Tools for OpenMP Programming

PPCES 2018

Tim Cramer

IT Center

RWTH AACHEN UNIVERSITY

OpenMP Tools

- **ThreadSanitizer**
  - → Overview
- **Intel Inspector XE**
  - → Overview
  - → Live Demo
- **Intel VTune Amplifier XE**
  - → Overview
  - → Live Demo

Race Condition

- **Data Race: the typical OpenMP programming error, when:**
  - → two or more threads access the same memory location, and
  - → at least one of these accesses is a write, and
  - → the accesses are not protected by locks or critical regions, and
  - → the accesses are not synchronized, e.g. by a barrier.
- **Non-deterministic occurrence: e.g. the sequence of the execution of parallel loop iterations is non-deterministic and may change from run to run**
- **In many cases *private* clauses, *barriers* or *critical regions* are missing**
- **Data races are hard to find using a traditional debugger**
  - → Use the Intel Inspector XE or the ThreadSanitizer

OpenMP Tools

- **ThreadSanitizer**
  - → Overview
- **Intel Inspector XE**
  - → Overview
  - → Live Demo
- **Intel VTune Amplifier XE**
  - → Overview
  - → Live Demo

OpenMP Correctness checking: ThreadSanitizer – Overview

- Correctness checking for threaded applications
- Integrated in clang and gcc compiler
- Low runtime overhead: 2x – 15x
- Used to find data races in browsers like Chrome and Firefox

OpenMP Correctness checking: ThreadSanitizer – Usage

```
module load archer gcc/6
```

- Compile the program with clang compiler:

C

```
clang -fsanitize=thread -fopenmp -g myprog.c -o myprog
```

C++

```
clang++ -fsanitize=thread -fopenmp -g myprog.cpp
            -o myprog
```

Fortran

```
gfortran -fsanitize=thread -fopenmp -g myprog.f -c
clang -fsanitize=thread -fopenmp -lgfortran myprog.o
            -o myprog
```

- Execute:

```
OMP_NUM_THREADS=4 ./myprog
```

- Understand and correct the detected threading errors
- Edit the source code
- Repeat until no errors reported

OpenMP Correctness checking: ThreadSanitizer – Result Summary

```
 1 #include <stdio.h>
 2
 3 int main(int argc, char **argv) {
 4       int a = 0;
 5       #pragma omp parallel
 6       {
 7           if (a < 100) {
 8               #pragma omp critical
 9               a++;
10           }
11       }
12 }
```

WARNING: ThreadSanitizer: data race

- Read of size 4 at 0x7fffffffdcdc by thread T2:

  #0 .omp_outlined. race.c:7 (race+0x0000004a6dce)

  #1 __kmp_invoke_microtask <null> (libomp_tsan.so)

- Previous write of size 4 at 0x7fffffffdcdc by main thread:

  #0 .omp_outlined. race.c:9 (race+0x0000004a6e2c)

  #1 __kmp_invoke_microtask <null> (libomp_tsan.so)

OpenMP Tools

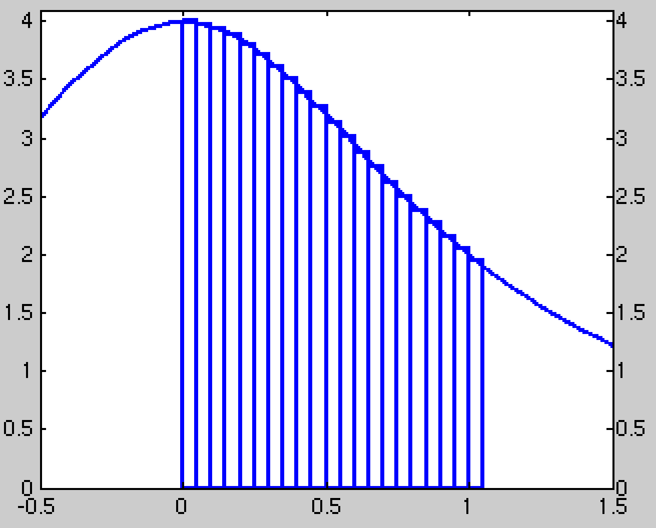
- **ThreadSanitizer**
  - → Overview
- **Intel Inspector XE**
  - → Overview
  - → Live Demo
- **Intel VTune Amplifier XE**
  - → Overview
  - → Live Demo

Intel Inspector XE

- **Detection of**
  - → Memory Errors
  - → Deadlocks
  - → Data Races
- **Support for**
  - → Linux (32bit and 64bit) and Windows (32bit and 64bit)
  - → WIN32-Threads, Posix-Threads, Intel Threading Building Blocks and OpenMP
- **Features**
  - → Binary instrumentation gives full functionality
  - → Independent stand-alone GUI for Windows and Linux

PI Example Code

$$\pi = \int_0^1 \frac{4}{1+x^2}$$




```
double f(double x)
{
   return (4.0 / (1.0 + x*x));
}

double CalcPi (int n)
{
   const double fH   = 1.0 / (double) n;
   double fSum = 0.0;
   double fX;
   int i;

#pragma omp parallel for private(fX,i) reduction(+:fSum)
   for (i = 0; i < n; i++)
   {
      fX = fH * ((double)i + 0.5);
      fSum += f(fX);
   }
   return fH * fSum;
}
```

PI Example Code

```
double f(double x)
{
   return (4.0 / (1.0 + x*x));
}

double CalcPi (int n)
{
   const double fH   = 1.0 / (double) n;
   double fSum = 0.0;
   double fX;
   int i;

#pragma omp parallel for private(fX,i) reduction(+:fSum)
   for (i = 0; i < n; i++)
   {
      fX = fH * ((double)i + 0.5);
      fSum += f(fX);
   }
   return fH * fSum;
}
```

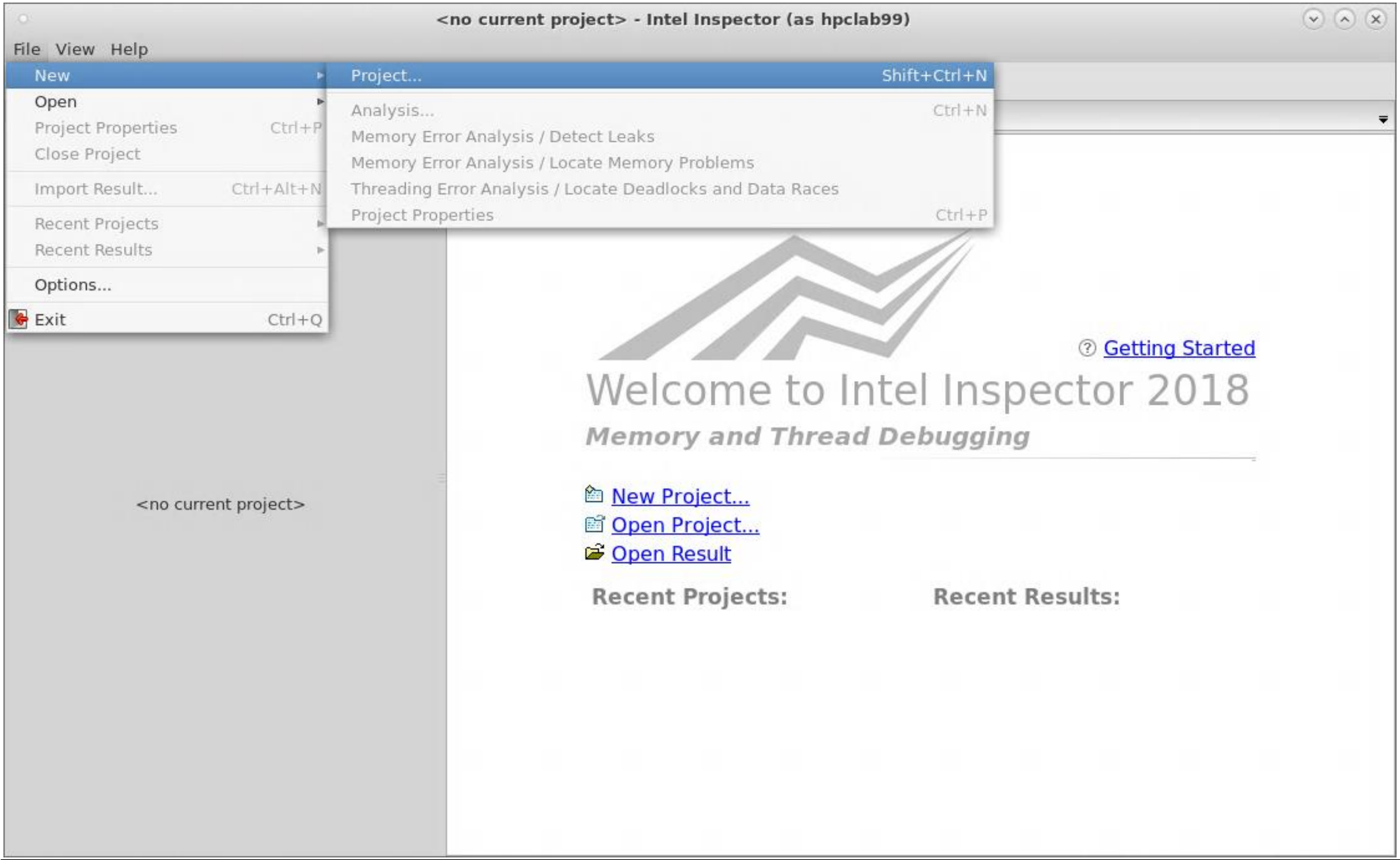
What if we would have forgotten this?

