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

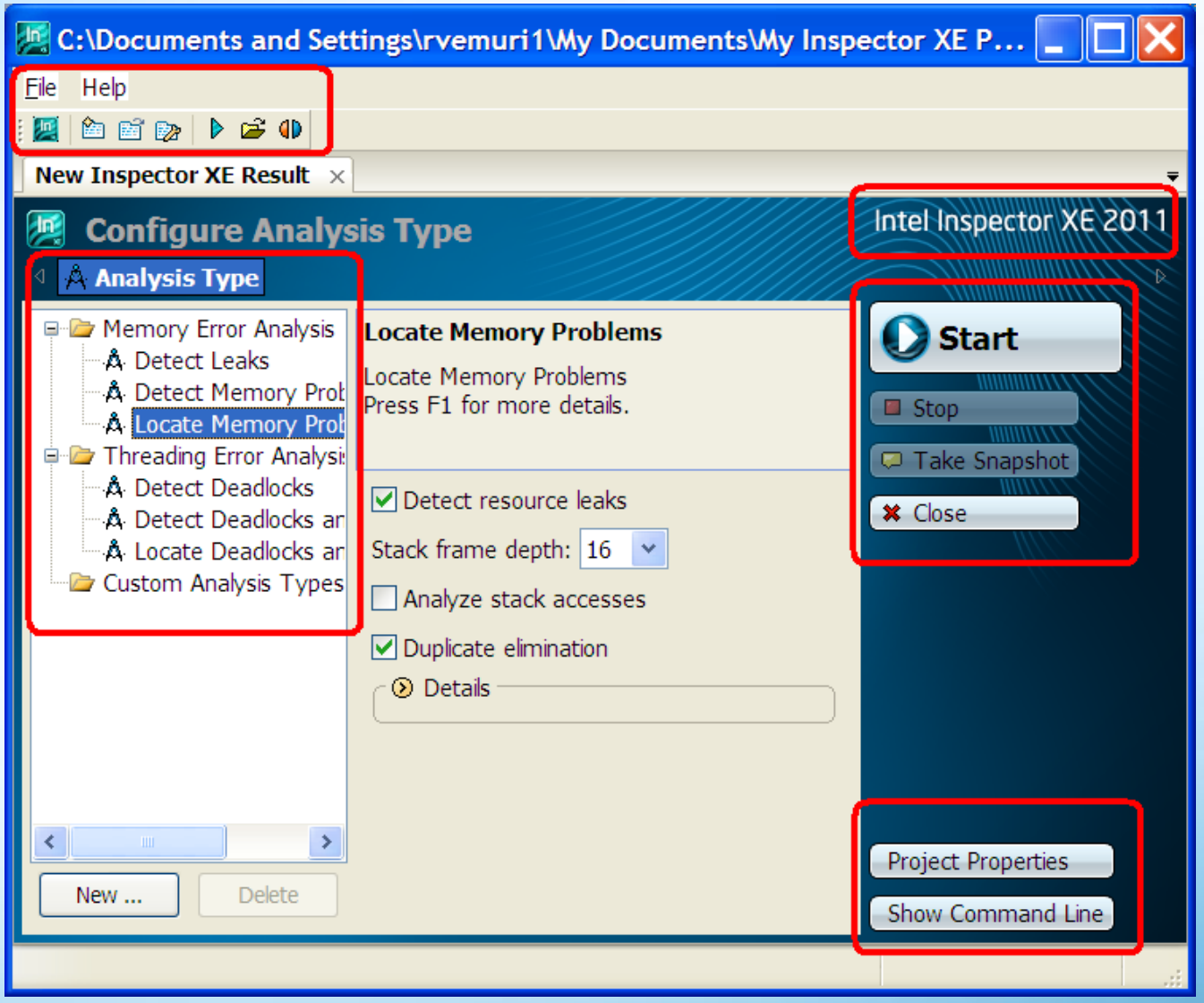
- Is a specialized *debugging tool* for threaded software.
- Detects:
  - Memory issues
    - Memory leaks (Allocated Memory is never released)
    - Memory incorrect API usage (Allocated with malloc and deallocated with delete instead of free)
  - Threading issues
    - Data races (two or more threads 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

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

Standalone GUI for Windows* and Linux*



Visual Studio* Integration on Windows*

DataRaces - Microsoft Visual Studio

File Edit View Project Build Debug Tools Test Window Community Help

Debug Win32

r000mi3

Locate Memory Problems

Intel Inspector XE 2011

Target | Analysis Type | Collection Log | Summary

Problems

ID		Problem	Sources	Modules	Object Size	State
P1		Kernel resource ...	datarac...	DataRaces. ...		New

Code Locations | Code Locations / Timeline

ID	Description	Source	Func...	Module	Object...	Sta..	Off ...
X	Process or thread ha...	datara...	main	DataRa...		New	

Filters Sort

Severity
Error 1 item(s)

Problem
Kernel resource leak 1 item(s)

Source
dataraces.c 1 item(s)

Module
DataRaces.exe 1 item(s)

