Visual Studio for HPC Application Development

Christian Terboven
terboven@rz.rwth-aachen.de

Center for Computing and Communication
RWTH Aachen University

WinHP³C

PPCES 2010
March 22nd, RWTH Aachen University

Agenda

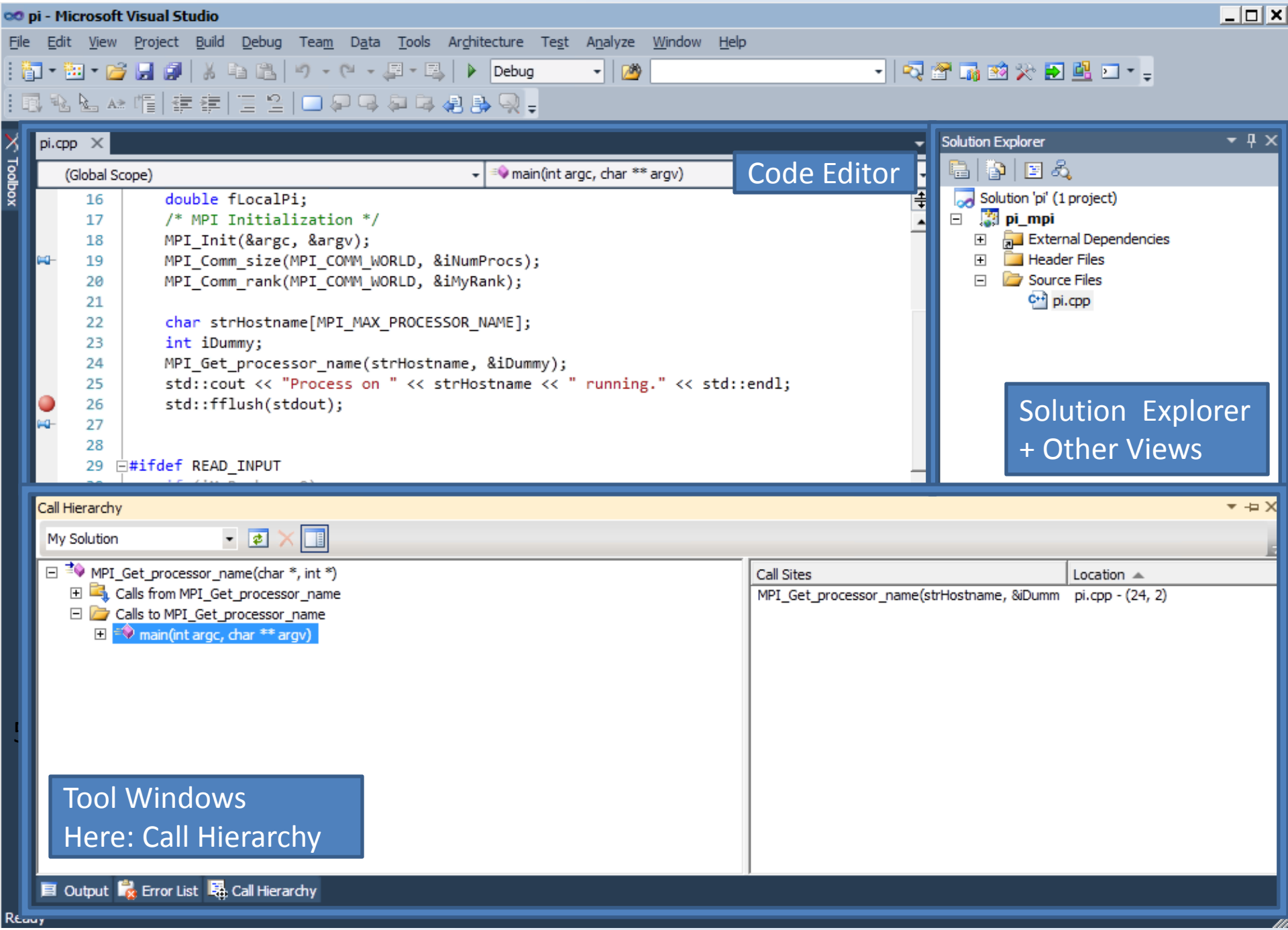
- Overview and Project Management
- The Microsoft and Intel compilers
- Using MPI and OpenMP
- Debugging OpenMP programs
- Debugging MPI programs