Live Demo

Intel Inspector XE

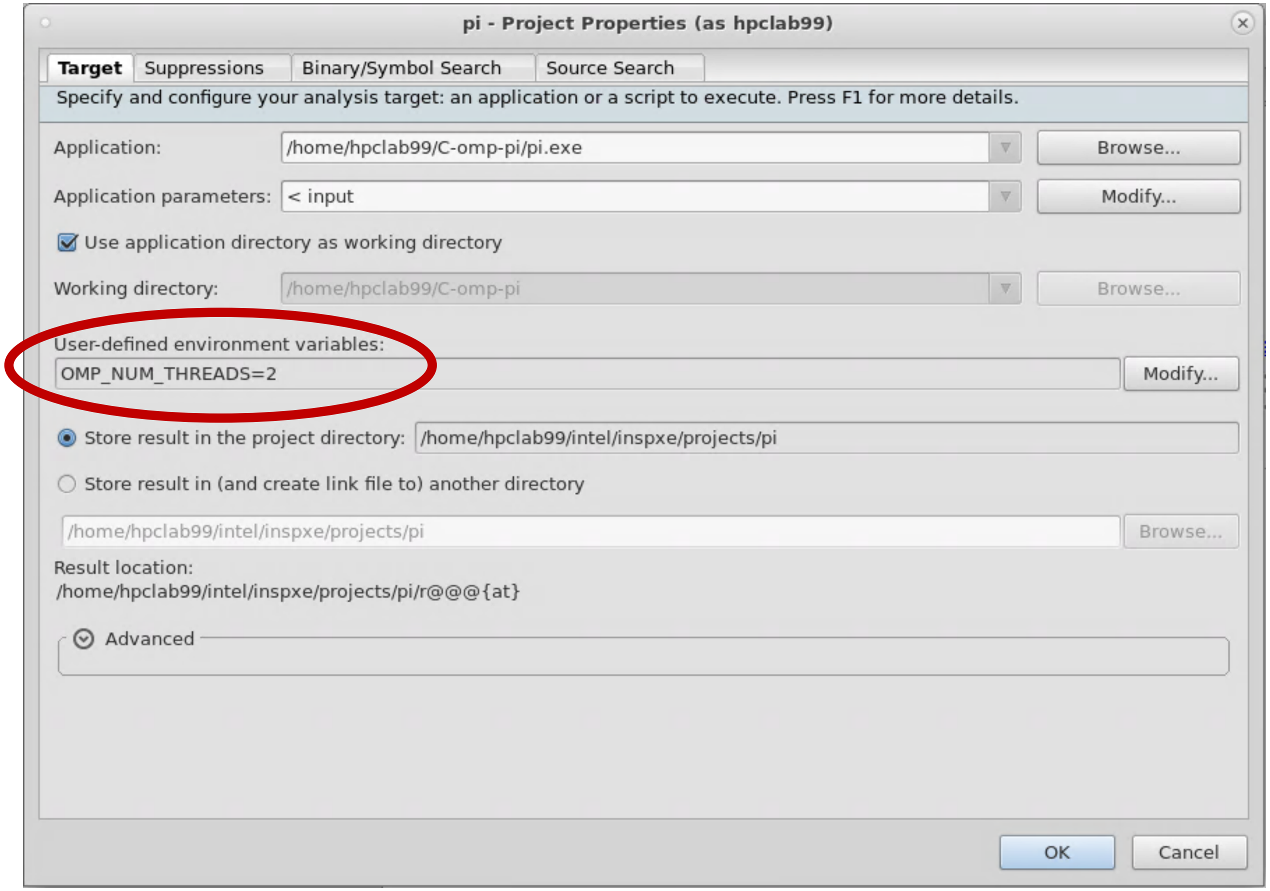
Inspector XE – Create Project

```
$ module load intelixe ; inspxe-gui
```



Inspector XE – Create Project

- ensure that multiple threads are used
- choose a small dataset (really!), execution time can increase 10X – 1000X

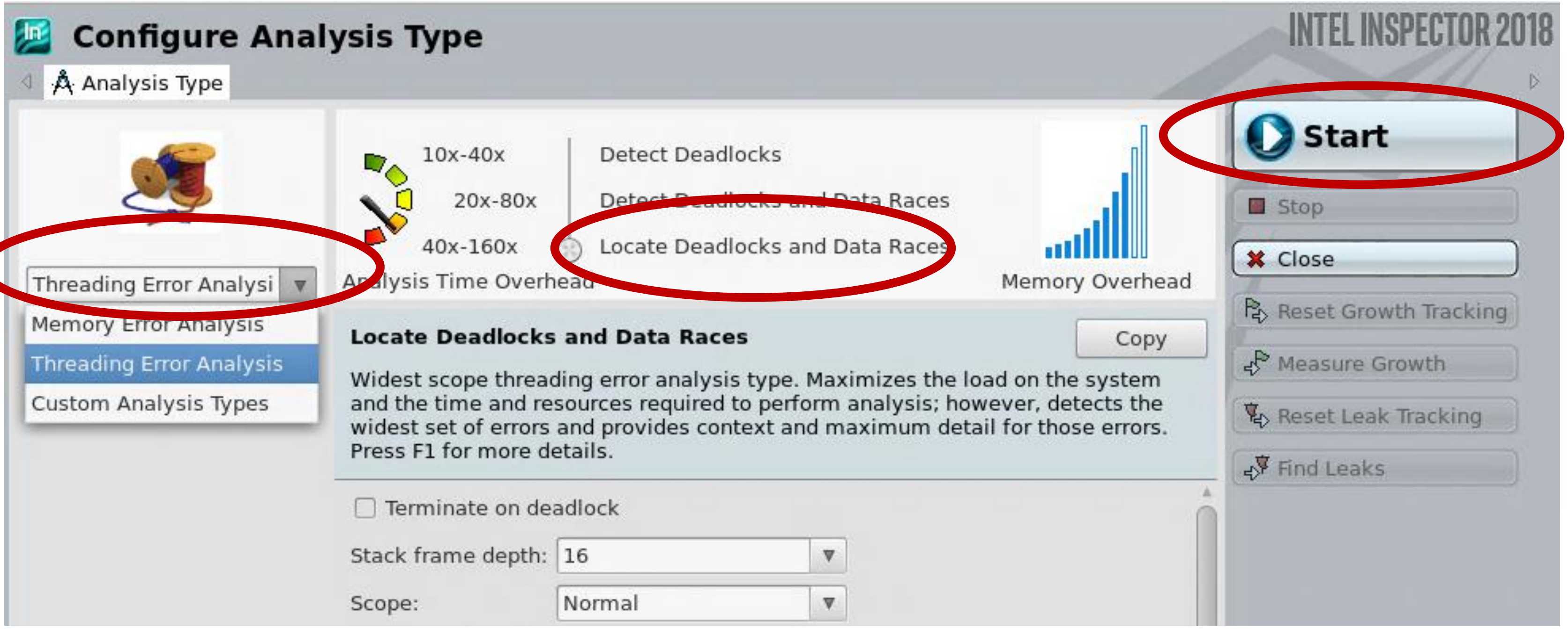


Inspector XE – Configure Analysis

Threading Error Analysis Modes

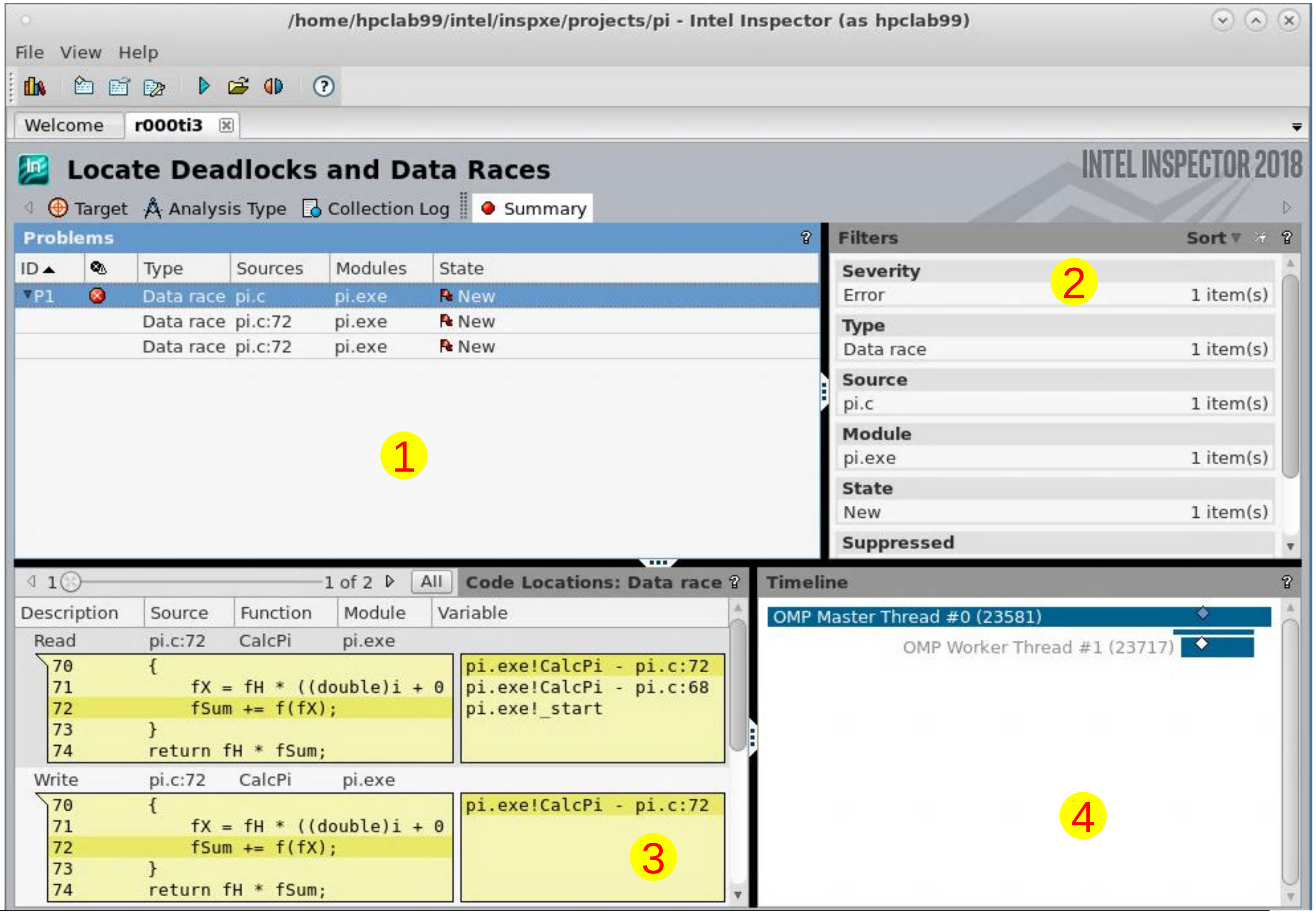
1. Detect Deadlocks
2. Detect Deadlocks and Data Races
3. Locate Deadlocks and Data Races