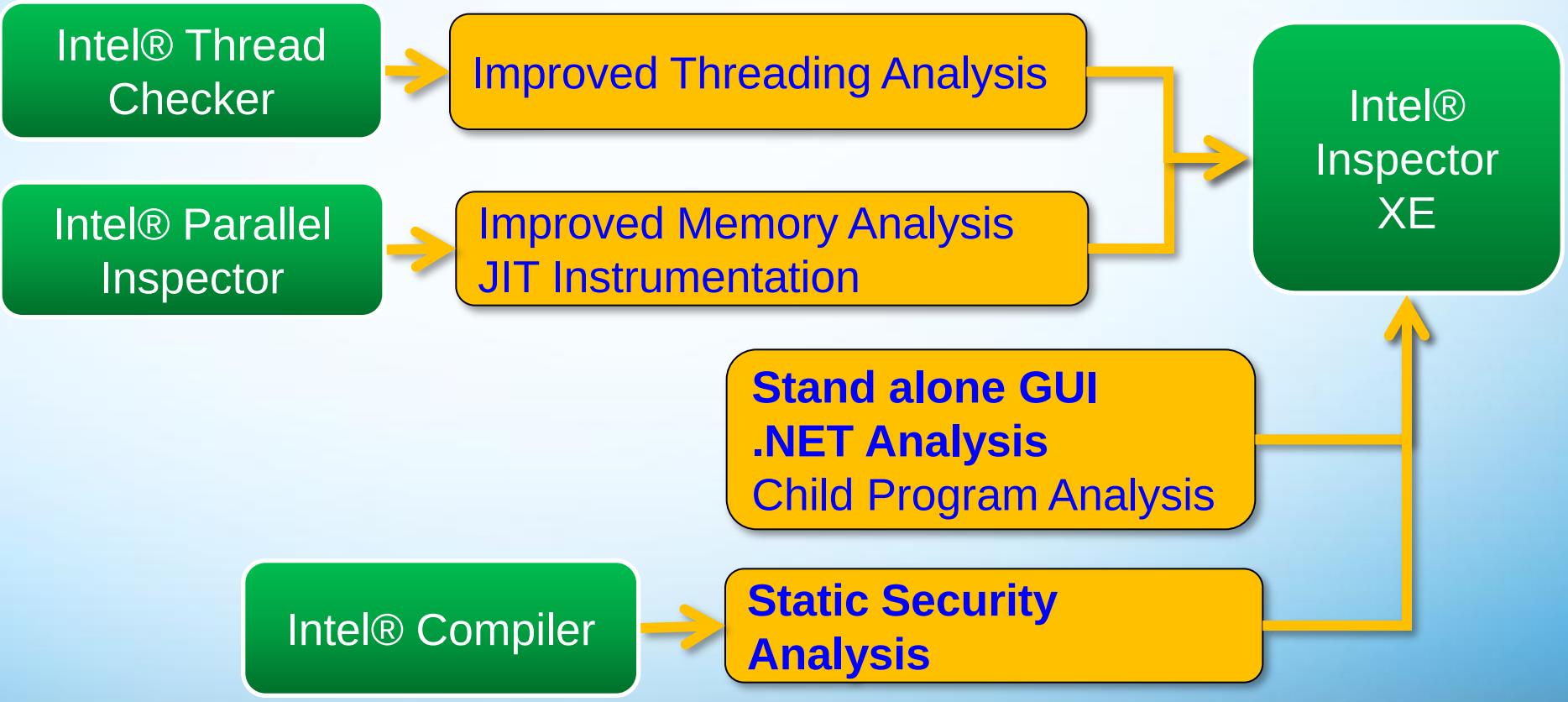
State

Ready

Evolution of The Inspector XE

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

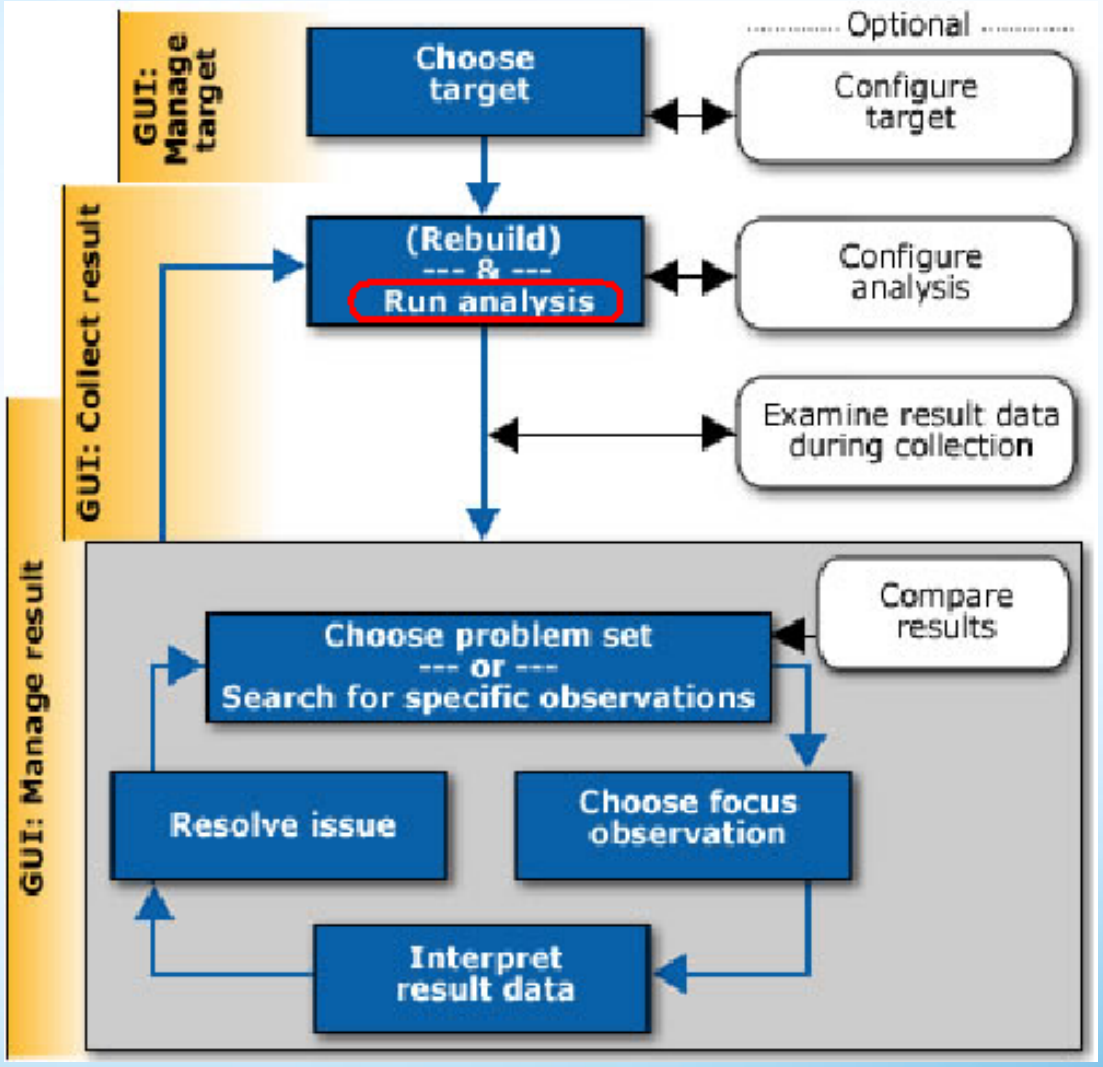
In addition, new **exciting** features (in bold blue) are added.