Visual Studio Versions overview

- As of today Visual Studio 2008 is the official product version and installed on `cluster-win.rz.rwth-aachen.de`.
  - Runtime available on all compute node
  - Intel C/C++ and Fortran compilers integration available
  - Allinea DDTlight plug available
- Visual Studio 2010 is expected to be released on April 12th and currently installed as RC on `cluster-win-lab.rz....`
  - Runtime available only on `RZ_HPCLAB` compute nodes
  - No integration of Intel C/C++ and Fortran compilers
  - Comes with many enhancements for parallel programming
  - Will be installed on `cluster-win` soon after release
- This course will cover VS2010 already (where possible)

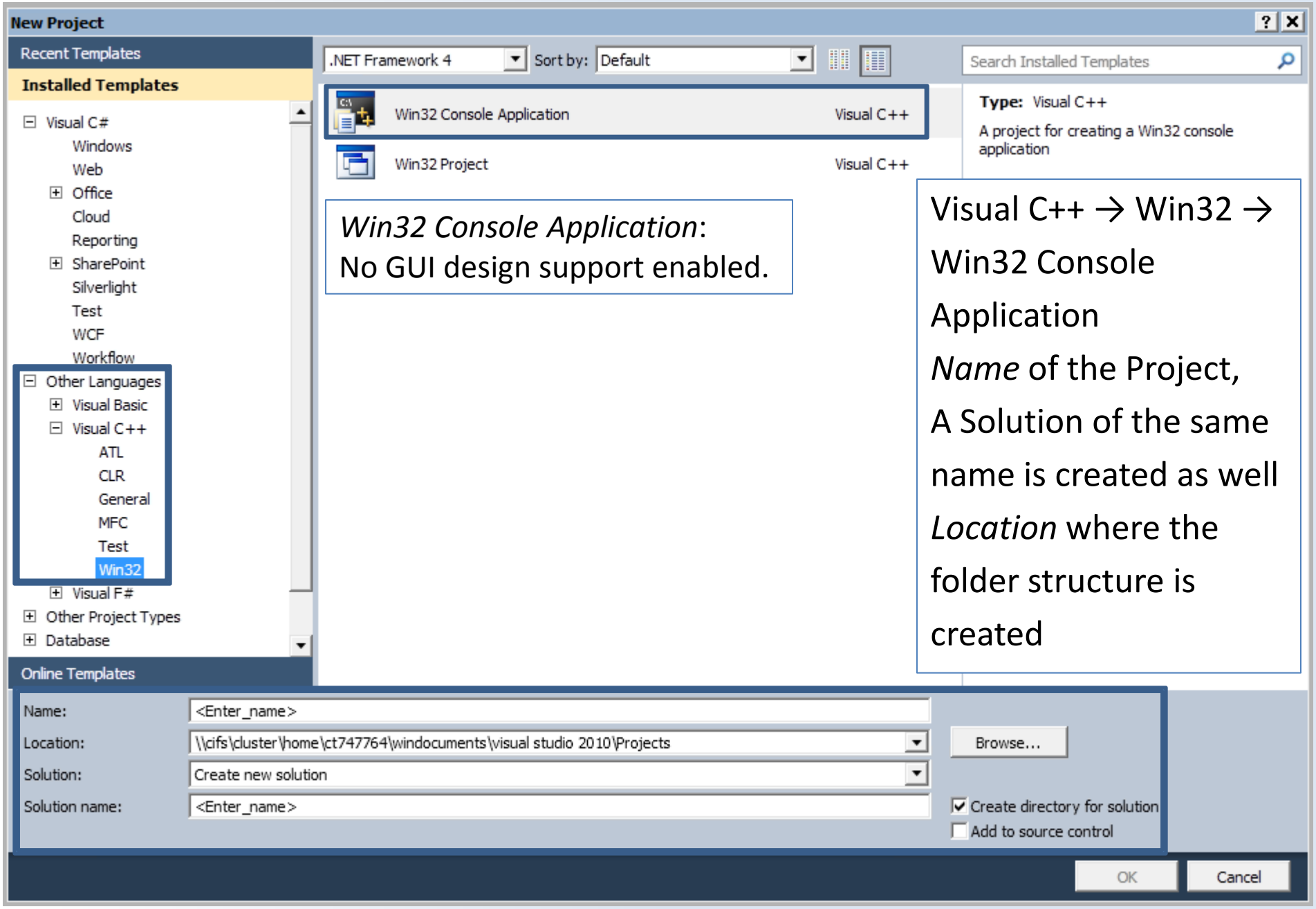
VS2010 Overview (1/2)

