HPC Tools on Windows

- Excerpt -

Christian Terboven
terboven@rz.rwth-aachen.de

Center for Computing and Communication
RWTH Aachen University

WinHP³C

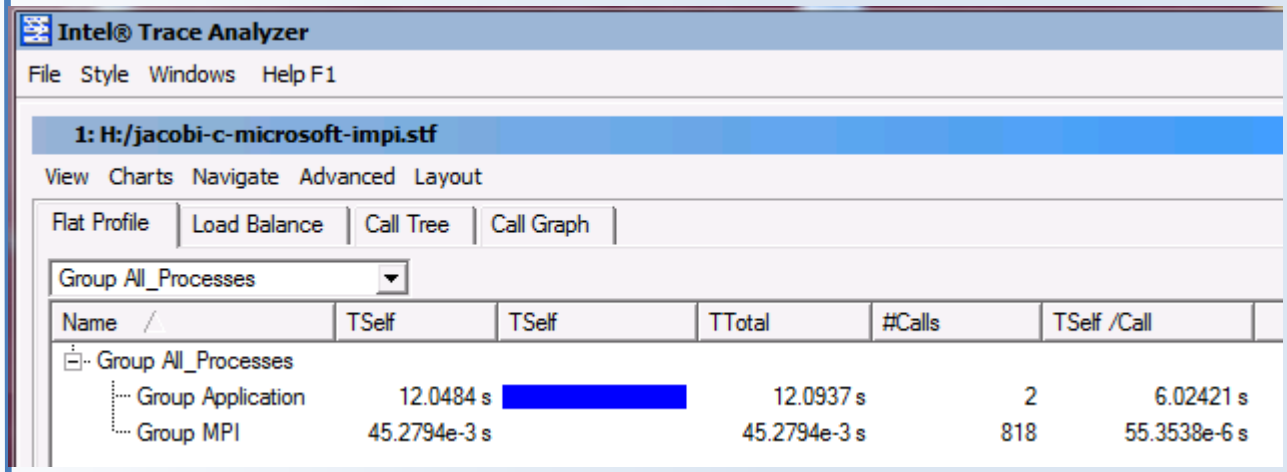
PPCES 2010
March 25th, RWTH Aachen University

Agenda

- Intel Trace Analyzer and Collector
- MS-MPI + Event Tracing for Windows
  - Epilog4Win
- Intel Parallel Studio
- Visual Studio 2010

Intel Trace Analyzer & Collector Overview

- MPI analysis tool available on Windows
  - Low-overhead, event-based Tracing
  - Profiling of message-passing (and multi-threading) applications



- Get overview …

- or detailed MPI profile information:

Name	TSelf	TSelf	TTotal
Group All_Processes			
Group Application	401.022e-3 s		401.604e-3 s
MPI_Isend	106.298e-6 s		106.298e-6 s
MPI_Irecv	47.6318e-6 s		47.6318e-6 s
MPI_Waitall	188.362e-6 s		188.362e-6 s
MPI_Allreduce	238.997e-6 s		238.997e-6 s

Intel Trace Analyzer and Collector: Usage

- Usage instructions:
  - Link with `VT.lib` from `C:\Program Files (x86)\Intel\ICT\3.1\ITAC\7.1\lib\#arch`, arch is either `impi32` (32bit) or `impi64` (64bit)
  - Initialize environment with `C:\Program Files (x86)\Intel\ICT\3.1\ITAC\7.1\bin\itacvars.bat`, then execute program as usually (specify working directory)
  - A trace file named `#executable.stf` is written along with other data files. Open in Intel Trace Analyzer from Start → All Programs → Intel Software Development Tools → Intel Trace Analyzer and Collector 7.1

Agenda

- Intel Trace Analyzer and Collector
- MS-MPI + Event Tracing for Windows
  - Epilog4Win
- Intel Parallel Studio
- Visual Studio 2010

MS-MPI Tracing Overview

- Bundled with MS-MPI comes a tracing provider for ETL
  - ETL: Event tracing for Windows
  - Just add a switch to mpiexec
- No native GUI for ETL-based MPI analysis (yet), but:
  - Converter to OTF format: Use Vampir GUI
    - Vampir GUI on Windows, available on `cluster-win-lab`
    - Vampir GUI on Linux can be used as well
  - Converter to CLOG format: Use JumpShot

MS-MPI Tracing: Usage

- Add option `-t` or `-trace` with optional argument *filter*:
  - all = (api + interconn), or any combination of
  - api: Trace MPI api events, can be limited to: pt2pt, poll, coll, rma, comm, errh, group, attr, dtype, io, topo, init, info, misc
  - interconn: Trace Interconnect-specific event, can be limited to: icsock, icshm, icnd
- Add option `-tracefile` *filename* to set trace name:
  - Default is *%USERPROFILE%\mpi_trace.etl*
  - _%CCP_JOBID%.%CCP_TASKID%.%CCP_TASKINSTANCEID% is appended to default name
- After tracing a clock synchronization is necessary:
  - `mpiexec -core 1 mpicsync trace.etl`