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

C:\Documents and Settings\rvemuri1\My Documents\My Inspector XE Projects\Sample - I...

File Help

r000ti3 | New Inspector XE Result

Configure Analysis Type

Intel Inspector XE 2011

Analysis Type

- Memory Error Analysis
  - Detect Leaks
  - Detect Memory Prob
  - Locate Memory Prob
- Threading Error Analysis
  - Detect Deadlocks
  - Detect Deadlocks ar
  - Locate Deadlocks ar
- Custom Analysis Types

Locate Deadlocks and Data Races

Locate Deadlocks and Data Races
Press F1 for more details.

Terminate on deadlock

Stack frame depth: 16

Scope: Normal

Duplicate elimination

Details

Start

Stop

Take Snapshot

Close

Project Properties

Show Command Line

New ...

Delete

1. Select Analysis Type

2. Click Start

Manage Results and Filter

C:\Documents and Settings\rvemuri1\My Documents\My Inspector XE Project

File Help

r000ti3

Locate Deadlocks and Data Races

Target | Analysis Type | Collection Log

Problems

ID		Problem	Sources	Modules	State
P1		Data race	dataraces.c	DataRaces.exe	New

Code Locations | Code Locations / Timeline

ID	Description	Source	Function	Module	State
X4	HINT: Synchronization alloc...	dataraces...	increme	aRace...	New
X1	Read	dataraces...	i	DataRace...	New
X2	Write	datar	ent	DataRace...	New
X3	Write		crement	DataRace...	New

Filters Sort

Severity	
Error	1 item(s)
Problem	
Data race	1 item(s)
Source	
dataraces.c	1 item(s)
Module	
DataRaces.exe	1 item(s)
State	
New	1 item(s)
Suppressed	
Not suppressed	1 item(s)
Investigated	
Not investigated	1 item(s)

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

Code locations grouped into Problems to simplify results management

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

Powerful filtration feature!

Workflow: *Navigate to sources*

C:\Documents and Settings\rvemuri1\My Documents\My Inspector XE Projects\Sample - I...

File Help

r000ti3

Locate Deadlocks and Data Races — Intel Inspector XE 2011

Target | Analysis Type | Collection Log | Summary | Sources

Focus Code Location: dataraces.c:28 - Write

```
26
27          EnterCriticalSection (&cs);
28              globalX++;
29          LeaveCriticalSection (&cs);
30
```

Call Stack: DataRaces.exe!increment - da...

Related Code Location: dataraces.c:28 - Read

```
26
27          EnterCriticalSection (&cs);
28              globalX++;
29          LeaveCriticalSection (&cs);
30
31      DeleteCriticalSection (&cs);
```

Call Stack: DataRaces.exe!increment - da...

Source code panes annotated for ease of use

Code Locations / Timeline

ID	Description	Source	Function	Module	State
X4	HINT: Synchronization allocation site	dataraces.c:25	increment	DataRaces.exe	New
X1	Read	dataraces.c:28	increment	DataRaces.exe	New
X2	Write	dataraces.c:28	increment	DataRaces.exe	New

Workflow: *Timeline view*

C:\Documents and Settings\rvemuri1\My Documents\My Inspector XE Projects\Sample - I...

File Help

r000ti3

Locate Deadlocks and Data Races — Intel Inspector XE 2011

Target | Analysis Type | Collection Log | Summary

Problems

ID		Problem	Sources	Modules	State
P1		Data race	dataraces.c	DataRaces.exe	New

Filters Sort

Severity	
Error	1
Problem	
Data race	1
Source	
dataraces.c	1
Module	
DataRaces.exe	1
State	
New	1
Suppressed	
Not suppressed	1
Investigated	
Not investigated	1

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

Timeline — Code Locations / Timeline

29000000 30000000 31000000 31658331 00000 330000

16760
16764

X2: Write [32186656]

X3: Write [32186656]

X2: Write [32186656] Duration: 38256640

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.