more details,
more overhead



Inspector XE – Results

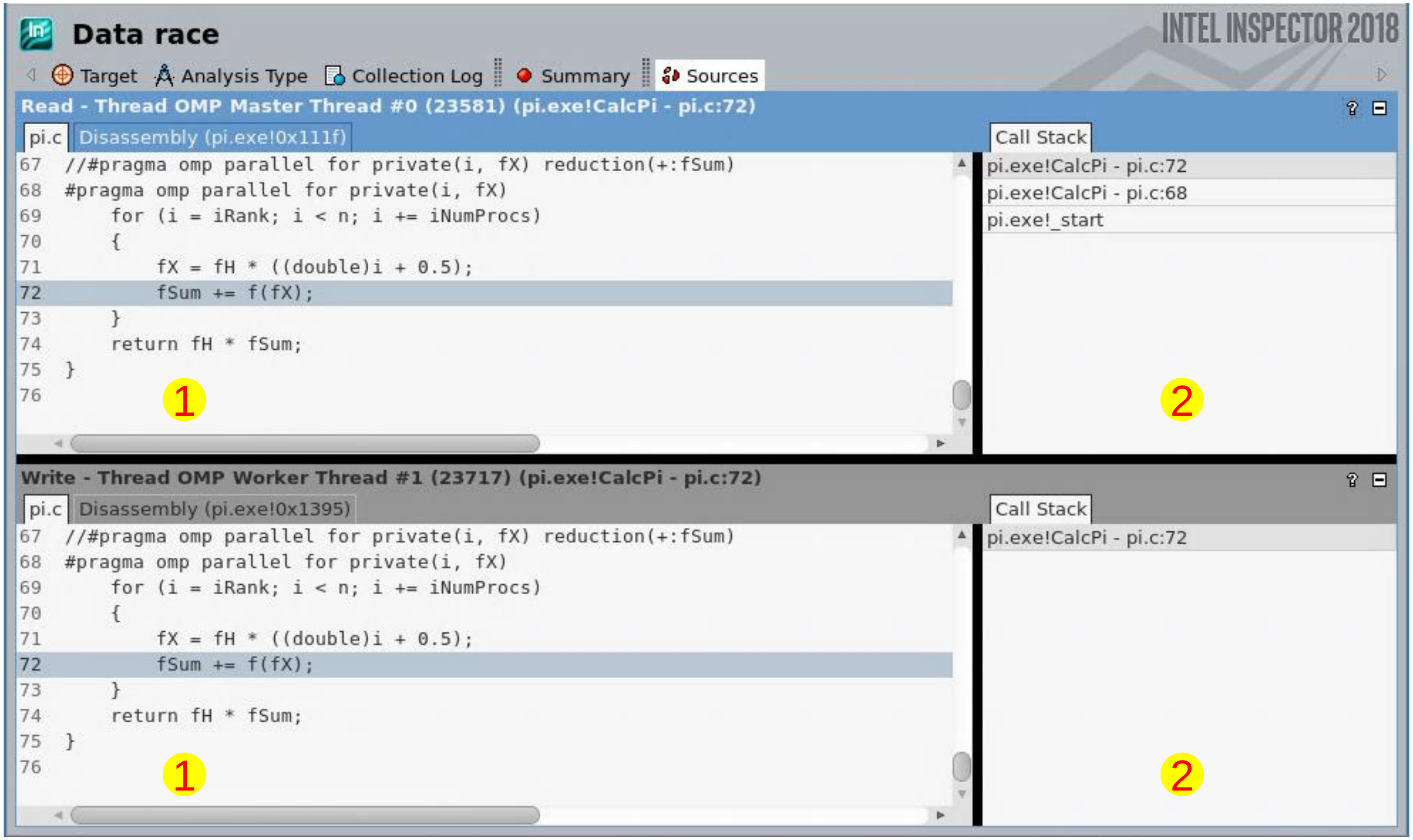
1 detected problems
2 filters
3 code location
4 Timeline



Inspector XE – Results

1 Source Code producing the issue – double click opens an editor

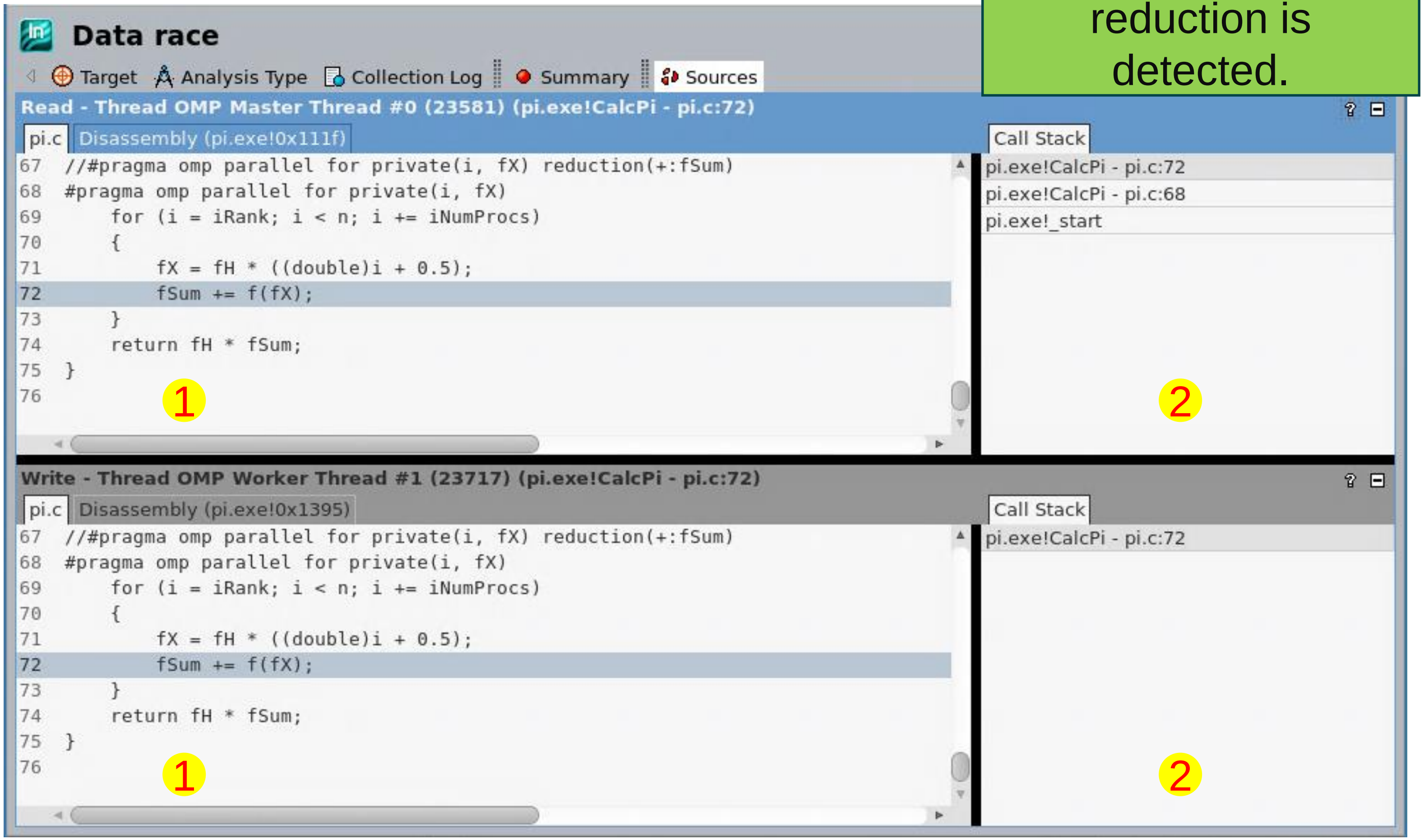
2 Corresponding Call Stack



Inspector XE – Results

1. Source Code producing the issue – double click opens an editor
2. Corresponding Call Stack

The missing reduction is detected.



Command Line Tool – inspxe-cl

Threading Error Analysis Modes

1. **Detect Deadlocks (ti1)**
2. **Detect Deadlocks and Data Races (ti2)**
3. **Locate Deadlocks and Data Races (ti3)**

$ inspxe-cl -collect ti3 -- pi.exe < input

Data collection without GUI allows to use batch jobs.

$ inspxe-cl -report problems ...

Viewing results in text mode is helpful, when remote connections are slow.

OpenMP Tools

- ThreadSanitizer
  - → Overview
- Intel Inspector XE
  - → Overview
  - → Live Demo
- **Intel VTune Amplifier XE**
  - → Overview
  - → Live Demo

Intel VTune Amplifier XE

- **Performance Analyses for**
  - → Serial Applications
  - → Shared Memory Parallel Applications
- **Sampling-based measurements**
- **Features:**
  - → Hot Spot Analysis
  - → Concurrency Analysis
  - → Wait
  - → Hardware Performance Counter Support
- **Prerequisites:**
  - → ssh -Y login-t
  - → module load intelvtune
- **Application**
  - → Use representative data set
  - → Build with debug information (-g) AND optimization (-O3)
  - → Set environment variables (e.g., OMP_NUM_THREADS, OMP_PROC_BIND, OMP_PLACES, etc.)

Stream

- **Standard Benchmark to measure memory performance.**
- **Version is parallelized with OpenMP.**

```
#pragma omp parallel for
        for (j=0; j<N; j++)
            b[j] = scalar*c[j];
```

Measures Memory bandwidth for:

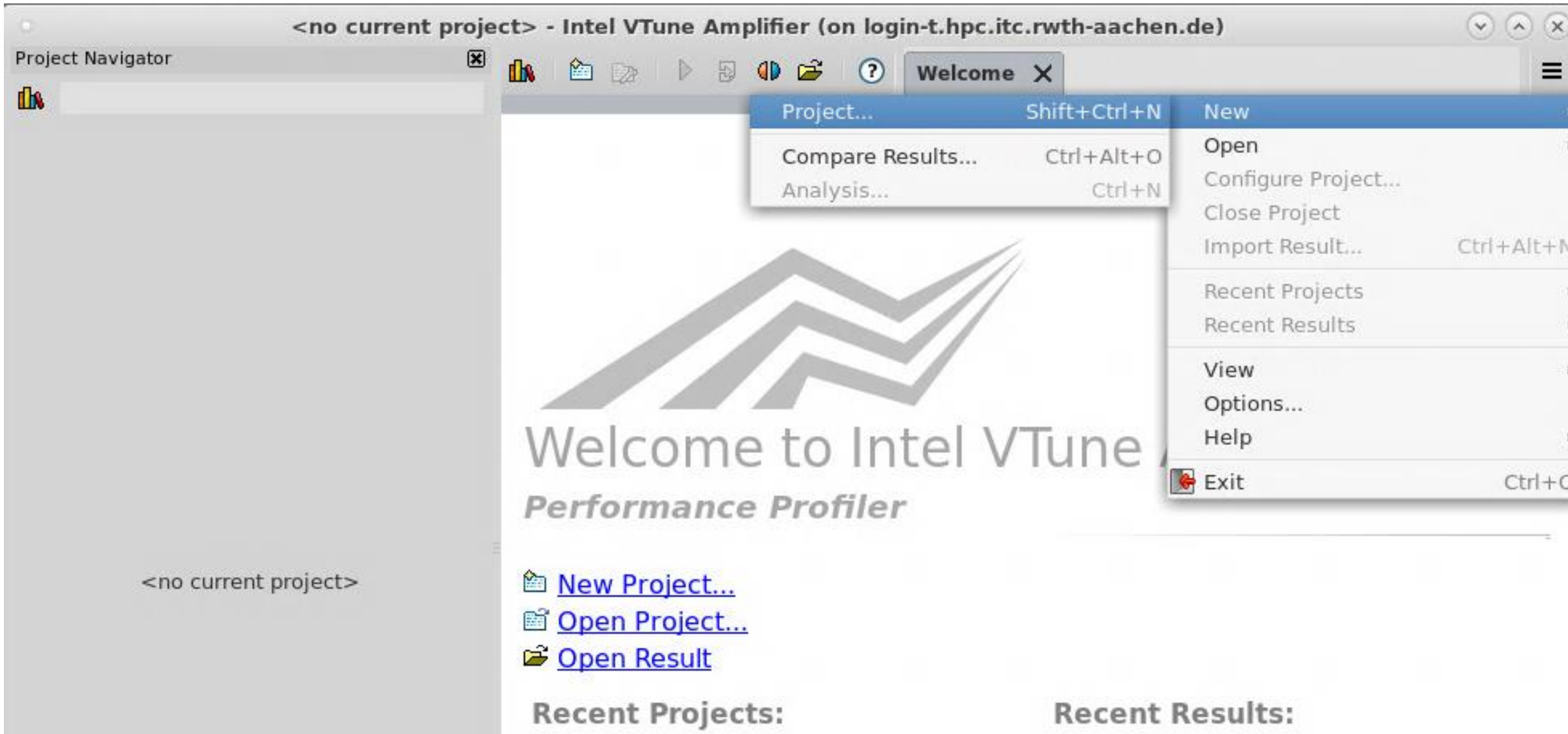
y=x (copy)
y=s*x (scale)
y=x+z (add)
y=x+s*z (triad)