Epilog4Win: Usage

1. Download the code package from http://icl.cs.utk.edu/projectsfiles/kojak/software/kojak/win_epilog.zip - will be available in *Shared_Software* share shortly
2. Link your application with `epilog.lib` instead of `msmpi.lib`
3. When executing your application, put `epilog.dll` and `elg_merge.exe` in path

- Access the resulting `a.elg` trace file from Unix and use the tool of your choice:
  - First Expert and then Cube: Search for inefficiency patterns
    - `module load VIHPS scalasca`
    - `expert a.elg; cube a.cube;`
  - First elg2vtf and then Vampir: Visual Message Trace

Demo

trace Jacobi (Debug, cpp) on `cluster-win`

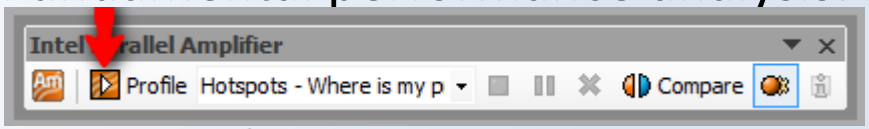
view trace in Vampir on `cluster-lab`

Agenda

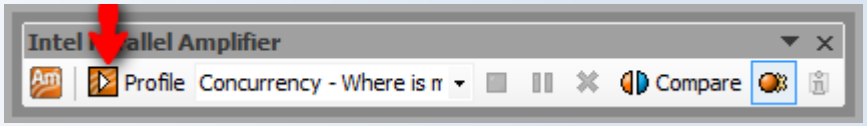
- Intel Trace Analyzer and Collector
- MS-MPI + Event Tracing for Windows
  - Epilog4Win
- Intel Parallel Studio
- Visual Studio 2010

Intel Parallel Studio: Overview

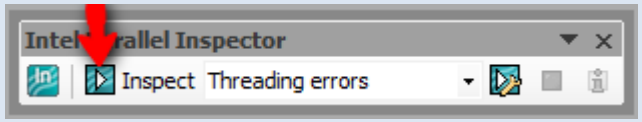
- Intel Parallel Studio provides (parts of) the functionality of the following products in a GUI from within Visual Studio:
  - Intel VTune: Performance Analysis
  - Intel Thread Profiler: Performance Analysis for Multi-Threading
  - Intel Thread Checker: Correctness Checking for Multi-Threading
- Intel Parallel Amplifier ( )
  - Fundamental performance analysis:



  - Analysis of multi-threaded programs:



- Intel Parallel Inspector ( )
  - Check for data races and the like:



  - Similar to Sun Studio Thread Analyzer

Intel Parallel Amplifier (1/2)

- Hotspot: Where is your program spending the time?

r000hs

Hotspots

Bottom-up | Top-down Tree

Function - Caller Function Tree	CPU Time:Self	Module
Jacobiomp1	7.063s	jacobi_SendRecv_cpp.exe
NtDelayExecution	2.628s	ntdll.dll
ExchangeJacobiMpiDataomp1	2.306s	jacobi_SendRecv_cpp.exe
Jacobi	1.038s	jacobi_SendRecv_cpp.exe
ExchangeJacobiMpiData	0.476s	jacobi_SendRecv_cpp.exe
InitializeMatrix	0.090s	jacobi_SendRecv_cpp.exe
new[]	0.060s	jacobi_SendRecv_cpp.exe
Init	0.034s	jacobi_SendRecv_cpp.exe
CheckError	0.031s	jacobi_SendRecv_cpp.exe
Finish	0.031s	jacobi_SendRecv_cpp.exe
CreateProcessInternalW	0.019s	kernel32.dll
NtContinue	0.010s	ntdll.dll
PrintResults	0.010s	jacobi_SendRecv_cpp.exe
NtAllocateVirtualMemory	0.008s	ntdll.dll
allshl	0.002s	ntdll.dll

```
60  #pragma omp para
61              for
62              {
63
64                  {
65                      fLRes = ( ax * (UOLD(j, i-1) + UOLD(j, i+1))
66                                + ay * (UOLD(j-1, i) + UOLD(j+1, i))
67                                +  b * UOLD(j, i) - F(j, i)) / b;          5.980s
68
69                      /* update solution */
70                      U(j,i) = UOLD(j,i) - data.fRelax * fLRes;           0.877s
71
72                      /* accumulate residual error */
73                      residual += fLRes * fLRes;                          0.067s
74                  }                                                       0.068s
75              }                                                           0.026s
```

