Detecting Data Race Conditions: Intel® Inspector XE

2011 Par Lab Boot Camp
The Parallel Computing Laboratory
August 17, 2011

Gary Carleton, Intel Corp.
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Case Study: Threading the Intel® Compiler
5. Readying your sources and builds
6. Advanced features
Intel® Inspector XE

• Is a specialized debugging tool for threaded software.
  – Detects:
    – Memory issues
      – Memory leaks (Allocated Memory is never released)
      – Memory incorrect API usage (Allocated with malloc and deallocated with delete instead of free)
    – Threading issues
      – Data races (two or more threads access memory without synchronization. At least one thread is writing data)
      – Deadlocks (Thread waits for an event that will never happen)
  
• GUI and command line based
  – Provides powerful results management, navigation and filtering
• Also supports common threading models
  – OpenMP*
  – Intel® Threading Building Blocks
  – Win32* and Posix Threads
# Key Features at a glance

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analyses</strong></td>
<td>• Dynamic Memory and Threading Analysis (including .NET analysis)</td>
</tr>
<tr>
<td></td>
<td>• Static Security Analysis (with Intel® Compiler Pro 12.0)</td>
</tr>
<tr>
<td><strong>GUI</strong></td>
<td>• Microsoft Visual Studio IDE integration (2005, 2008 and 2010)</td>
</tr>
<tr>
<td></td>
<td>• Stand alone GUI on both Windows and Linux</td>
</tr>
<tr>
<td><strong>Compilers supported</strong></td>
<td>• Microsoft* Visual* C++, .NET*</td>
</tr>
<tr>
<td></td>
<td>• Intel® Parallel Composer and Intel® Composer XE</td>
</tr>
<tr>
<td></td>
<td>• gcc</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>• Windows XP, Vista, 7</td>
</tr>
<tr>
<td></td>
<td>• Linux (various distros)</td>
</tr>
<tr>
<td><strong>Languages</strong></td>
<td>• C/C++</td>
</tr>
<tr>
<td></td>
<td>• C#</td>
</tr>
<tr>
<td></td>
<td>• Fortran</td>
</tr>
</tbody>
</table>
Standalone GUI for Windows* and Linux*
Visual Studio* Integration on Windows*

[Image of Visual Studio integration interface]
Evolution of The Inspector XE

The Inspector XE retains key features of Intel® Thread Checker and Intel® Parallel Inspector.

In addition, new **exciting** features (in bold blue) are added.

- **Intel® Thread Checker**
  - Improved Threading Analysis
  - Improved Memory Analysis
  - JIT Instrumentation

- **Intel® Parallel Inspector**
  - Stand alone GUI
  - .NET Analysis
  - Child Program Analysis

- **Intel® Compiler**
  - Static Security Analysis
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Case Study: Threading the Intel® Compiler
5. Readying your sources and builds
6. Advanced features
Workflow
Select Analysis and Start

1. Select Analysis Type

2. Click Start
# Manage Results and Filter

1. Double click on Problem to navigate to source (next slide)

2. Double click on Timeline to bring up timeline view (slide after next slide)

- Code locations grouped into Problems to simplify results management

<table>
<thead>
<tr>
<th>Code Locations</th>
<th>Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Description</td>
</tr>
<tr>
<td>X4</td>
<td>HINT: Synchronization alloc ...</td>
</tr>
<tr>
<td>X1</td>
<td>Read</td>
</tr>
<tr>
<td>X2</td>
<td>Write</td>
</tr>
<tr>
<td>X3</td>
<td>Write</td>
</tr>
</tbody>
</table>

- Powerful filtration feature!
Workflow: **Navigate to sources**

Source code panes annotated for ease of use.
Workflow: *Timeline view*

Individual Code Locations are seen in Timeline view in the context of their respective threads.
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Case Study: Threading the Intel® Compiler
5. Readying your sources and builds
6. Advanced features
Memory Analysis

• Analyzed as software runs
  – Data (workload) -driven execution
  – Program can be single or multi-threaded
  – Diagnostics reported incrementally as they occur

• Includes monitoring of:
  – Memory allocation and allocating functions
  – Memory deallocation and deallocating functions
  – Memory leak reporting
  – Inconsistent memory API usage.