for double vectors x,y,z and scalar double value s

```
-------------------------------------------------------------
Function      Rate (MB/s)    Avg time      Min time      Max
time
Copy:       121755.3877      0.0119        0.0118        0.0122
Scale:      122280.4397      0.0118        0.0118        0.0119
Add:        124754.8422      0.0173        0.0173        0.0176
Triad:      124797.8048      0.0174        0.0173        0.0178
```

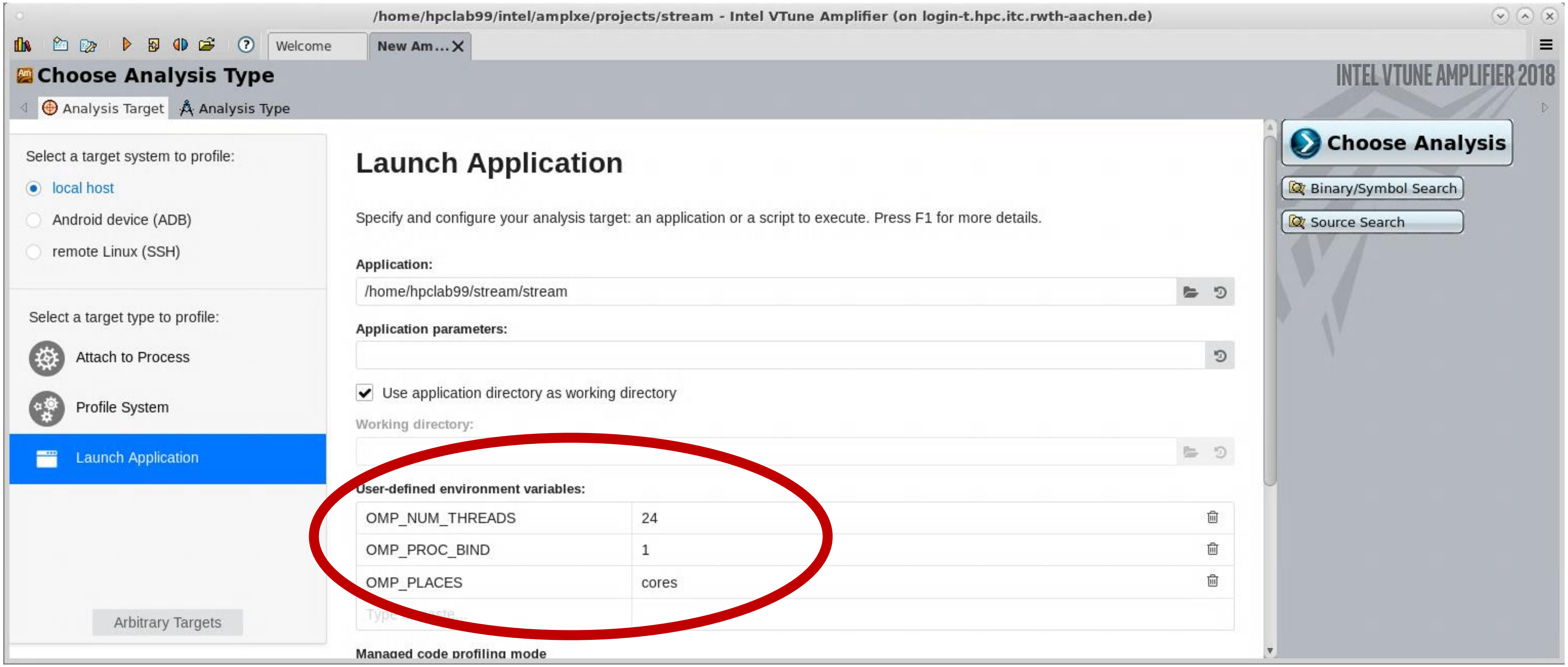
Intel VTune Amplifier XE – Create Project

- `$ module load intelvtune; amplxe-gui`
- Create a Project in the same way as with the inspector.
- Executable should be build with optimization.
- Use a reasonable sized data set.



Intel VTune Amplifier XE – Create Project

- **Set environment variables**



Amplifier XE – Algorithm Analysis

Different Analysis Types

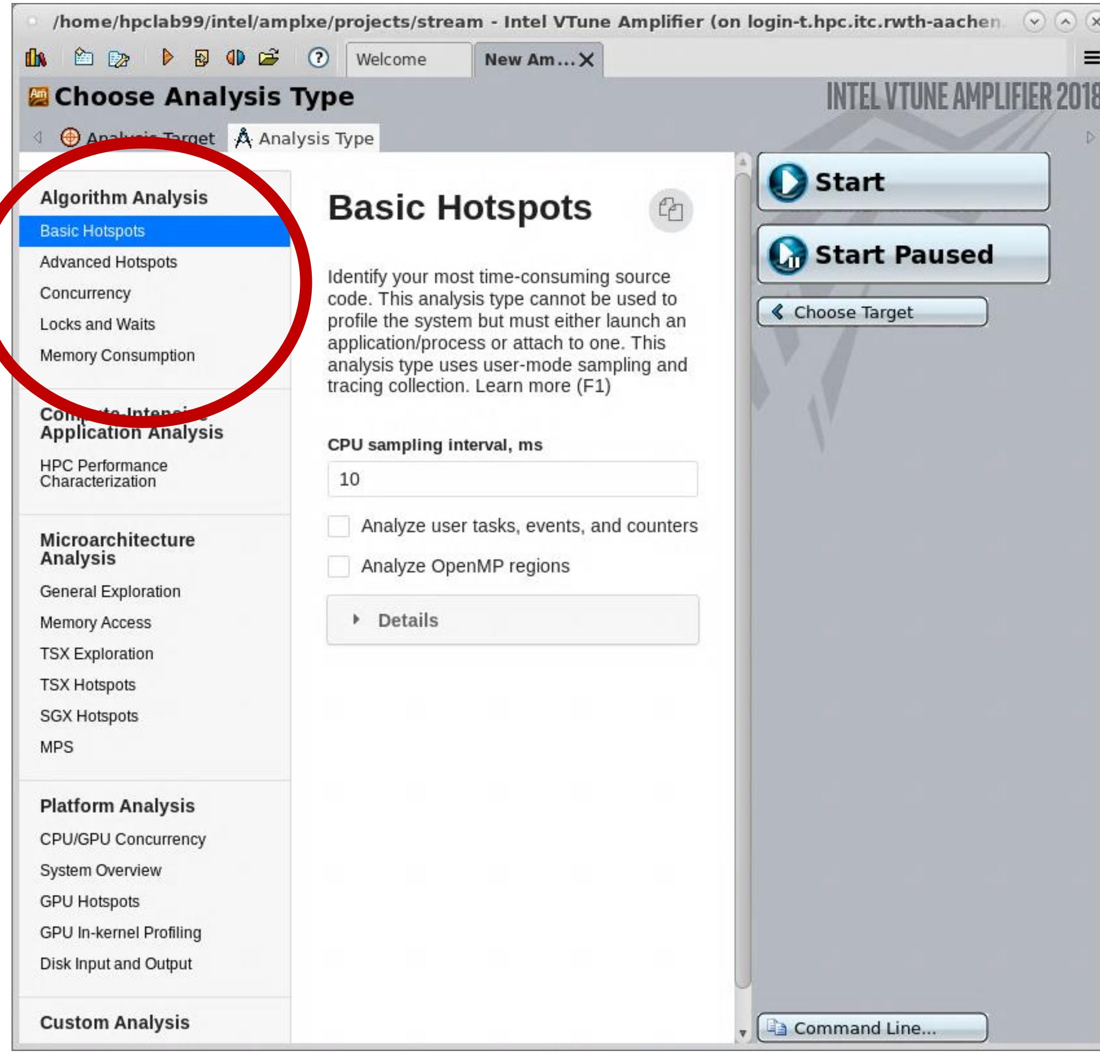
Basic Hotspots: Identify most time-consuming source code

Advanced Hotspots: Uses additional OS kernel support and analyses the CPI (Cycles Per Instruction) metric

Concurrency: Identify synchronization overhead and potential candidates for parallelism

Locks and Waits: Identify waiting times and discover how this affects your application performance