Only Threading Analysis is supported for .NET

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

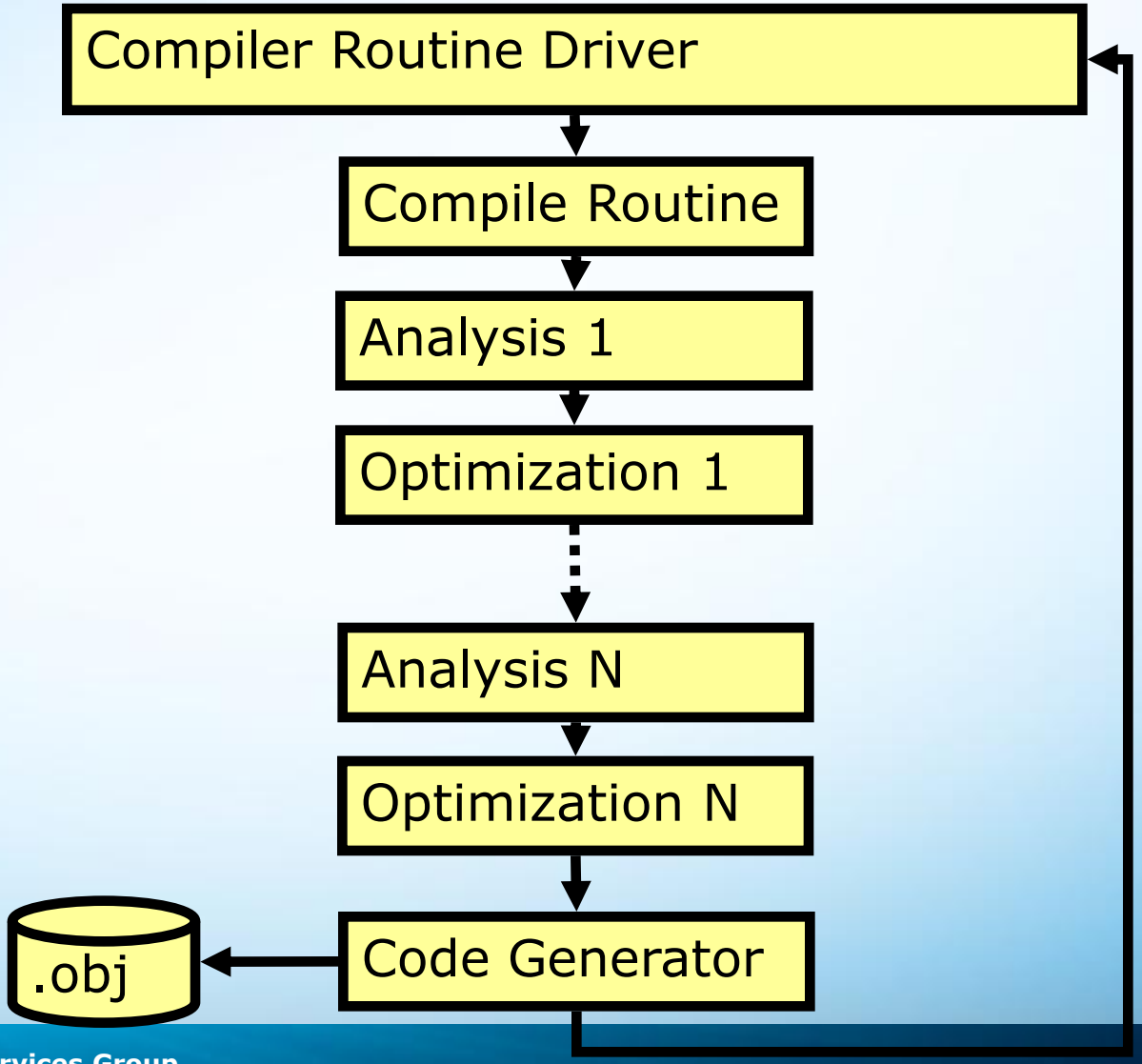
- We parallelized the Intel C++ compiler using extensions to Intel Thread Checker.
  - Compiler source is ~ 1000 C files
  - Millions lines of complex dynamic data structures
  - Prototype of Intel® Inspector XE used

- 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 http://download.intel.com/technology/itj/2007/v11i4/6-tools/6-Methodology_Tools_and_Techniques.pdf

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

Serial Compilation



Parallelism Discovery: where is time being spent?