- Introduction into using Visual Studio 2008
  - Only text-mode programs are considered here, as HPC applications typically do not use GUIs
  - VS2010 offers great support for GUI development on Windows
- VS2010 provides good support for Parallel Programming:
  - Support for OpenMP for Shared-Memory parallel compilation
  - Debugging of parallel programs: OpenMP and MPI
  - Architecture-specific compiler optimizations
- The first start of VS2010 may take a few minutes as the help system has to be updated. You are prompted to choose from several pre-defined sets of VS2010 configuration options: I nowadays choose the C# settings (previously: C++).


pi - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
pi.cpp
(Global Scope)
main(int argc, char ** argv)
Code Editor
16 double fLocalPi;
17 /* MPI Initialization */
18 MPI_Init(&argc, &argv);
19 MPI_Comm_size(MPI_COMM_WORLD, &iNumProcs);
20 MPI_Comm_rank(MPI_COMM_WORLD, &iMyRank);
21
22 char strHostname[MPI_MAX_PROCESSOR_NAME];
23 int iDummy;
24 MPI_Get_processor_name(strHostname, &iDummy);
25 std::cout << "Process on " << strHostname << " running." << std::endl;
26 std::fflush(stdout);
27
28
29 #ifdef READ_INPUT
Solution Explorer
Solution 'pi' (1 project)
pi_mpi
External Dependencies
Header Files
Source Files
pi.cpp
Solution Explorer + Other Views
Call Hierarchy
My Solution
MPI_Get_processor_name(char *, int *)
Calls from MPI_Get_processor_name
Calls to MPI_Get_processor_name
main(int argc, char ** argv)
Call Sites
Location
MPI_Get_processor_name(strHostname, &iDumm
pi.cpp - (24, 2)
Tool Windows Here: Call Hierarchy
Output
Error List
Call Hierarchy
Ready

Visual Studio: Project Management (1/5)

- Everything that you do in Visual Studio will take place within the context of a *Solution*.
  - A Solution is a higher-level container for other items, for example a *Project*. Any other kind of file type can also be added to a Solution, for example documentation items.
  - A Solution can not contain another Solution.
  - Solutions group and apply properties across projects.
- A *Project* maps one to one with a compiler target.
  - A Project organizes the code.
- To start your work, a new Project has to be created with *File → New → Project…*



Win32 Console Application:
No GUI design support enabled.