Memory consumption

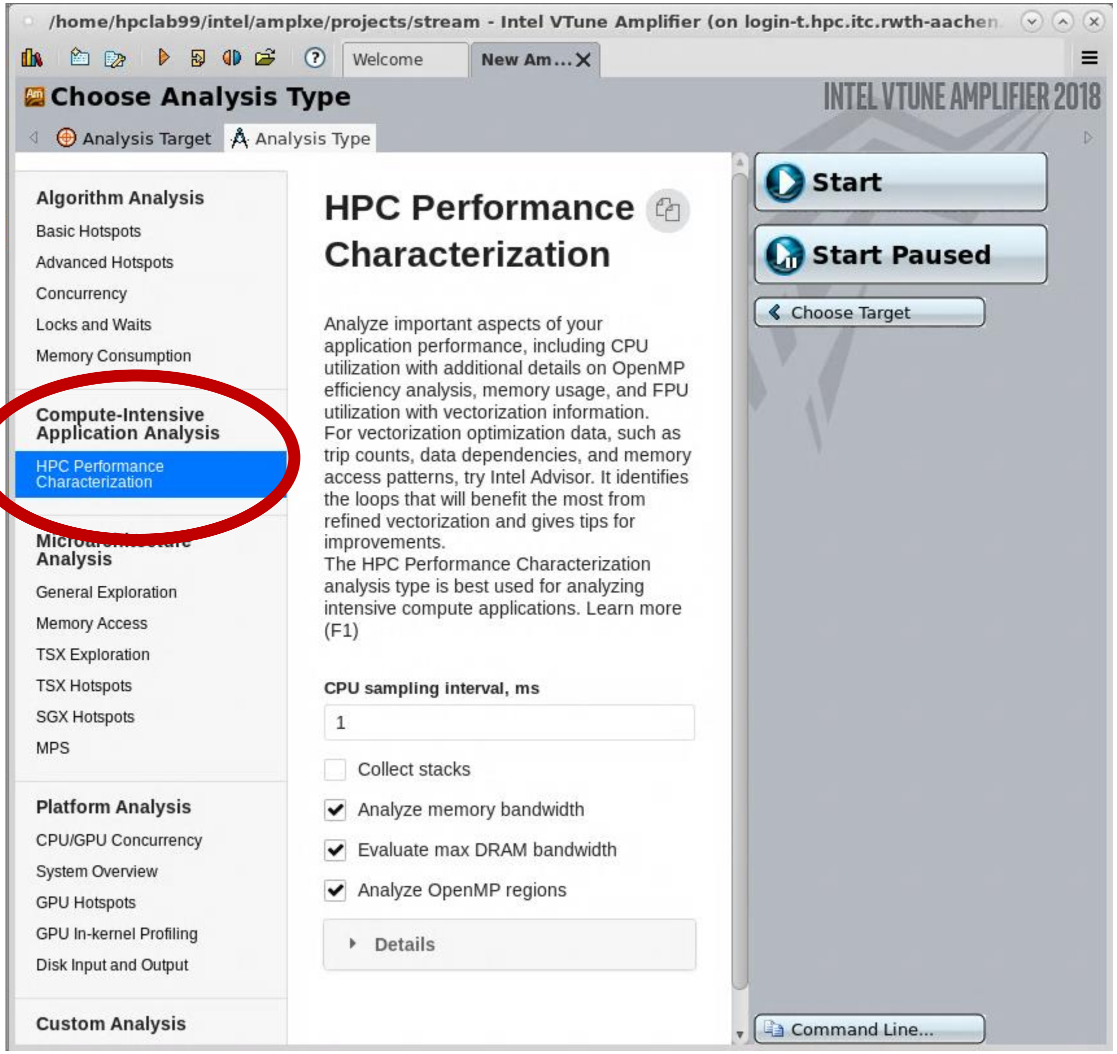


Amplifier XE – HPC Performance Characterization

Different Analysis Types

HPC Performance Characterization: Analyze

- CPU utilization
- OpenMP efficiency
- memory usage
- FPU utilization
- vectorization

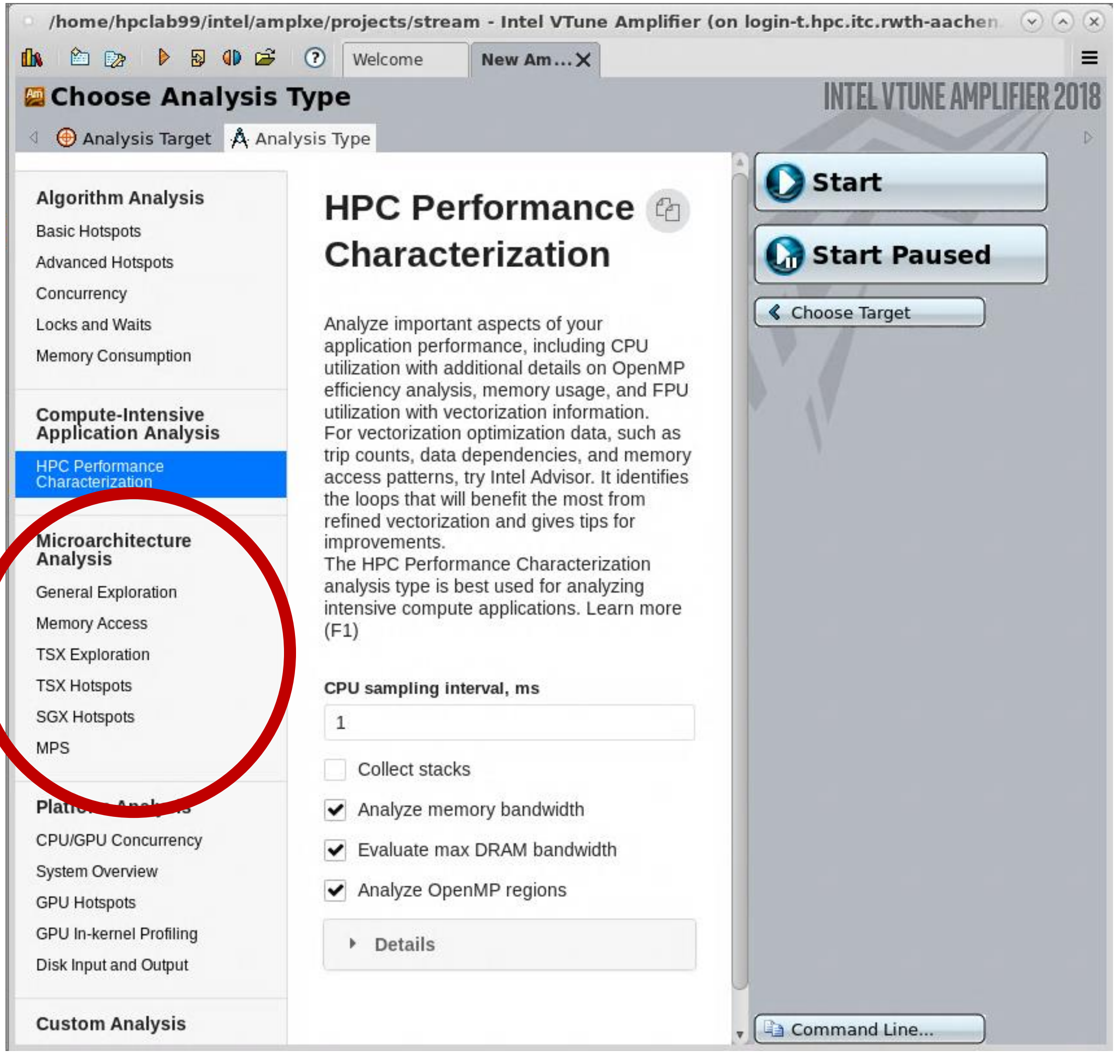


Amplifier XE – Microarchitecture Analysis

Different Analysis Types

Microarchitecture Analysis:

- Based on hardware sampling
- Identify memory access related issues (e.g., NUMA effects)
- Analyze Intel Transactional Synchronization Extensions (TSX)
- MPI Performance Snapshot (MPS)

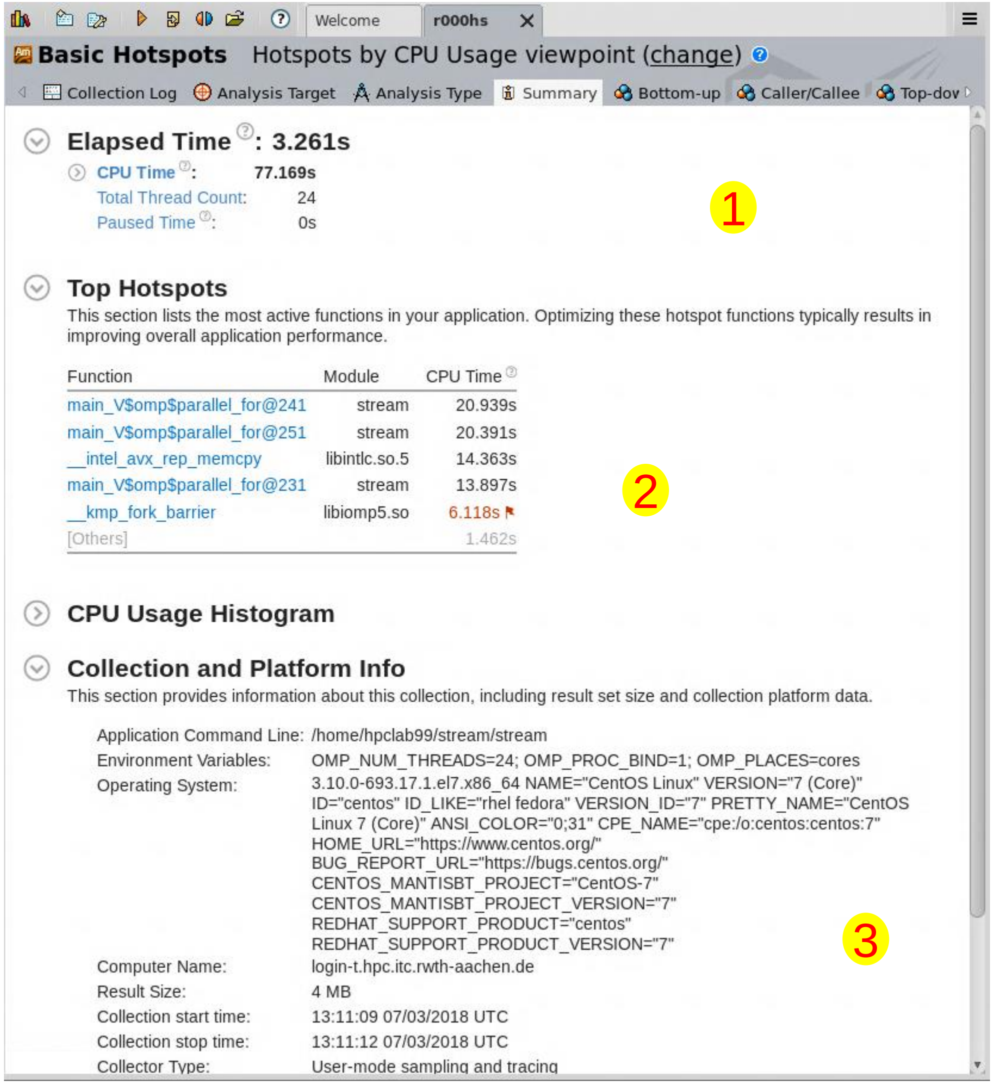


~~Live Demo~~

Intel VTune Amplifier XE

Warning: Intel Vtune Amplifier 2018 will crash the maschine, if HW performance counters are used. Please load module `intelvtune/XE2017-u00` instead.