Code path must be executed to be analyzed
Threading Analysis

• Dynamic as software runs
  – Data (workload) -driven execution
  – Program needs to be multi-threaded
  – Diagnostics reported incrementally as they occur

• Includes monitoring of:
  – Thread and Sync APIs used
  – Thread execution order
    – Scheduler impacts results
  – Memory accesses between threads

Code path must be executed to be analyzed
Threading Analysis (.NET)

- .NET support is only for Windows C#.
- Only Threading Analysis is supported for managed code or mixed mode code.
- Memory Checking works only on the native portion of a mixed-mode application (does not track any memory in managed heap).
- Analysis can be performed using the Inspector XE integrated within Visual Studio* or using Windows Standalone Inspector GUI.
- Supported .NET versions are from 2.0 to 3.5.
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Case Study: Threading the Intel® Compiler
5. Readying your sources and builds
6. Advanced features
Threading the Intel® Compiler

• Prototyping proof-of-concept of *Threading Checker*: a tool to assist application architects in threading their applications

• We parallelized the Intel C++ compiler using extensions to Intel Thread Checker.
  – Compiler source is ~ 1000 C files
  – Millions lines of complex dynamic data structures
  – Prototype of Intel® Inspector XE used

• Authors:
  – Mohammad Haghighat
    – Thanks to Moh for allowing me to plagiarize his results
  – Knud J. Kirkegaard, Ravi Narayanaswamy, David Sehr, Bhanu Shankar, Neil Faiman
  – “Methodology, Tools, and Techniques to Parallelize Large-Scale Applications: A Case Study”, Mohammad Haghighat, et al
Threading Methodology

1. Discover Parallelism
   - Compiler Loop Profiler
   - Thread Checker & extensions

2. Express Parallelism
   - OpenMP initially

3. Debug Threaded App
   - Thread Checker

4. Tune Threaded App
   - Thread Profiler / VTune™ Analyzer
Parallelism Discovery: where is time being spent?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ticks(%)</td>
<td>ticks(abs)</td>
<td>self(%)</td>
<td>self(abs)</td>
<td>entry_cnt</td>
<td>exit_cnt</td>
<td>file:line:col:ndx</td>
</tr>
<tr>
<td>2</td>
<td>51.7</td>
<td>675952724</td>
<td>0</td>
<td>32548</td>
<td>1</td>
<td>1 shared/ip/layout.c:657:39:1 IPO/Layout_Driver</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>235966016</td>
<td>1</td>
<td>13027220</td>
<td>14782</td>
<td>14782</td>
<td>shared/edgpfe/lexical.c:956</td>
</tr>
<tr>
<td>4</td>
<td>16.1</td>
<td>211070764</td>
<td>0</td>
<td>343808</td>
<td>1</td>
<td>1 shared/edgpfe/decls.c:145 translation_unit</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6.5</td>
<td>84369904</td>
<td>0.5</td>
<td>6961592</td>
<td>1</td>
<td>1 shared/intrin/intrin.c:1536:5 intrin_Process</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4.7</td>
<td>61261408</td>
<td>0</td>
<td>264604</td>
<td>2</td>
<td>2 shared/edggue/edg_code.c dump_all_routine</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4.5</td>
<td>59420636</td>
<td>0.4</td>
<td>5868756</td>
<td>87</td>
<td>87 shared/iLO/verify.c:3014:5:0 verify_ILO_BB_W</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4.2</td>
<td>54363924</td>
<td>0.2</td>
<td>2808160</td>
<td>217</td>
<td>217 shared/edgpfe/preproc.c:46 skip_to_endif@source</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4.1</td>
<td>54079124</td>
<td>0.7</td>
<td>9447284</td>
<td>4228</td>
<td>4228 shared/edgpfe/lexical.c:67(skip_white_space</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.9</td>
<td>51635388</td>
<td>0.7</td>
<td>260892</td>
<td>27</td>
<td>27 shared/edgpfe/lexical.c:44 search_for_input</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3.5</td>
<td>5165540</td>
<td>0.2</td>
<td>2365540</td>
<td>188</td>
<td>188 shared/edgpfe/decls.c:133 declaration@shared/edgpfe/decls.c:133</td>
<td></td>
</tr>
</tbody>
</table>

Ready
Parallel Compilation

Compiler Routine Driver

Compile Routine
Analysis 1
Optimization 1
Analysis N
Optimization N

Compile Routine
Analysis 1
Optimization 1
Analysis N
Optimization N

Compile Routine
Analysis 1
Optimization 1
Analysis N
Optimization N

.obj

Code Generator
Expressing Parallelism

Used OpenMP for portability & better ITT support
Parallelism Discovery: dependences?