Microsoft Excel - icc_prof.xls

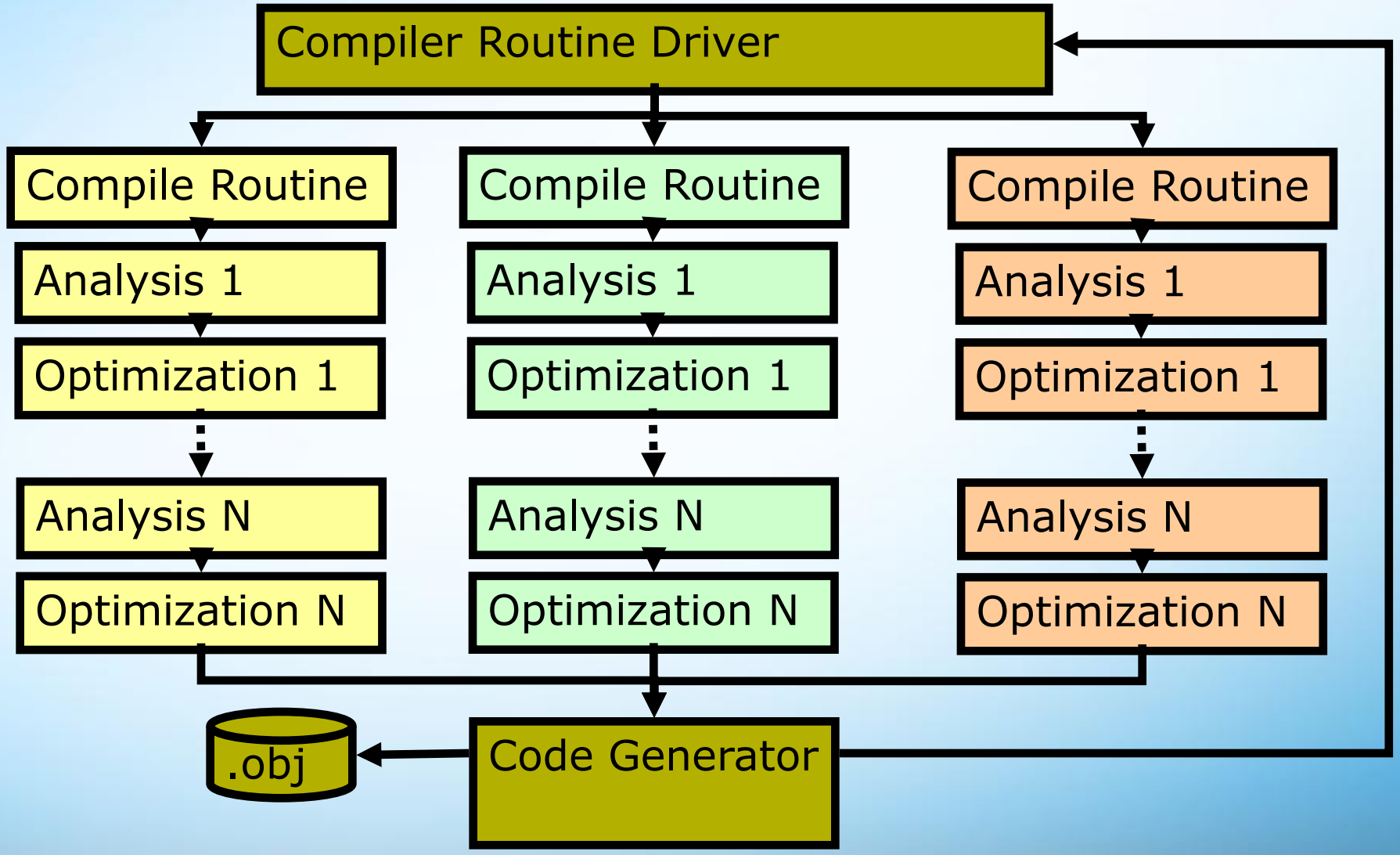
A2 | 51.7

	A	B	C	D	E	F	G	H
1	ticks(%)	ticks(abs)	self(%)	self(abs)	entry_cnt	exit_cnt	file:line:col:ndx	function@file:line
2	51.7	675952724	0	32548	1	1	shared/ip/layout.c:657:39:1	IPO_Layout_Driv
3	18	235966016	1	13027220	14782	14782	shared/edgcpfe/lexical.c:958	get_token@shar
4	16.1	211070764	0	343808	1	1	shared/edgcpfe/decls.c:145	translation_unit@
5	6.5	84369904	0.5	6961592	1	1	shared/intrin/intrin.c:1536:5:	intrin_Process_In
6	4.7	61261408	0	264604	2	2	shared/edgglue/edg_code.c:	dump_all_routine
7	4.5	59420636	0.4	5868756	87	87	shared/il0/verify.c:3014:5:0	verify_Il0_BB_W:
8	4.2	54363924	0.2	2808160	217	217	shared/edgcpfe/preproc.c:48	skip_to_endif@s
9	4.1	54079124	0.7	9447284	4228	4228	shared/edgcpfe/lexical.c:670	skip_white_spac
10	3.9	51635388	0	260892	27	27	shared/edgcpfe/lexical.c:44	search_for_input
11	3.5	45165460	0.2	2365540	189	189	shared/edgcpfe/decls.c:132	declaration@sho

icc_prof

Ready

Parallel Compilation



Expressing Parallelism

Used OpenMP for portability & better ITT support

Microsoft Development Environment [design] - layout.c [Read Only]*

File Edit View Debug Tools Window Help

pragma omp

Start Page | layout.c*

```
static void IPO_Layout_Driver_elts(BOOL opt_level)
{
    int numThreads = getNumWorkingThreads();

    if (numThreads <= 1) {
        CLAYOUT_FOR_EACH_ORDER_ELT(elt) {
            IPO_Layout_Driver_elt(opt_level, elt);
        } CLAYOUT_END_FOR_EACH_ORDER_ELT;
    } else {
        int j, numElts;
        PTR* elts;

        numElts= getElts(numThreads, &elts);

#ifdef USE_OMP_TLS
        #pragma omp parallel for
#endif
        for (j = 0; j < numElts; j++) {
            PHASE2_TLS_Global_State_Set();
            IPO_Layout_Driver_elt(opt_level, elts[j]);
        }
    }
}
```