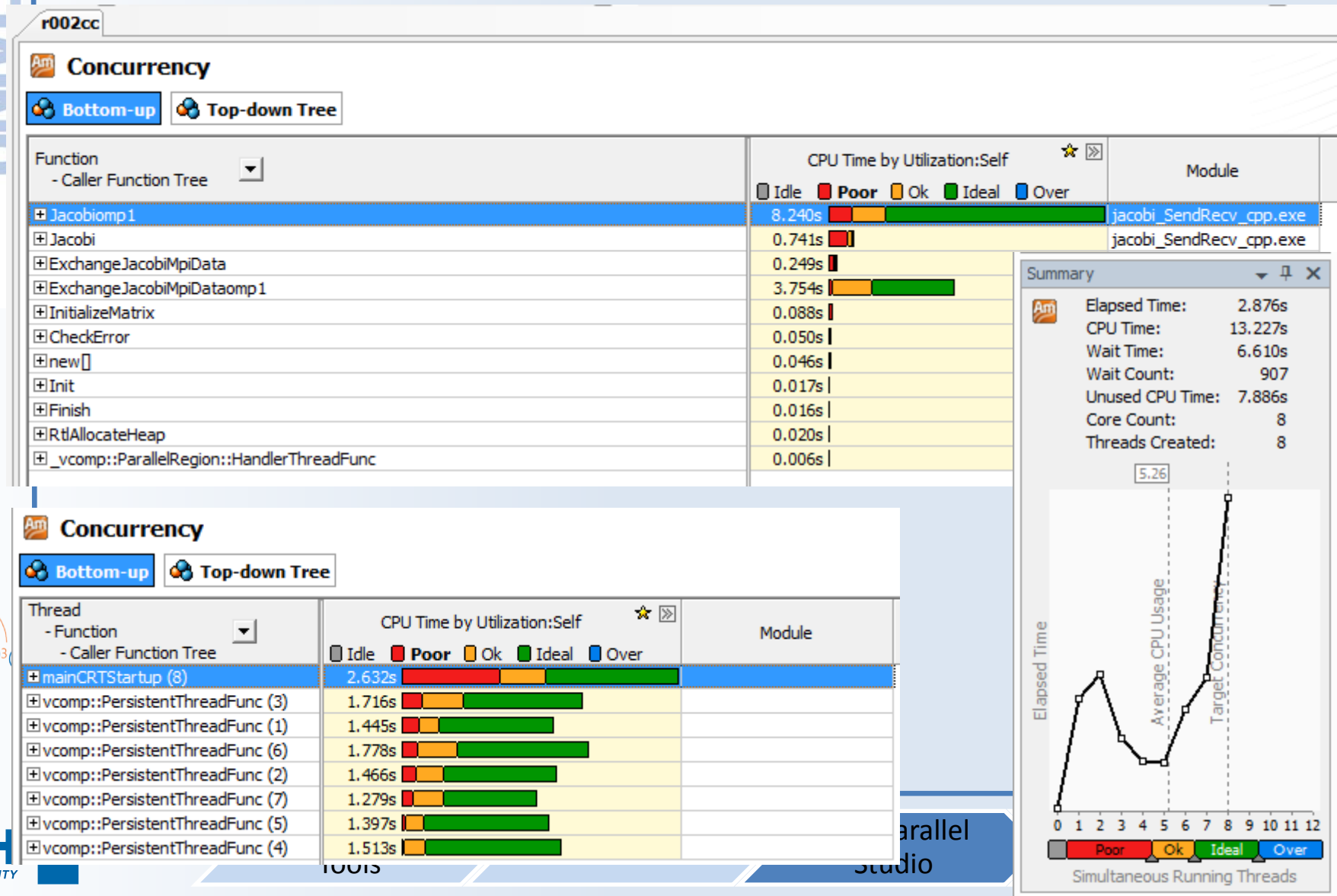
12

Drill down to the source!

Visual Studio 2010

Intel Parallel Amplifier (2/2)

- Concurrency: What are you doing in parallel?

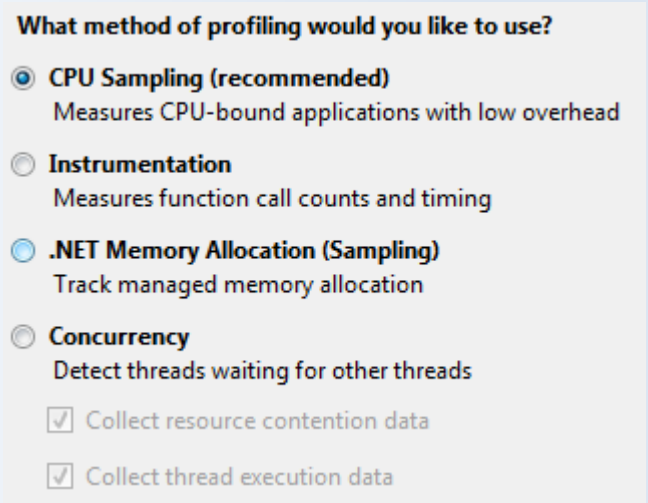


Agenda

- Intel Trace Analyzer and Collector
- MS-MPI + Event Tracing for Windows
  - Epilog4Win
- Intel Parallel Studio
- Visual Studio 2010

Visual Studio 2010: Performance Analyzer

- VS2010 comes with new tools to analyze your (parallel) application's performance: *Analyze* → Profiler → *New Performance Session*, then *Analyze* → *Launch Performance Wizard*



All this can be done on the local Workstation, or in the Cluster!