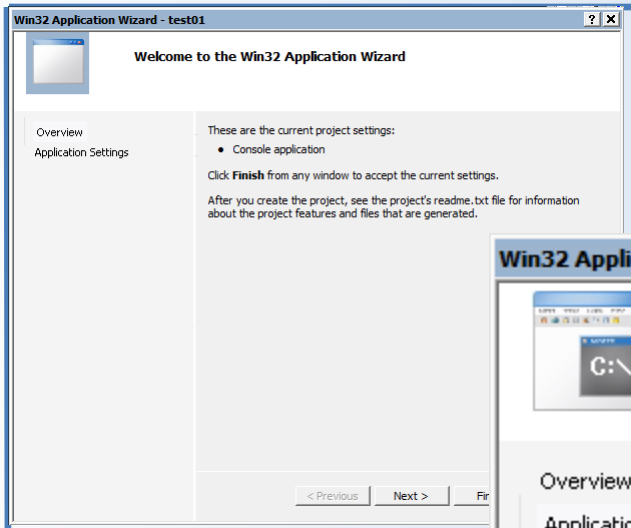
Visual C++ → Win32 → Win32 Console Application

Name of the Project,

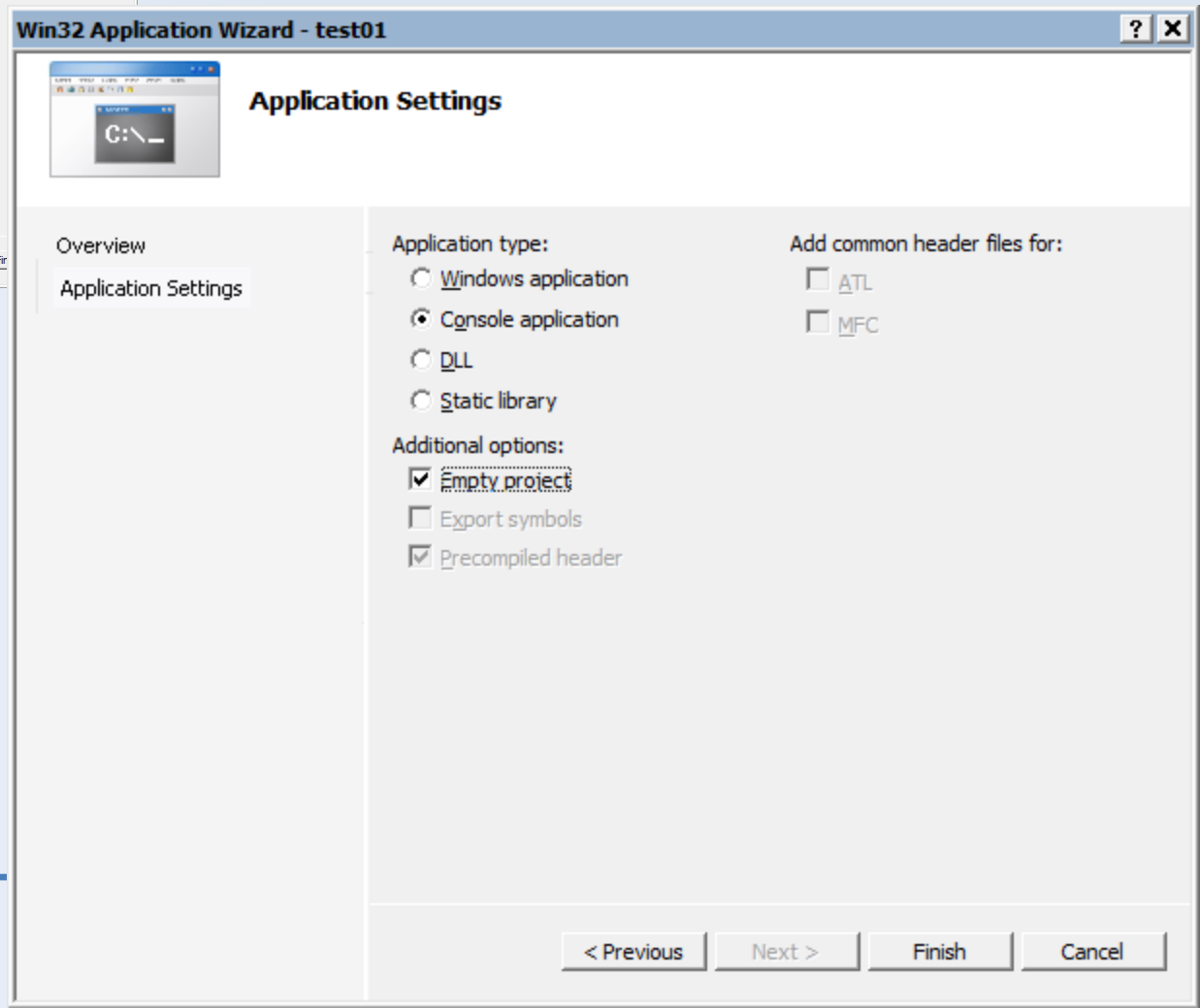
A Solution of the same name is created as well