Amplifier XE – Hotspot Analysis



Summary:

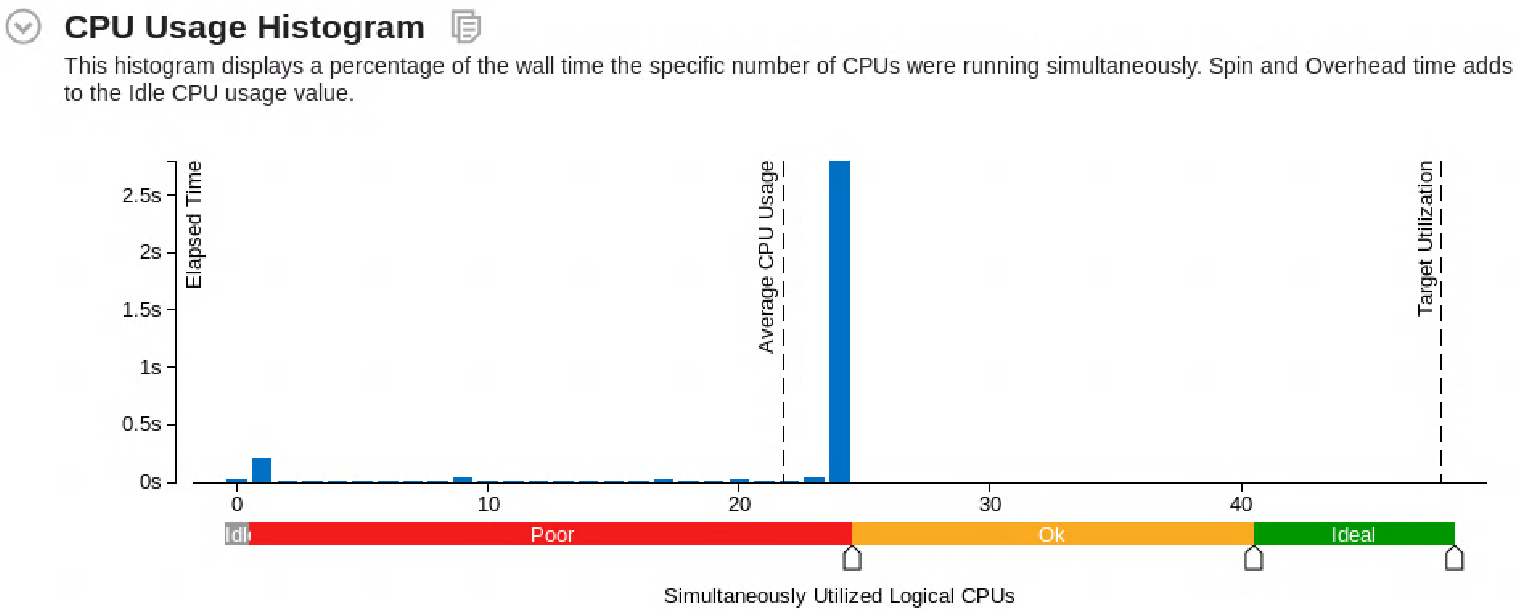
1. General Timing Information
2. Top Hotspots
3. Platform Information

Tool marks locations which might have a performance issue with a red flag.

Amplifier XE – Hotspot Analysis

CPU Usage Histogram

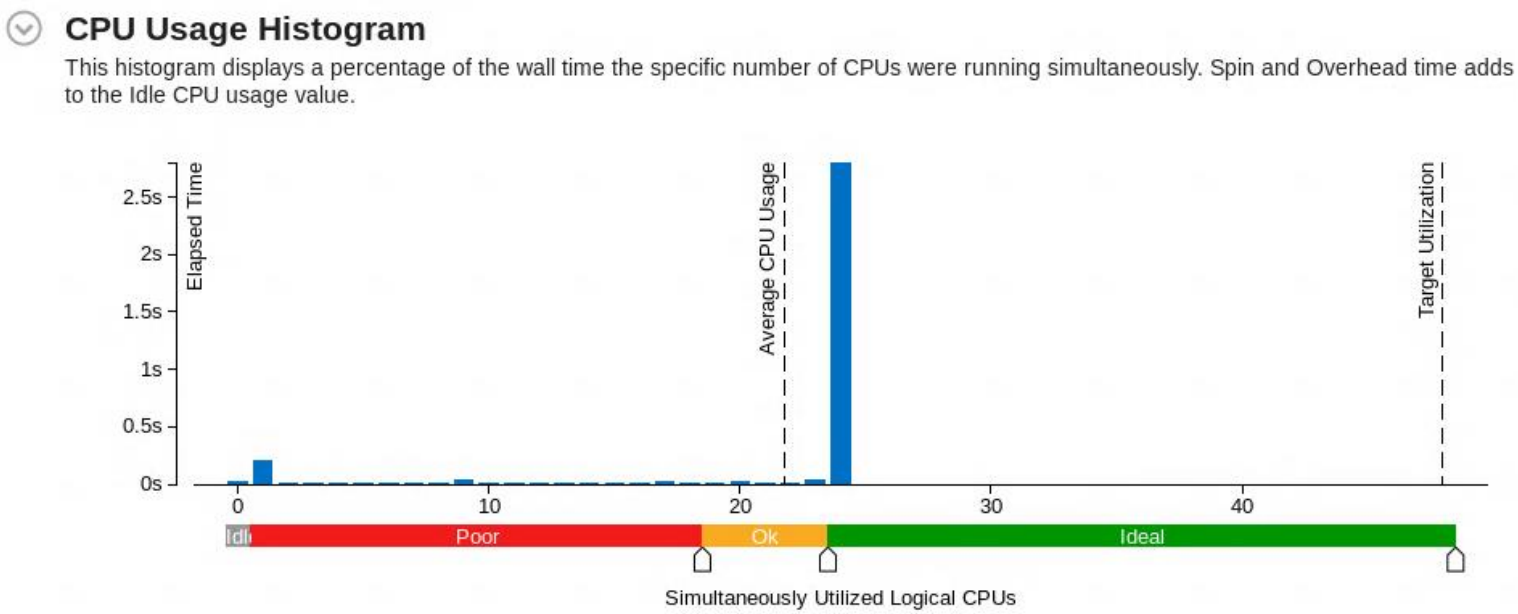
- **Show CPU Utilization**
- **Tool counts logical cores**
  - → Hyperthreads are not used in most HPC application
  - → Use slider to change



Amplifier XE – Hotspot Analysis

CPU Usage Histogram

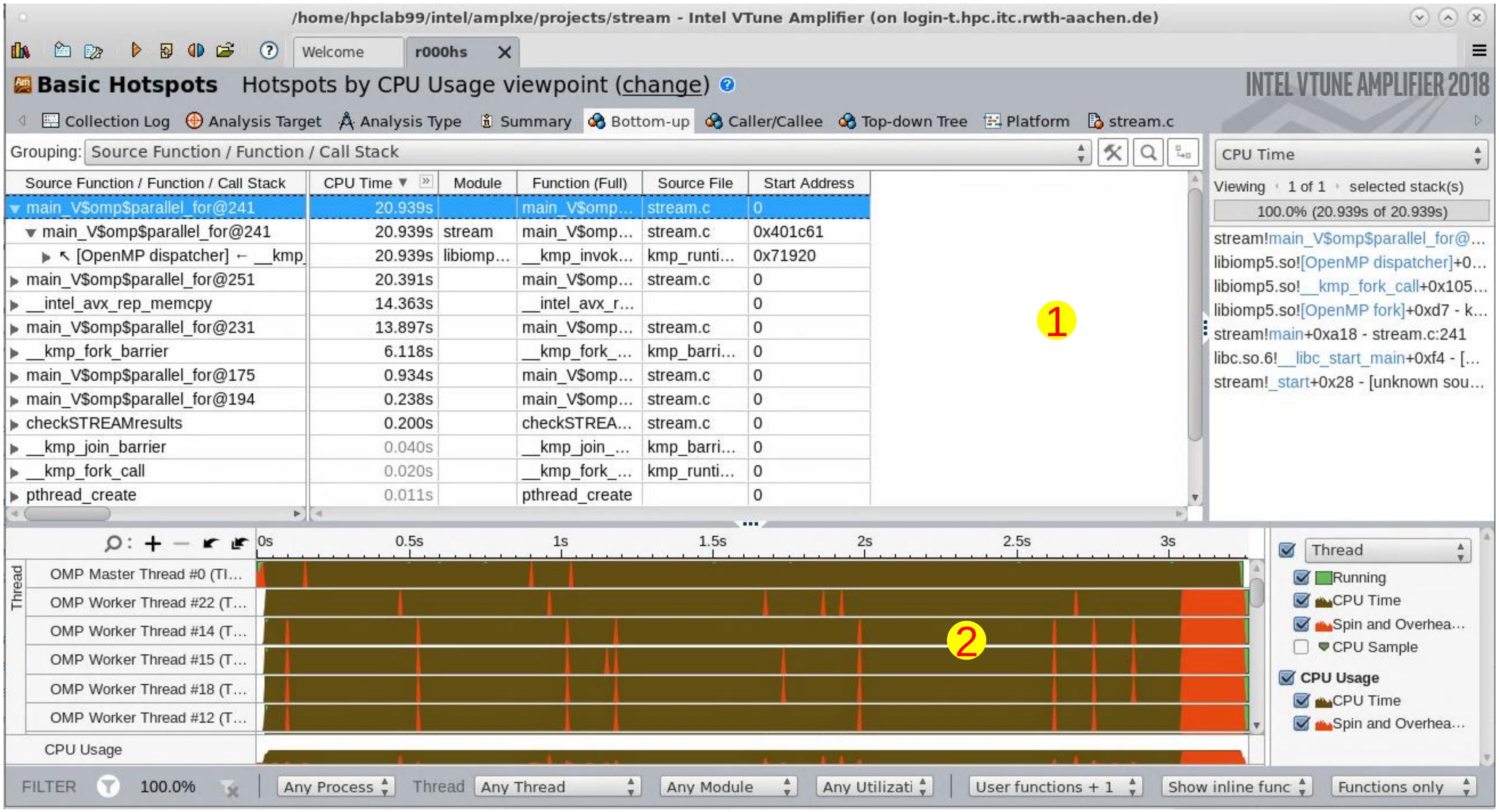
- **Show CPU Utilization**
- **Tool counts logical cores**
  - → Hyperthreads are not used in most HPC application
  - → Use slider to change



