/**
- Helgrind of the Valgrind tool suite
  - Open source data race detection tool
  - For POSIX threaded programs on Linux