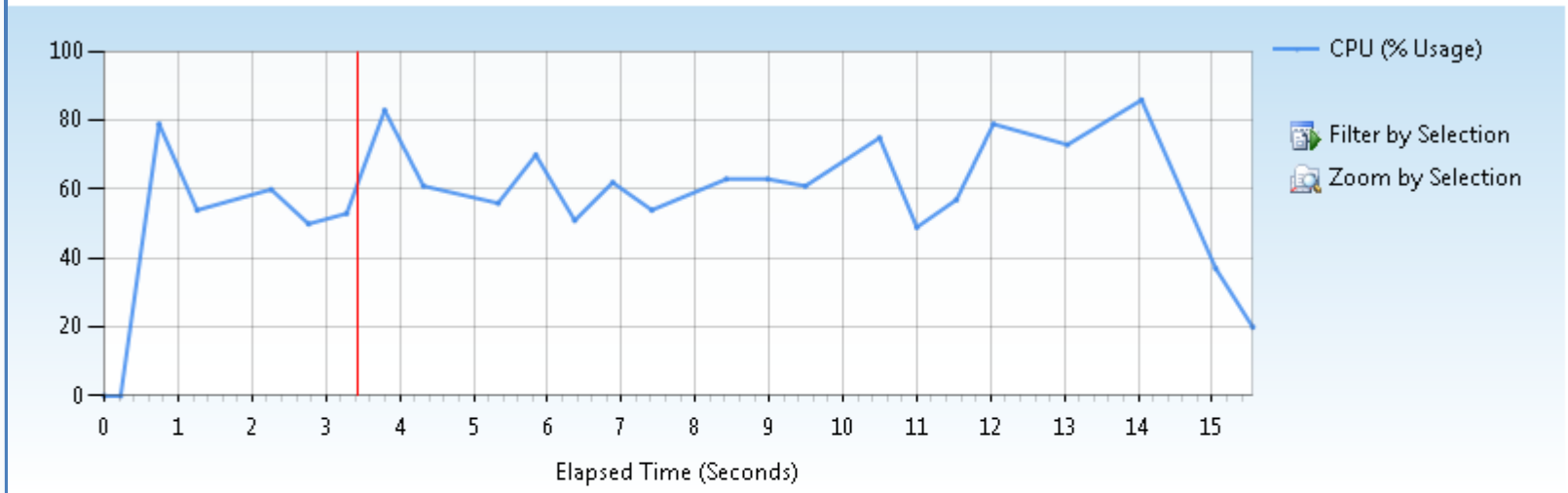
- CPU Sampling:
  - Will run the application under the control of a sampling performance analyzer (snapshots of the program's call tree are taken at regular intervals → non-intrusive, low overhead)

Performance Analyzer: CPU Sampling (1/2)

- Summary View highlights the program's Critical Path:

Sample Profiling Report

2,702 total samples collected



Hot Path

The most expensive call path based on sample counts

Name	Inclusive %	Exclusive %
jacobi_aut.exe	100,00	0,00
_mainCRTStartup	100,00	0,00
__tmainCRTStartup	100,00	0,00
_main	100,00	0,00
_Jacobi	**98,70**	**98,48**

Related Views: Call Tree Functions

Performance Analyzer: CPU Sampling (2/2)

- Performance information can be display on source level:

```
                /* compute stencil, residual and update */
                for (j = data->iRowFirst + 1; j <= data->iRowLast - 1; j++)
                {
  1.9 %             for (i = 1; i <= data->iCols - 2; i++)
                    {
                        fLRes = ( ax * (UOLD(j, i-1) + UOLD(j, i+1))
                                + ay * (UOLD(j-1, i) + UOLD(j+1, i))
 66.5 %                 +  b * UOLD(j, i) - F(j, i)) / b;

                        /* update solution */
 11.4 %                 U(j,i) = UOLD(j,i) - data->fRelax * fLRes;
```

Cost Distribution

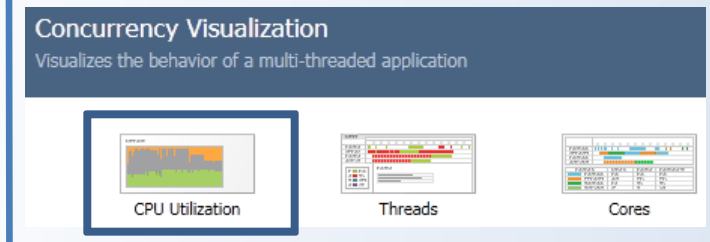
The cost distribution for the function and functions it calls is shown below.