Ready | Ln 488 | Col 41 | Ch 41 | INS

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

- ~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

TIC Dependence Violations Summary on ref - Microsoft Internet E

File Edit View Favorites Tools Help

Address \\ssgfs002\cmplr\usr3\mhaghigh\tic\tic.html

Back Search Favorites Media

22	Aug 29	18:14	684	
21	Aug 29	10:01	723	
20	Aug 28	13:44	708	
19	Aug 26	15:55	712	
18	Aug 25	18:29	791	
17	Aug 25	15:07	889	
16	Aug 25	09:40	971	
15	Aug 24	22:39	1,021	
14	Aug 24	18:59	1,031	
13	Aug 24	12:41	1,502	
12	Aug 23	18:39	1,663	
11	Aug 23	16:02	1,815	
10	Aug 22	16:02	2,087	
9	Aug 21	18:10	3,437	
8	Aug 21	16:49	3,430	
7	Aug 17	10:34	4,435	
6	Aug 16	16:26	12,132	
5	Aug 15	11:53	12,514	
4	Aug 9	11:43	12,809	
3	Aug 4	13:59	15,713	
2	Aug 3	15:13	52,287	
1	Jul 21	11:34	294,487	

TIC Dependence Violations Summary on ref - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address \\ssgfs002\cmplr\usr3\mhaghigh\tic\tic.html Go

Back Search Favorites Media

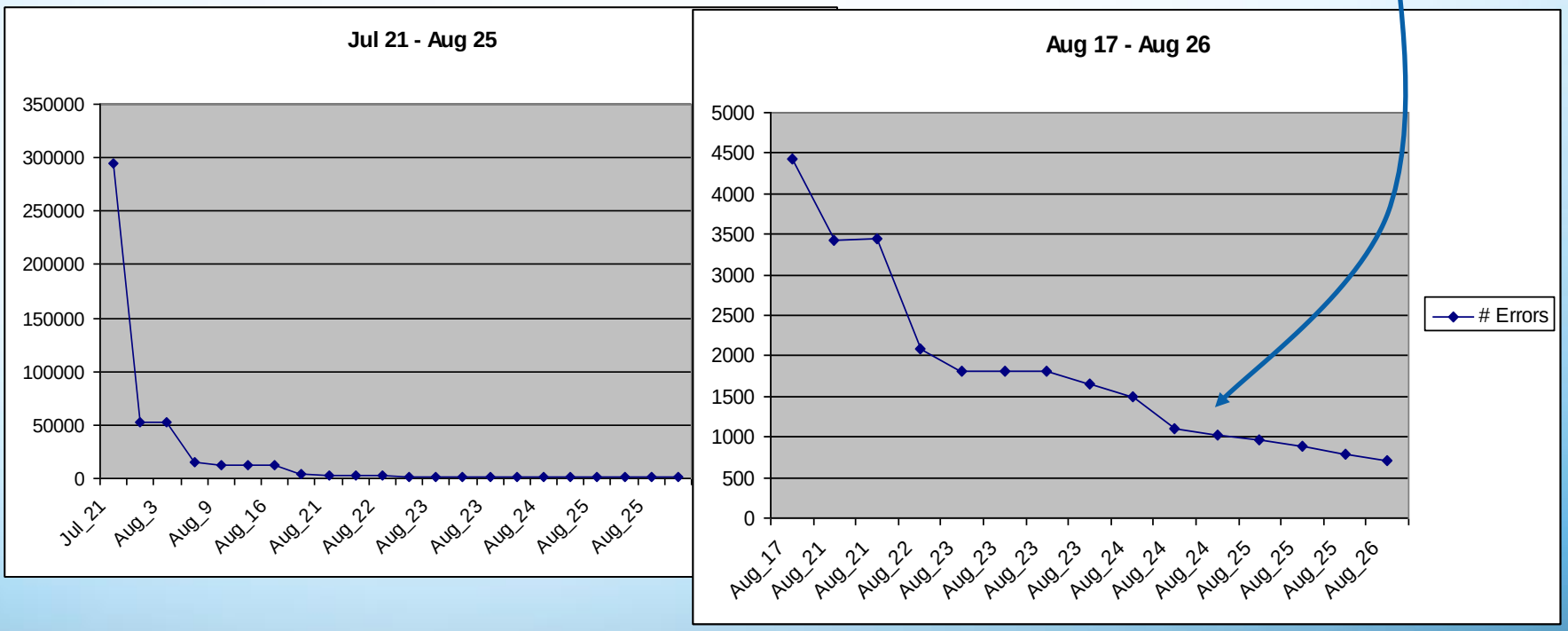
Run	Date	Time	Violations	Delta
40	Sep 15	15:44	73	
39	Sep 14	15:59	75	
38	Sep 9	13:39	95	8
37	Sep 7	17:37	117	
36	Sep 7	16:17	190	70
35	Sep 6	15:40	120	
34	Sep 6	15:02	124	13
33	Sep 6	13:35	181	
32	Sep 5	17:50	206	3
31	Sep 5	10:32	277	
30	Sep 3	16:51	278	
29	Sep 1	17:00	280	
28	Aug 31	21:56	356	
27	Aug 31	18:43	370	
26	Aug 31	16:14	622	
25	Aug 30	21:44	630	
24	Aug 30	18:48	654	
23	Aug 30	13:13	701	60