Location where the folder structure is created

Visual Studio: Project Management (3/5)

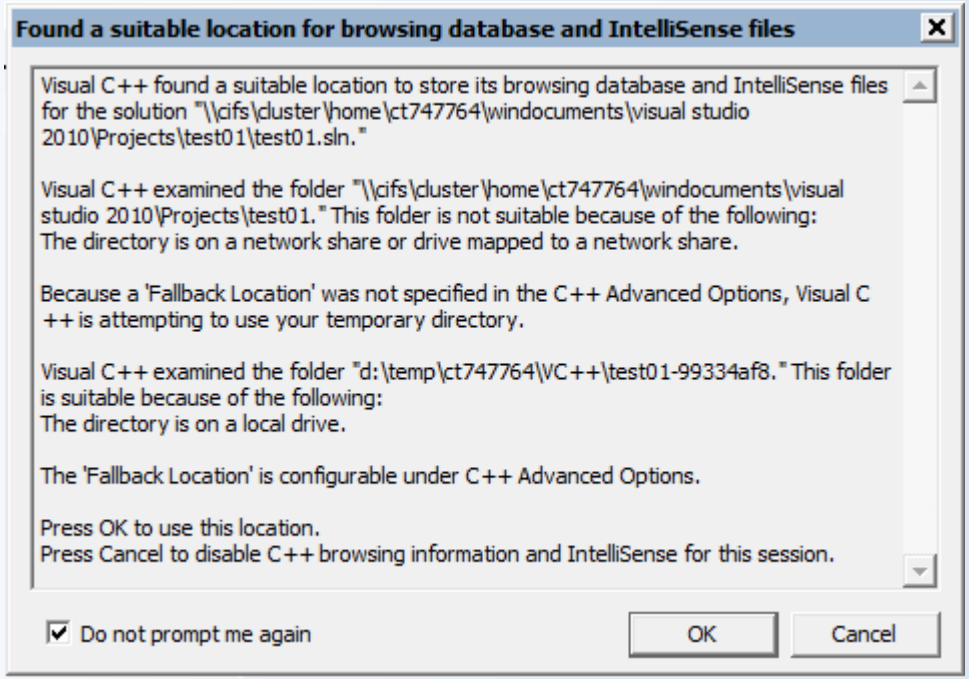


Choose *Empty project* if you already have source files.

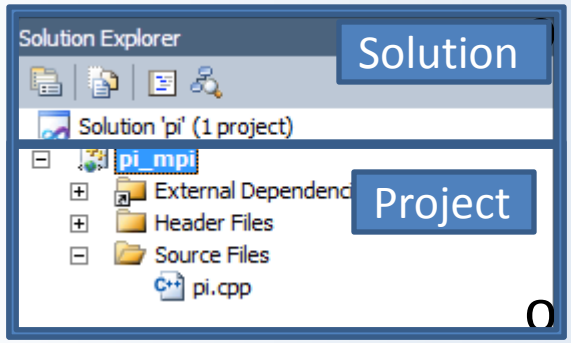


Visual Studio: Project Management (4/5)

- An issue specific to our Cluster: The IntelliSense database may not be stored on a network drive. VS2010 resolves this automatically for you by selecting *Ok*.