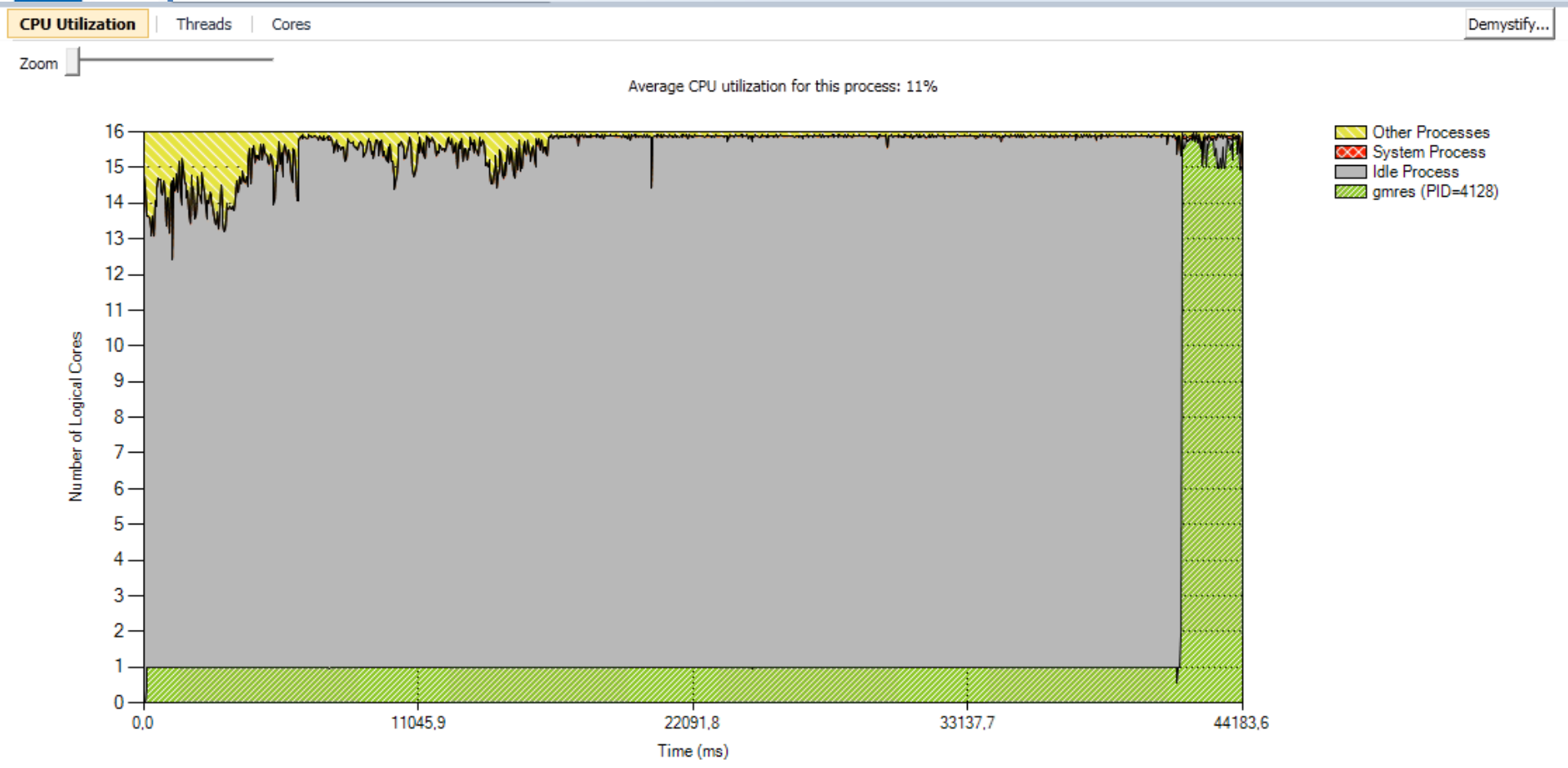
Performance metric: Inclusive Samples %

Calling Functions	
__tmainCRTStartup	100.0%

_main	Total: 100.0%
Function Body	< 0.1%
_Jacobi	98.7%
_InitializeMatrix	0.6%
_CheckError	0.4%
_Finish	0.3%
_Init	0.1%

Concurrency (1/3)





Concurrency (2/3)