Internet

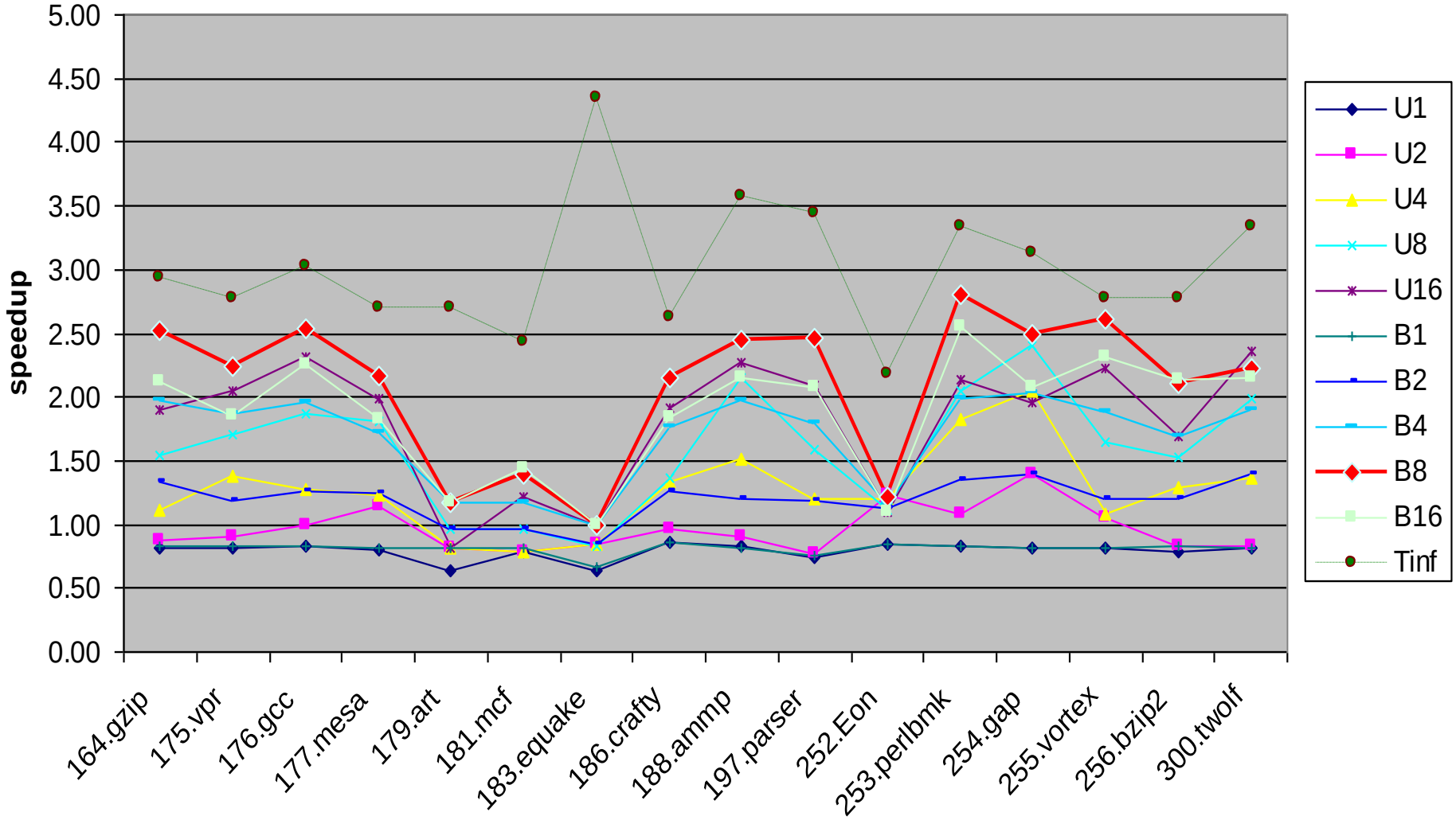
Reducing violations

- 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

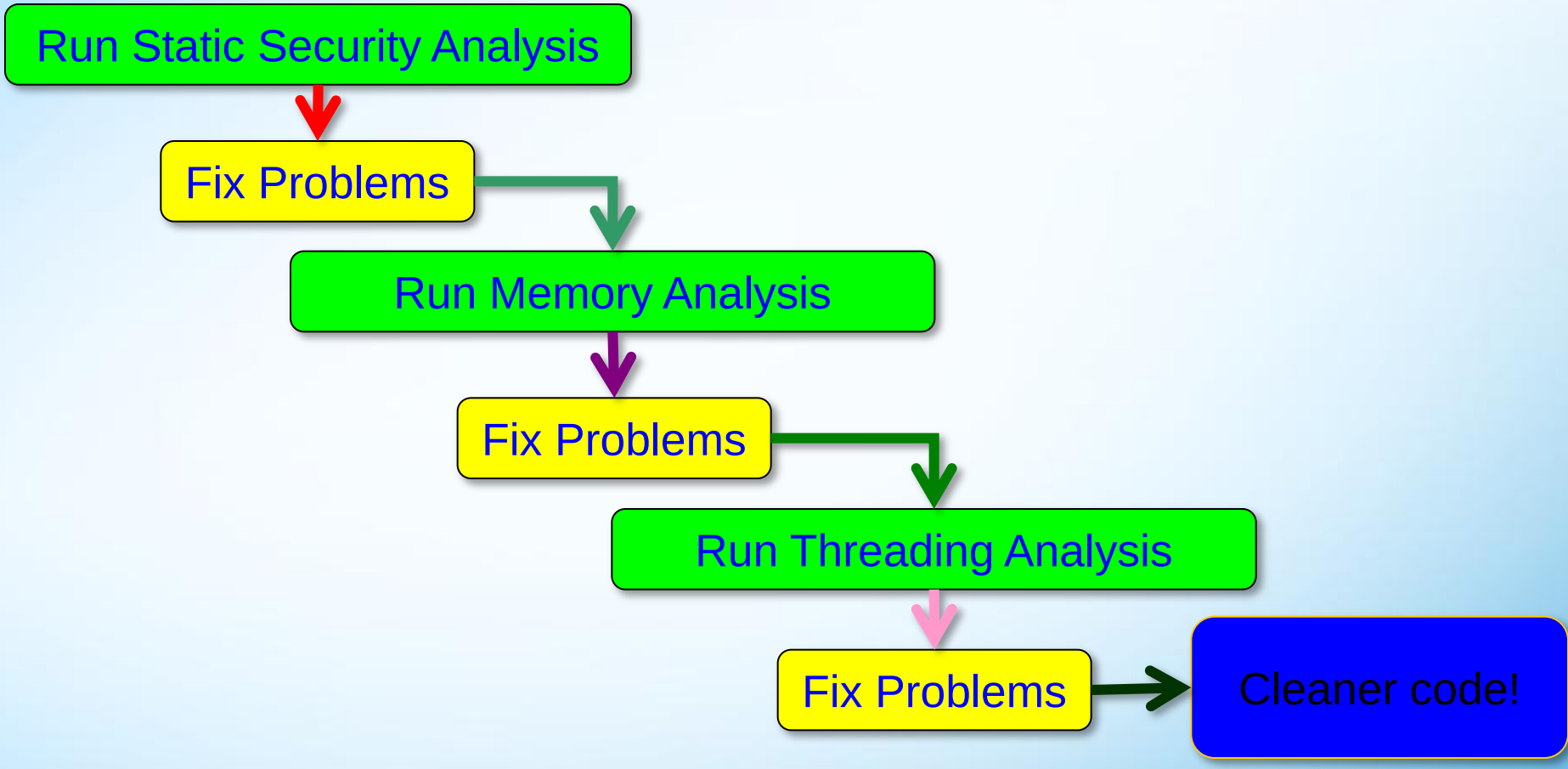
Scale down workload to speed up analysis!

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



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
  http://download.intel.com/technology/itj/2007/v11i4/6-tools/6-Methodology_Tools_and_Techniques.pdf