- Used Intel Thread Checker in Projection mode
  - Runs app serially looking for potential Data Race sites
  - “A Theory of Data Race Detection”, Banerjee, Bliss, Ma, Petersen
    [http://cs.ucsb.edu/~arch/spr07-seminar/papers/threadchecker06.pdf](http://cs.ucsb.edu/~arch/spr07-seminar/papers/threadchecker06.pdf)

- ~300,000 errors initially reported

- Obvious need to filter / organize this many reported errors
## Tracking Progress in Parallelization Process

Reported Data Races as project progressed

<table>
<thead>
<tr>
<th>Run</th>
<th>Date</th>
<th>Time</th>
<th>Violations</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>Sep 15</td>
<td>15:44</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Sep 14</td>
<td>15:59</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Sep 9</td>
<td>13:39</td>
<td>95</td>
<td>8</td>
</tr>
<tr>
<td>37</td>
<td>Sep 7</td>
<td>17:37</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Sep 7</td>
<td>16:17</td>
<td>190</td>
<td>70</td>
</tr>
<tr>
<td>35</td>
<td>Sep 6</td>
<td>15:40</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Sep 6</td>
<td>15:02</td>
<td>124</td>
<td>13</td>
</tr>
<tr>
<td>33</td>
<td>Sep 6</td>
<td>13:35</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Sep 5</td>
<td>17:50</td>
<td>206</td>
<td>3</td>
</tr>
<tr>
<td>31</td>
<td>Sep 5</td>
<td>10:32</td>
<td>277</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Sep 3</td>
<td>16:51</td>
<td>278</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Sep 1</td>
<td>17:00</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Aug 31</td>
<td>21:56</td>
<td>356</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Aug 31</td>
<td>18:43</td>
<td>370</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Aug 31</td>
<td>16:14</td>
<td>622</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Aug 30</td>
<td>21:44</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Aug 30</td>
<td>18:48</td>
<td>654</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Aug 30</td>
<td>13:13</td>
<td>701</td>
<td>60</td>
</tr>
<tr>
<td>22</td>
<td>Aug 29</td>
<td>18:14</td>
<td>684</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Aug 29</td>
<td>10:01</td>
<td>723</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Aug 26</td>
<td>13:44</td>
<td>706</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Aug 26</td>
<td>15:55</td>
<td>712</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Aug 25</td>
<td>18:29</td>
<td>791</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Aug 25</td>
<td>15:07</td>
<td>889</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Aug 25</td>
<td>09:40</td>
<td>971</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Aug 24</td>
<td>22:39</td>
<td>1,021</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Aug 24</td>
<td>18:59</td>
<td>1,031</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Aug 24</td>
<td>12:41</td>
<td>1,502</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Aug 23</td>
<td>10:39</td>
<td>1,053</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Aug 23</td>
<td>16:02</td>
<td>1,815</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Aug 22</td>
<td>16:02</td>
<td>2,087</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Aug 21</td>
<td>18:10</td>
<td>3,437</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Aug 21</td>
<td>16:49</td>
<td>3,430</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Aug 17</td>
<td>10:34</td>
<td>4,435</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Aug 18</td>
<td>16:26</td>
<td>12,132</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Aug 15</td>
<td>11:53</td>
<td>12,514</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Aug 9</td>
<td>11:43</td>
<td>12,009</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Aug 4</td>
<td>13:59</td>
<td>15,713</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Aug 3</td>
<td>15:13</td>
<td>52,287</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Jul 21</td>
<td>11:34</td>
<td>294,487</td>
<td></td>
</tr>
</tbody>
</table>
Reducing violations

- Prioritize violations reduction
  - Identify allocation routines
  - Fix variables with highest violations count
- Linear reduction wouldn’t work
  - Goal not achievable in 6 months (~2300/day)