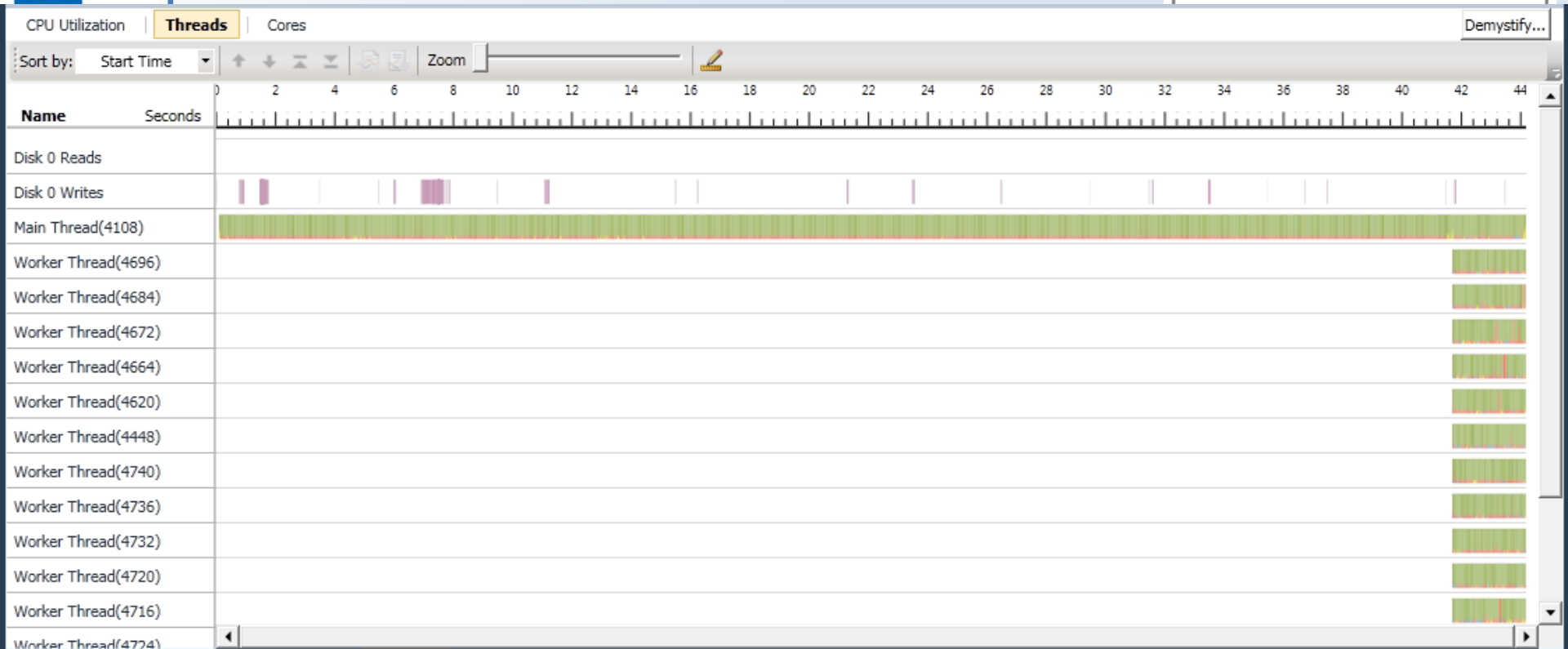
Concurrency Visualization
Visualizes the behavior of a multi-threaded application

Visible Timeline Profile

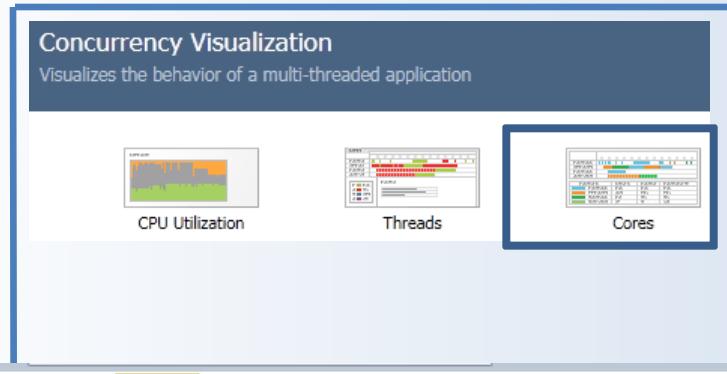
98%	Execution
1%	Synchronization
0%	I/O
0%	Sleep
0%	Memory Management
0%	Preemption
0%	UI Processing