Amplifier XE – Hotspot Analysis

1 Function Summary

2 Timeline View



IT Center

RWTH AACHEN UNIVERSITY

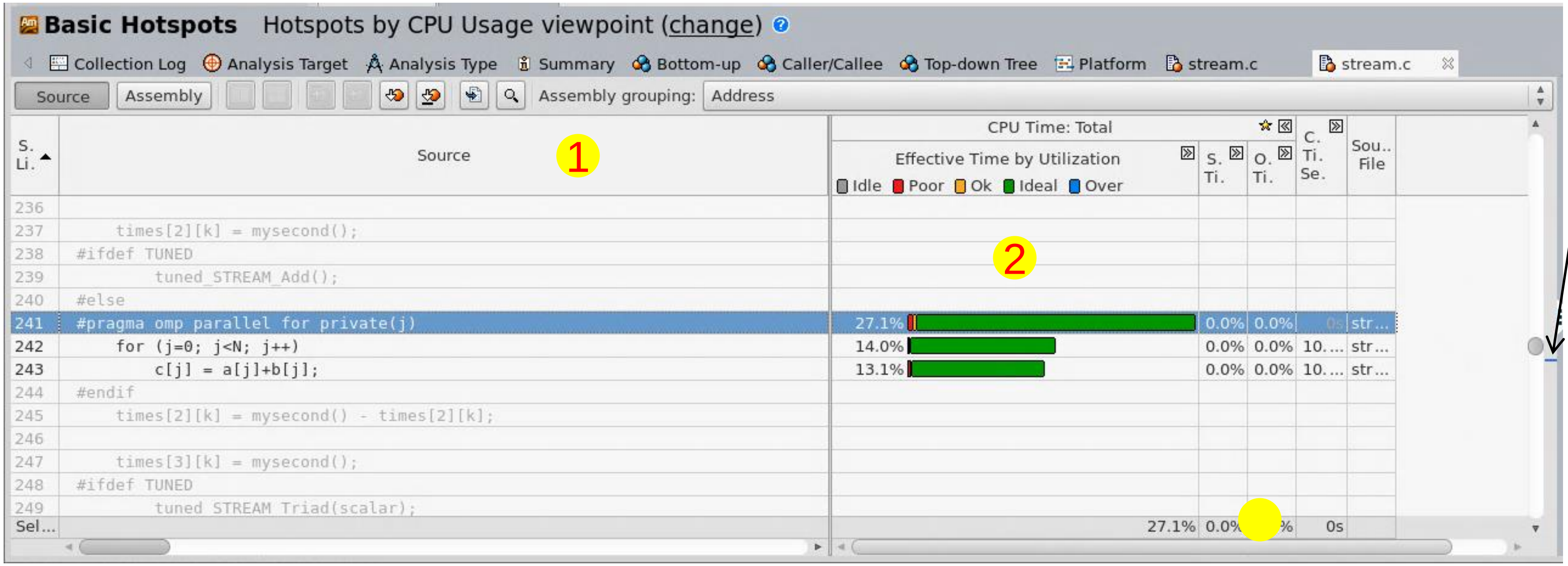
Amplifier XE – Hotspot Analysis

Double clicking on a function opens source code view.

1 Source Code View (only if compiled with -g)

2 CPU Utilization (would be all “poor”, if slider in histogram is unmodified)

Hotspots



Hardware Performance Counters

- **Hardware Counters of our Intel Nehalem Processor:**

L1I.HITS:
Counts all instruction fetches that hit the L1 instruction cache.

SB_DRAIN.ANY, STORE_BLOCKS.AT_RET, … S.STLB_HIT, DTLB_LOAD_MISSES.PDE_MIS,
DTLB_LOAD_MISSES.LARGE_W, MEM_INS … OPS_ISSUED.STALLED_CYCLE, UOPS_ISSUED.FUSED,
MEM_UNCORE_RETIRED.L3_D, MEM_UNC … RETIRED.UNCA, FP_COMP_OPS_EXE.X87,
FP_COMP_OPS_EXE.MMX, FP_COMP_OPS … E_SING, FP_COMP_OPS_EXE.SSE_DOU,
SIMD_INT_128.PACKED_MPY, SIMD_INT_1 … T_128.SHUFFLE_MOV, LOAD_DISPATCH.RS,
LOAD_DISPATCH.RS_DELAYED, LOAD_DIS … DED, INST_QUEUE_WRITE_CYCLES, LSD_OVERFLOW,
L2_RQSTS.LD_HIT, L2_RQSTS.LD_MISS, L2 … TS.IFETCHES, L2_RQSTS.PREFETCH_HIT,
L2_RQSTS.PREFETCH_MISS, L2_RQSTS.P … M_, L2_DATA_RQSTS.DEMAND.ME,
L2_DATA_RQSTS.PREFETCH.I_, L2_DATA … STATE, L2_WRITE.RFO.S_STATE,
L2_WRITE.RFO.M_STATE, L2_WRITE.RFO. … WRITE.LOCK.HIT, L2_WRITE.LOCK.MESI,
L1D_WB_L2.I_STATE, L1D_WB_L2.S_STATE, L1D_WB_L2.E_STATE, L1D_WB_L2.M_STATE, L1D_WB_L2.MESI, CPU_CLK_UNHALTED.THREAD, CPU_CLK_UNHALTED.REF_P, L1D_CACHE_LD.I_STATE, L1D_CACHE_LD.S_STATE,
L1D_CACHE_LD.E_STATE, L1D_CACHE_LD.M_STATE, L1D_CACHE_LD.MESI, L1D_CACHE_ST.S_STATE, L1D_CACHE_ST.E_STATE, L1D_CACHE_ST.M_STATE, L1D_CACHE_LOCK.HIT, L1D_CACHE_LOCK.S_STATE,
L1D_CACHE_LOCK.E_STATE, L1D_CACHE_LOCK.M_STATE, L1D_ALL_REF.ANY, L1D_ALL_REF.CACHEABLE, DTLB_MISSES.ANY, DTLB_MISSES.WALK_COMPLET, DTLB_MISSES.STLB_HIT, DTLB_MISSES.PDE_MISS,
DTLB_MISSES.LARGE_WALK_C, LOAD_HIT_PRE, L1D_PREFETCH.REQUESTS, L1D_PREFETCH.MISS, L1D_PREFETCH.TRIGGERS, L1D.M_REPL, L1D.M_EVICT, L1D.M_SNOOP_EVICT, L1D_CACHE_PREFETCH_LOCK,
L1D_CACHE_LOCK_FB_HIT, CACHE_LOCK_CYCLES.L1D_L2, CACHE_LOCK_CYCLES.L1D, IO_TRANSACTIONS, L1I.CYCLES_STALLED, LARGE_ITLB.HIT, ITLB_MISSES.ANY, ITLB_MISSES.WALK_COMPLET, ILD_STALL.LCP, ILD_STALL.MRU,
ILD_STALL.IQ_FULL, ILD_STALL.REGEN, ILD_STALL.ANY, BR_INST_EXEC.COND, BR_INST_EXEC.DIRECT, BR_INST_EXEC.INDIRECT_NON, BR_INST_EXEC.NON_CALLS, BR_INST_EXEC.RETURN_NEA, BR_INST_EXEC.DIRECT_NEAR,
BR_INST_EXEC.INDIRECT_NEA, BR_INST_EXEC.NEAR_CALLS, BR_INST_EXEC.TAKEN, BR_MISP_EXEC.COND, BR_MISP_EXEC.DIRECT, BR_MISP_EXEC.INDIRECT_NO, BR_MISP_EXEC.NON_CALLS, BR_MISP_EXEC.RETURN_NEA,
BR_MISP_EXEC.DIRECT_NEAR, BR_MISP_EXEC.INDIRECT_NEA, BR_MISP_EXEC.NEAR_CALLS, BR_MISP_EXEC.TAKEN, RESOURCE_STALLS.ANY, RESOURCE_STALLS.LOAD, RESOURCE_STALLS.RS_FULL, RESOURCE_STALLS.STORE,
RESOURCE_STALLS.ROB_FULL, RESOURCE_STALLS.FPCW, RESOURCE_STALLS.MXCSR, RESOURCE_STALLS.OTHER, MACRO_INSTS.FUSIONS_DECO, BACLEAR_FORCE_IQ, ITLB_FLUSH, OFFCORE_REQUESTS.L1D_WR,
UOPS_EXECUTED.PORT0, UOPS_EXECUTED.PORT1, UOPS_EXECUTED.PORT2_COR, UOPS_EXECUTED.PORT3_COR, UOPS_EXECUTED.PORT4_COR, UOPS_EXECUTED.PORT5, UOPS_EXECUTED.PORT015,
UOPS_EXECUTED.PORT234, OFFCORE_REQUESTS_SQ_FUL, OFF_CORE_RESPONSE_0, SNOOP_RESPONSE.HIT, SNOOP_RESPONSE.HITE, SNOOP_RESPONSE.HITM, OFF_CORE_RESPONSE_1, INST_RETIRED.ANY_P,
INST_RETIRED.X87, INST_RETIRED.MMX, UOPS_RETIRED.ANY, UOPS_RETIRED.RETIRE_SLOTS, UOPS_RETIRED.MACRO_FUSE, MACHINE_CLEARS.CYCLES, MACHINE_CLEARS.MEM_ORDE, MACHINE_CLEARS.SMC,
BR_INST_RETIRED.ALL_BRAN, BR_INST_RETIRED.CONDITION, BR_INST_RETIRED.NEAR_CAL, BR_MISP_RETIRED.ALL_BRAN, BR_MISP_RETIRED.NEAR_CAL, SSEX_UOPS_RETIRED.PACKED, SSEX_UOPS_RETIRED.SCALAR,
SSEX_UOPS_RETIRED.PACKED, SSEX_UOPS_RETIRED.SCALAR, SSEX_UOPS_RETIRED.VECTOR, ITLB_MISS_RETIRED, MEM_LOAD_RETIRED.L1D_HIT, MEM_LOAD_RETIRED.L2_HIT, MEM_LOAD_RETIRED.L3_UNS,
MEM_LOAD_RETIRED.OTHER_, MEM_LOAD_RETIRED.L3_MISS, MEM_LOAD_RETIRED.HIT_LFB, MEM_LOAD_RETIRED.DTLB_MI, FP_MMX_TRANS.TO_FP, FP_MMX_TRANS.TO_MMX, FP_MMX_TRANS.ANY, MACRO_INSTS.DECODED,
UOPS_DECODED.MS, UOPS_DECODED.ESP_FOLDING, UOPS_DECODED.ESP_SYNC, RAT_STALLS.FLAGS, RAT_STALLS.REGISTERS, RAT_STALLS.ROB_READ_POR, RAT_STALLS.SCOREBOARD, RAT_STALLS.ANY,
SEG_RENAME_STALLS, ES_REG_RENAMES, UOP_UNFUSION, BR_INST_DECODED, BPU_MISSED_CALL_RET, BACLEAR.BAD_TARGET, BPU_CLEARS.EARLY, BPU_CLEARS.LATE, L2_TRANSACTIONS.LOAD, L2_TRANSACTIONS.RFO,
L2_TRANSACTIONS.IFETCH, L2_TRANSACTIONS.PREFETCH, L2_TRANSACTIONS.L1D_WB, L2_TRANSACTIONS.FILL, L2_TRANSACTIONS.WB, L2_TRANSACTIONS.ANY, L2_LINES_IN.S_STATE, L2_LINES_IN.E_STATE, L2_LINES_IN.ANY,
L2_LINES_OUT.DEMAND_CLEA, L2_LINES_OUT.DEMAND_DIRT, L2_LINES_OUT.PREFETCH_CLE, L2_LINES_OUT.PREFETCH_DIR, L2_LINES_OUT.ANY, SQ_MISC.SPLIT_LOCK, SQ_FULL_STALL_CYCLES, FP_ASSIST.ALL,
FP_ASSIST.OUTPUT, FP_ASSIST.INPUT, SIMD_INT_64.PACKED_MPY, SIMD_INT_64.PACKED_SHIFT, SIMD_INT_64.PACK, SIMD_INT_64.UNPACK, SIMD_INT_64.PACKED_LOGICA, CPUID, SIMD_INT_64.PACKED_ARITH,
SIMD_INT_64.SHUFFLE_MOVE, UNC_GQ_CYCLES_FULL.READ_, UNC_GQ_CYCLES_FULL.WRITE, UNC_GQ_CYCLES_FULL.PEER_, UNC_GQ_CYCLES_NOT_EMPTY, UNC_GQ_CYCLES_NOT_EMPTY, UNC_GQ_CYCLES_NOT_EMPTY,
UNC_GQ_ALLOC.READ_TRACK, UNC_GQ_ALLOC.RT_L3_MISS, UNC_GQ_ALLOC.RT_TO_L3_RE, UNC_GQ_ALLOC.RT_TO_RTID_, UNC_GQ_ALLOC.WT_TO_RTID, UNC_GQ_ALLOC.WRITE_TRAC, UNC_GQ_ALLOC.PEER_PROBE,