- "A Theory of Data Race Detection", Banerjee, Bliss, Ma, Petersen
  http://cs.ucsb.edu/~arch/spr07-seminar/papers/threadchecker06.pdf

Optimization Notice

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.

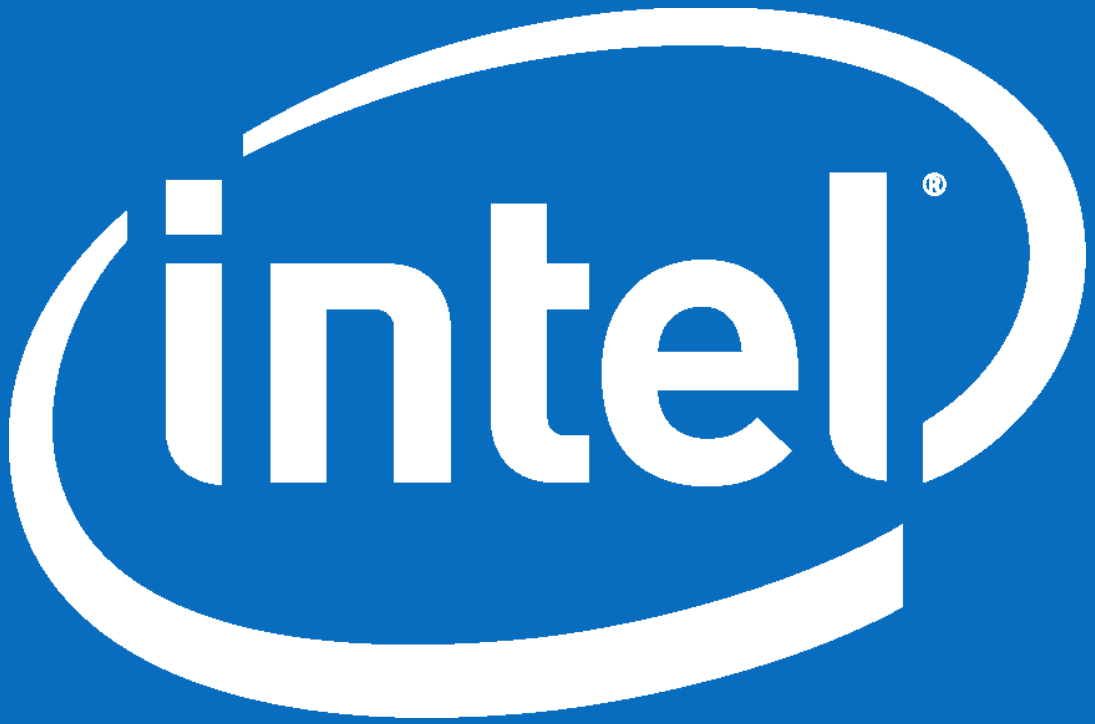
Intel compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel Streaming SIMD Extensions 2 (Intel SSE2), Intel Streaming SIMD 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

http://intel.com/software/products


intel