Per Thread Summary

File Operations



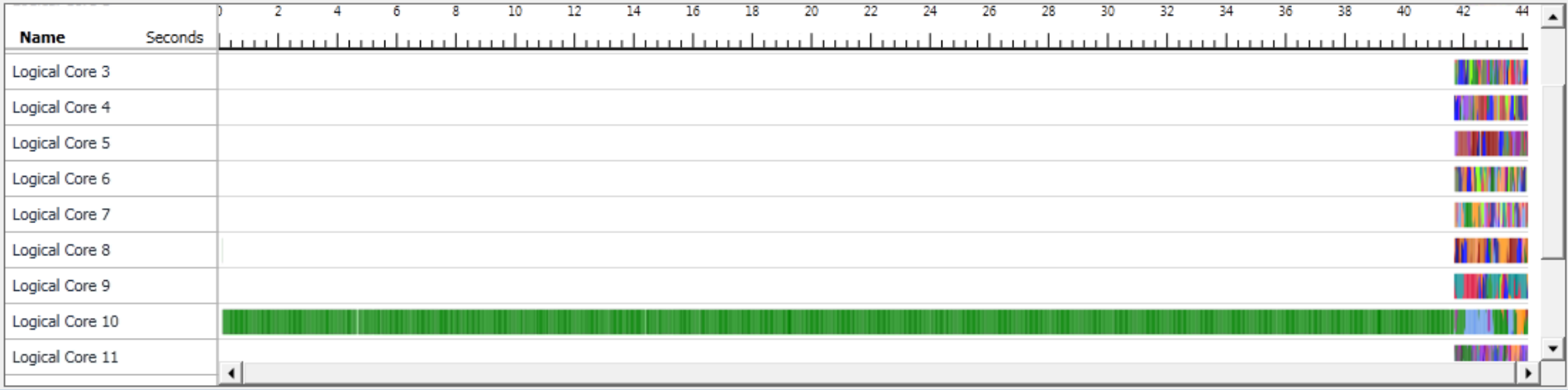
Concurrency (3/3)



CPU Utilization | Threads | **Cores** Demystify...

Context switches that also cross from one logical core to another can reduce the performance of your process.

Zoom



Thread Name	Cross-Core Context Switches	Total Context Switches	Percent of Context Switches that Cross Cores
Worker Thread(4768)	272	808	33,66%
Worker Thread(4708)	256	998	25,65%
Worker Thread(4712)	244	979	24,92%
Worker Thread(4620)	227	667	34,03%
Worker Thread(4736)	220	938	23,45%
Worker Thread(4672)	219	1.328	16,49%

Performance Analyzer: Thread Contention (1/2)

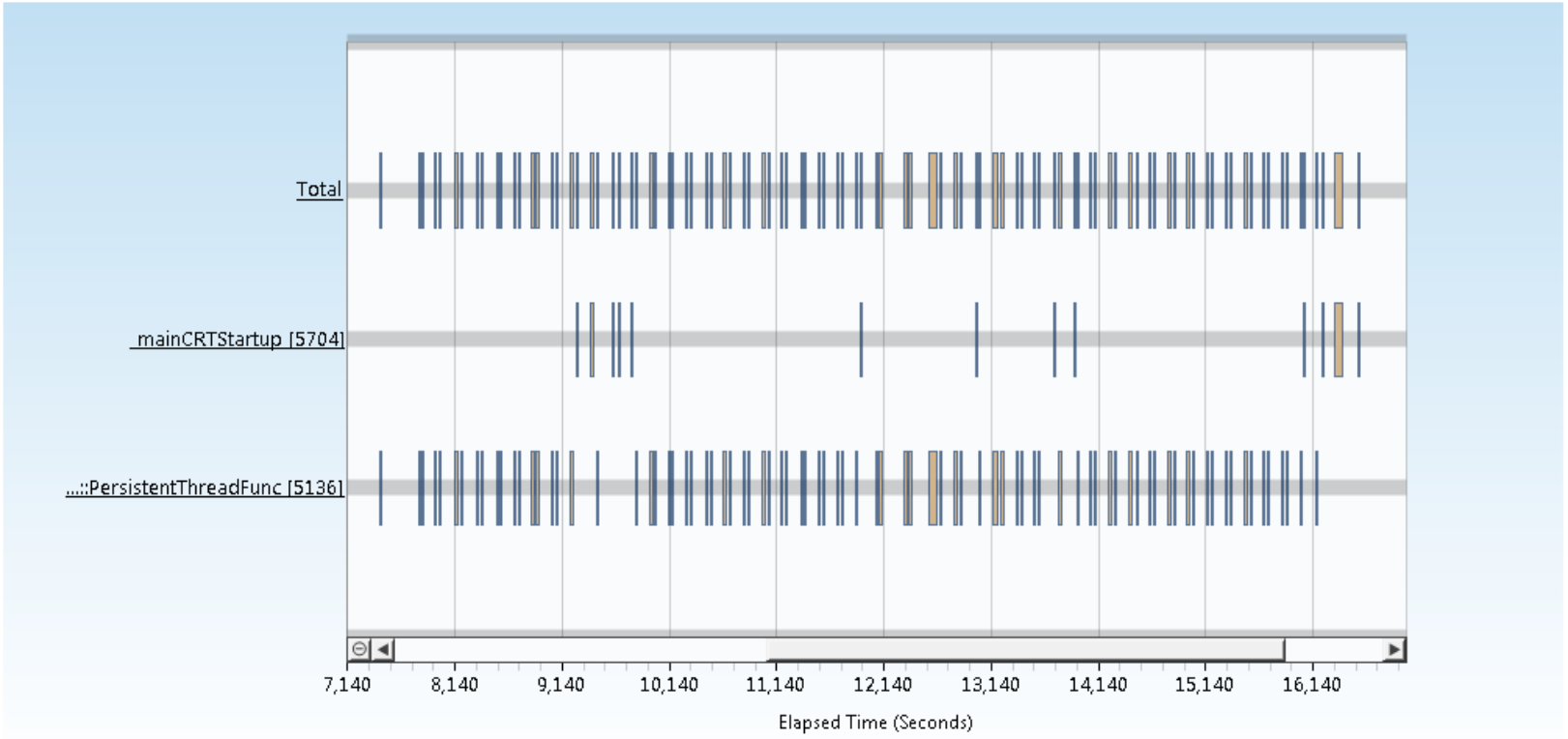
- If threads compete for resources, they can get stalled:

Most Contended Resources

Resources with the highest number of total contentions

Name	Contentions %	Contentions
Handle 1	100,00	97

Contentions of "Handle 1"



Performance Analyzer: Thread Contention (2/2)

- Reason for this contention: OpenMP Barrier

Current View: Call Tree

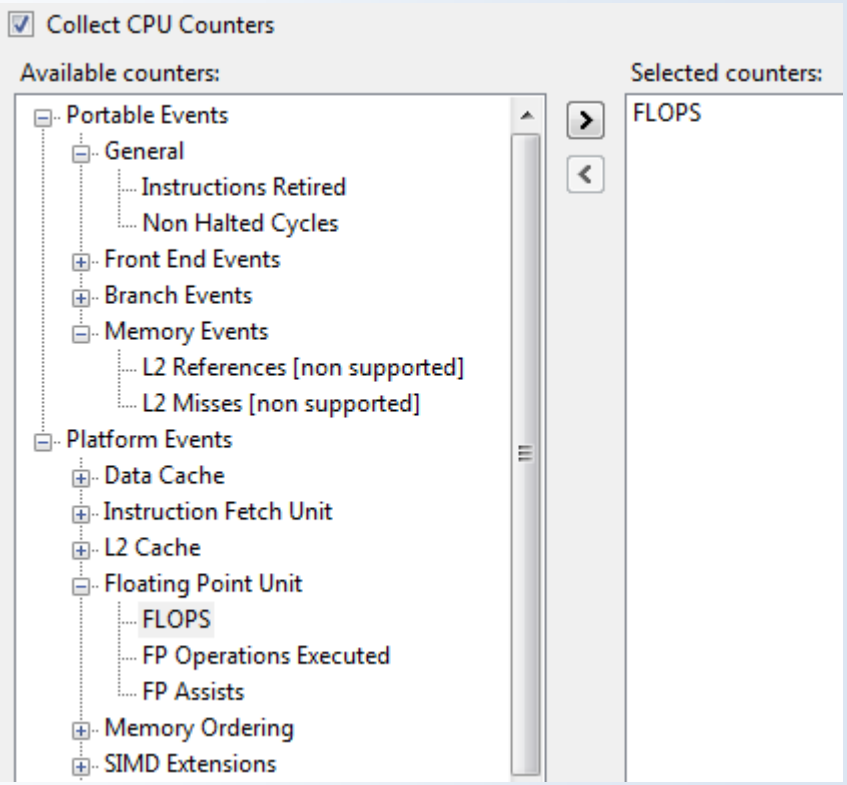
Function Name	Module Name	Inclusive Contentions	Exclusive Contentions	Inclusive Blocked Time	Exclusive Blocked Time
jacobi_aut.exe		97	0	1.518	0
Jacobiomp1	jacobi_aut.exe	84	0	1.346	0
__vcomp_barrier	VCOMP90D.DLL	84	84	1.346	1.346
_mainCRTStartup	jacobi_aut.exe	13	0	172	0
Unknown Frame(s)	UNKNOWN	13	0	172	0
Jacobiomp1	jacobi_aut.exe	13	0	172	0
__vcomp_barrier	VCOMP90D.DLL	13	13	172	172

  - Support for OpenMP constructs is not yet optimal
  - VCOMP = Visual C/C++ OpenMP Runtime Library

- This analysis is crucial if you do your own synchronization!
  - Don‘t do that, please…

Performance Analyzer: Even more features...

- Performance tuning can be a never ending story, so you need metrics to decide where to work / when to stop: Hardware Counter Information.



L2 information can be used to measure the memory bandwidth consumed by the application → is your scalability limited by the system architecture?

The FLOPS rate is good to estimate how efficient the code runs!

...

The End

Thank you for your attention!

Questions?