Thread Checker’s GUI particularly helpful here
Speedups vs. the sequential compiler
Conclusions

• Complex applications can also be effectively parallelized

• Intel Threading Tools provide great help

• Inspector XE/Thread Checker: very helpful in threading and debugging threaded applications

• Effective summarization is key to manage the size

• Semi-automatic source changes provide great help
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Case Study: Threading the Intel® Compiler
5. Readying your sources and builds
6. Advanced features
Correctness analyses dilate time & memory

• Adds calls to library to record information using PIN based *Just in time* instrumentation.
  – Thread and Sync APIs
  – Memory accesses
– Increases *execution time* and *memory consumed (potentially significantly)*

• Use *small* data sets (workloads)
  – Execution time and space is *expanded*
  – Multiple runs over different paths yield best results

The Inspector XE dilates both time and memory consumed significantly!
Workload Guidelines

- Execute problem code once per thread to be identified
- Use smallest possible working data set
  - Minimize data set size
    - Smaller image sizes
  - Minimize loop iterations or time steps
    - Simulate minutes rather than days
  - Minimize update rates
    - Lower frames per second

Scale down workload to speed up analysis!
Prepare your build for analysis

• Compile
  – Use dynamically linked thread-safe runtime libraries
    `/MDd` on Windows
  – Generate symbolic information
    `/ZI` on Windows
  – Disable optimization
    `/Od` on Windows

• Link
  – Preserve symbolic information
    `/DEBUG` on Windows
  – Specify relocatable code sections
    `/FIXED:NO` on Windows

Prior to using Inspector XE, sources should compile & link cleanly
Recommended Analysis Sequence

- Run Static Security Analysis
  - Fix Problems
- Run Memory Analysis
  - Fix Problems
- Run Threading Analysis
  - Fix Problems
- Cleaner code!

Use both Static and Dynamic analyses for cleaner code!
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Security Analysis
5. Readying your sources and builds
6. Advanced features
Child Program Analysis

• Running a top level script is the norm for some Linux apps. For such cases, a different Child Program can be analyzed (not necessarily the app launched by Inspector XE).

• Limitations:
  - Only the first instance of Child Program will be analyzed by Inspector XE analysis.
  - Child Program name is the one shown in Windows Task Manager or the name shown in “ps –aef” on Linux.
  - Multi-process analysis is not supported for .NET applications.

Child Program Analysis is very useful in multi-process scenario
Command Line Interface

- `inspex-cl` is the command to use Inspector XE CLI.
  - Windows: `C:\Program Files (x86)\Intel\Inspector XE 2011\bin32\inspex-cl.exe`
  - Linux: `/opt/intel/inspector_xe_2011/bin64(32)/inspex-cl`

- To get detailed help:
  `inspex-cl -help`

- Command examples:
  1. `inspex-cl -collect-list`
  2. `inspex-cl -knob-list=tc`
  3. `inspex-cl -collect=tc -knob=scope=l2 - knob=terminate-on-deadlock=true -- Deadlock.exe`

CLI is very useful for running The Inspector XE as part of regression tests
Reporting

• To generate a report:
  `inspex-cl -R=<report-type> <results directory name>`

• Sample commands:
  `inspex-cl -report-list`
  `inspex-cl -report=summary`
  `inspex-cl -report=problems`

• Example:
  `cd /home/user/testProgram/r000mi`
  `cd ..`
  `inspex-cl -R=observations r000mi`

Report generation is very convenient to use from command line.
References

• “Methodology, Tools, and Techniques to Parallelize Large-Scale Applications: A Case Study”, Mohammad Haghighat, et al

• “A Theory of Data Race Detection”, Banerjee, Bliss, Ma, Petersen
  [http://cs.ucsb.edu/~arch/spr07-seminar/papers/threadchecker06.pdf]
## Optimization Notice

Intel compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel Compiler User and Reference Guides" under “Compiler Options." Many library routines that are part of Intel compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel Streaming SIMD Extensions 2 (Intel SSE2), Intel Streaming SIMD Extensions 3 (Intel SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20110228
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.


*Other names and brands may be claimed as the property of others.

Copyright © 2011. Intel Corporation.

http://intel.com/software/products