Visual Studio: Project Management (5/5)

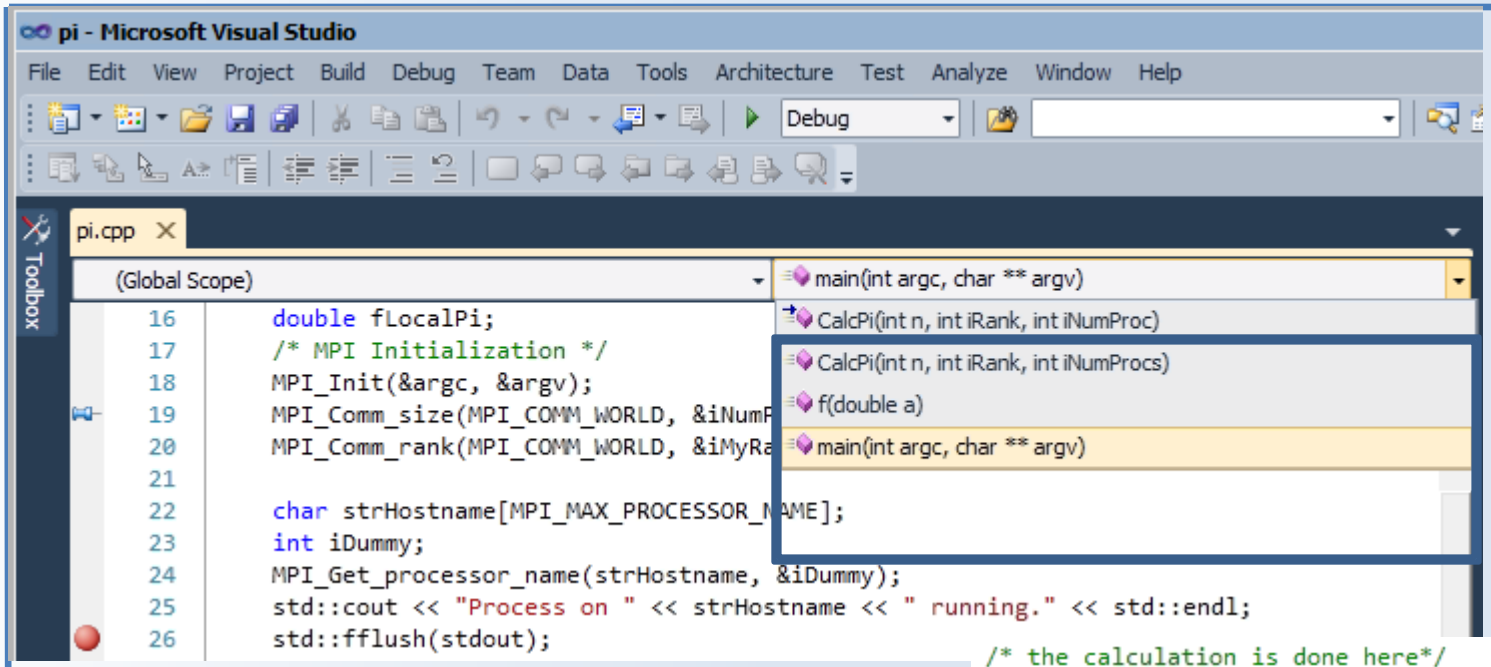


In many cases, the shortest way to a desired operation can be found by right-clicking on a GUI element and using the context menu.