BR_MISP_EXEC.COND:
Counts the number of mispredicted conditional near branch instructions executed, but not necessarily retired.

UNC_GQ_DATA.FROM_QPI, UNC_GQ_DAT … , UNC_GQ_DATA.TO_L3, UNC_GQ_DATA.TO_CORES,
UNC_SNP_RESP_TO_LOCAL_H, UNC_SNP … TO_LOCAL_H, UNC_SNP_RESP_TO_REMOTE,
UNC_SNP_RESP_TO_REMOTE, UNC_SNP … REMOTE, UNC_L3_HITS.READ, UNC_L3_HITS.WRITE,
UNC_L3_HITS.PROBE, UNC_L3_HITS.ANY, … STATE, UNC_L3_LINES_IN.S_STATE,
UNC_L3_LINES_IN.F_STATE, UNC_L3_LINE … NC_L3_LINES_OUT.F_STATE,
UNC_L3_LINES_OUT.ANY, UNC_QHL_REQ … LOCAL_, UNC_QHL_REQUESTS.LOCAL_,
UNC_QHL_CYCLES_FULL.IOH, UNC_QHL_ … S_NOT_EMPT, UNC_QHL_OCCUPANCY.IOH,
UNC_QHL_OCCUPANCY.REMOT, UNC_QH … NFLICT_CYCLES., UNC_QHL_CONFLICT_CYCLES.,
UNC_QHL_TO_QMC_BYPASS, UNC_QMC_ … ULL.WRI, UNC_QMC_NORMAL_FULL.WRI,
UNC_QMC_ISOC_FULL.READ.C, UNC_QMC … C_FULL.WRITE.C, UNC_QMC_BUSY.READ.CH0,
UNC_QMC_BUSY.READ.CH1, UNC_QMC_B … , UNC_QMC_OCCUPANCY.CH1,
UNC_QMC_OCCUPANCY.CH2, UNC_QMC_ … RMAL_READS.C, UNC_QMC_NORMAL_READS.C,
UNC_QMC_NORMAL_READS.C, UNC_QMC … PRIORITY_RE, UNC_QMC_CRITICAL_PRIORIT,
UNC_QMC_CRITICAL_PRIORIT, UNC_QMC … ULL.CH2, UNC_QMC_WRITES.FULL.ANY,
UNC_QMC_WRITES.PARTIAL.C, UNC_QMC … QMC_CANCEL.ANY, UNC_QMC_PRIORITY_UPDATE,
UNC_QMC_PRIORITY_UPDATE, UNC_QM … LED_SINGL, UNC_QPI_TX_STALLED_SINGL,
UNC_QPI_TX_STALLED_SINGL, UNC_QPI_TX_STALLED_SINGL, UNC_QPI_TX_STALLED_SINGL, UNC_QPI_TX_STALLED_SINGL, UNC_QPI_TX_STALLED_SINGL, UNC_QPI_TX_STALLED_MULTI, UNC_QPI_TX_STALLED_MULTI,
UNC_QPI_TX_STALLED_MULTI, UNC_QPI_TX_STALLED_MULTI, UNC_QPI_TX_STALLED_MULTI, UNC_QPI_TX_STALLED_MULTI, UNC_QPI_TX_STALLED_MULTI, UNC_QPI_TX_HEADER.BUSY.LI, UNC_QPI_TX_HEADER.BUSY.LI,
UNC_QPI_RX_NO_PPT_CREDI, UNC_QPI_RX_NO_PPT_CREDI, UNC_DRAM_OPEN.CH0, UNC_DRAM_OPEN.CH1, UNC_DRAM_OPEN.CH2, UNC_DRAM_PAGE_CLOSE.CH0, UNC_DRAM_PAGE_CLOSE.CH1, UNC_DRAM_PAGE_CLOSE.CH2,
UNC_DRAM_PAGE_MISS.CH0, UNC_DRAM_PAGE_MISS.CH1, UNC_DRAM_PAGE_MISS.CH2, UNC_DRAM_READ_CAS.CH0, UNC_DRAM_READ_CAS.AUTO, UNC_DRAM_READ_CAS.CH1, UNC_DRAM_READ_CAS.AUTO,
UNC_DRAM_READ_CAS.CH2, UNC_DRAM_READ_CAS.AUTO, UNC_DRAM_WRITE_CAS.CH0, UNC_DRAM_WRITE_CAS.AUTO, UNC_DRAM_WRITE_CAS.CH1, UNC_DRAM_WRITE_CAS.AUTO, UNC_DRAM_WRITE_CAS.CH2,
UNC_DRAM_WRITE_CAS.AUTO, UNC_DRAM_REFRESH.CH0

Hardware Performance Counters

Derived Metrics

- **Clock cycles per Instructions (CPI)**
  - → CPI indicates if the application is utilizing the CPU or not
  - → Take care: Doing “something” does not always mean doing “something useful”.
- **Floating Point Operations per second (FLOPS)**
  - → How many arithmetic operations are done per second?
  - → Floating Point operations are normally really computing and for some algorithms the number of floating point operations needed can be determined.

Amplifier XE – Hardware Counter

1 **CPI rate (Clock cycles per instruction)**:

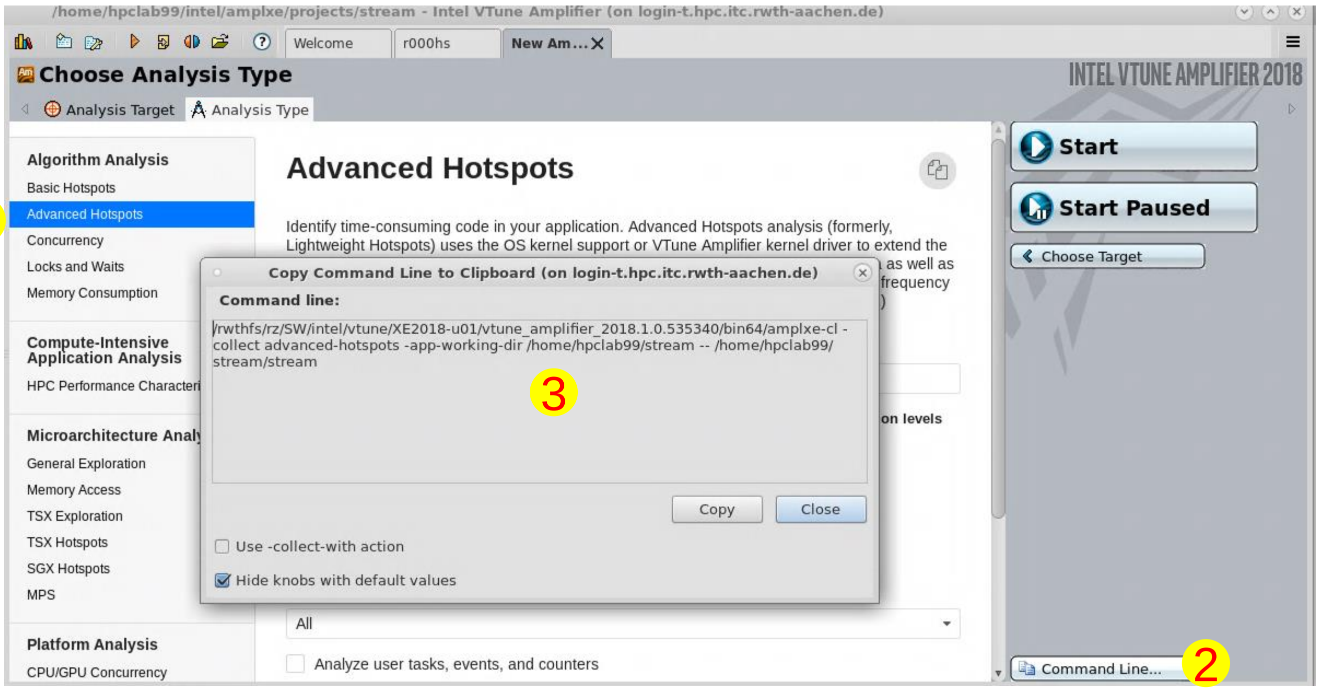
- In theory modern processors can finish 4 instructions in 1 cycle, so a CPI rate of 0.25 is possible
- A value between 0.25 and 1 is often considered as good for HPC applications
- Determine with “Advanced Hotspot” analysis

Elapsed Time: 3.327s

CPU Time:	**76.810s**
Instructions Retired:	45,579,600,000
1 CPI Rate:	4.213
CPU Frequency Ratio:	1.136
Total Thread Count:	24
Paused Time:	0s

VTune Amplifier in Command Line

- **Real-world analysis should be execute in the batch system**
  - 1 → Choose analysis type
  - 2 → Push “Command Line…”
  - 3 → Copy & paste command line and use in batch script



Summary

Correctness:

- Data Races are very hard to find, since they do not show up every program run.
- Intel Inspector XE or ThreadSanitizer help a lot in finding these errors.
- Use really small datasets, since the runtime increases significantly.
- If possible use non optimized code.

Performance:

- Start with simple performance measurements like hotspots analyses and then focus on these hot spots.
- In OpenMP applications analyze the waiting time of threads. Is the waiting time balanced?
- Hardware counters might help for a better understanding of an application, but they might be hard to interpret.