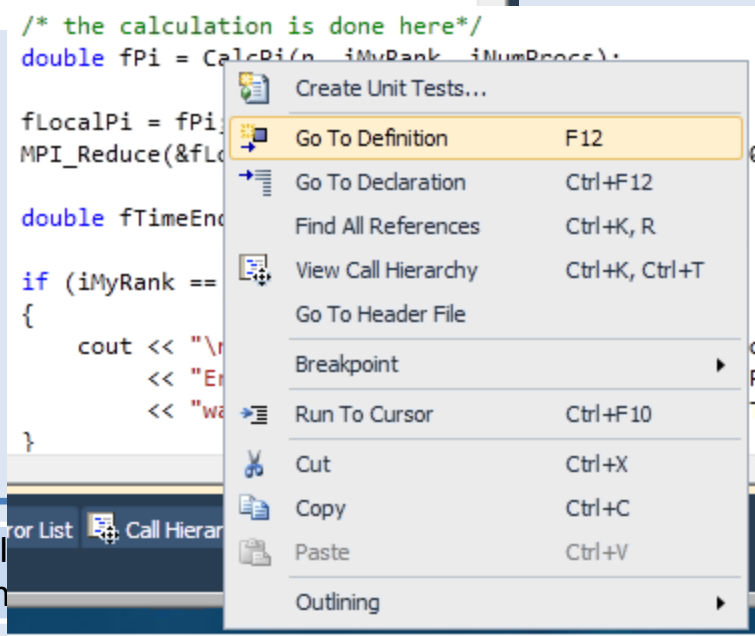
- Adding existing source code items (files) to a project: right-click on the Project (not the Solution !) and *Add → Existing Item…*
- Adding new items: right-click on the Project and *Add → New Item…*

- The folders (e.g. *Source Files*) do not have any other meaning than aiding you in structuring the files in a project. They do not map to physical folders. Creating your own folders may help to organize large projects.

Source navigation in Visual Studio 2010 (1/2)



- Selecting a scope + function to navigate right into it‘s imple-mentation.
- Right-clicking a symbol opens up a corresponding context menu:



Source navigation in Visual Studio 2010 (2/2)

- The Class View is available from the menu via *View → Class View* as well as the *Code Definition Window*.

```
22      char strHostname[MPI_MAX_PROCESSOR_NAME];
23      int iDummy;
24      MPI_Get_processor
25      std::cout << "Process on " << strHostname << " running
26      std::fflush(stdout);
27
```

#define MPI_MAX_PROCESSOR_NAME 128

Code Definition Window - #define MPI_DOUBLE ((MPI_Datatype)0x4c00080b) (mpi.h)

```
30  #define MPI_WCHAR           ((MPI_Datatype)0x4c00020e)
31  #define MPI_SHORT           ((MPI_Datatype)0x4c000203)
32  #define MPI_UNSIGNED_SHORT  ((MPI_Datatype)0x4c000204)
33  #define MPI_INT             ((MPI_Datatype)0x4c000405)
34  #define MPI_UNSIGNED        ((MPI_Datatype)0x4c000406)
35  #define MPI_LONG            ((MPI_Datatype)0x4c000407)
36  #define MPI_UNSIGNED_LONG   ((MPI_Datatype)0x4c000408)
37  #define MPI_FLOAT           ((MPI_Datatype)0x4c00040a)
38  #define MPI_DOUBLE          ((MPI_Datatype)0x4c00080b)
39  #define MPI_LONG_DOUBLE     ((MPI_Datatype)0x4c00080c)
40  #define MPI_LONG_LONG_INT   ((MPI_Datatype)0x4c000809)
41  #define MPI_UNSIGNED_LONG_LONG ((MPI_Datatype)0x4c000819)
42  #define MPI_LONG_LONG       MPI_LONG_LONG_INT
43
44  #define MPI_PACKED          ((MPI_Datatype)0x4c00010f)
45  #define MPI_LB              ((MPI_Datatype)0x4c000010)
```

Output | Error List | Code Definition Window | Call Hierarchy

Class View

<Search>

pi_mpi
- Global Functions and Variables
- Macros and Constants

- CalcPi(int n, int iRank, int iNumProc)
- CalcPi(int n, int iRank, int iNumProcs)
- f(double a)
- GetRealTime(void)
- main(int argc, char ** argv)

Solution Ex... | Team Explo... | Class View

Directory layout of Visual Studio solutions

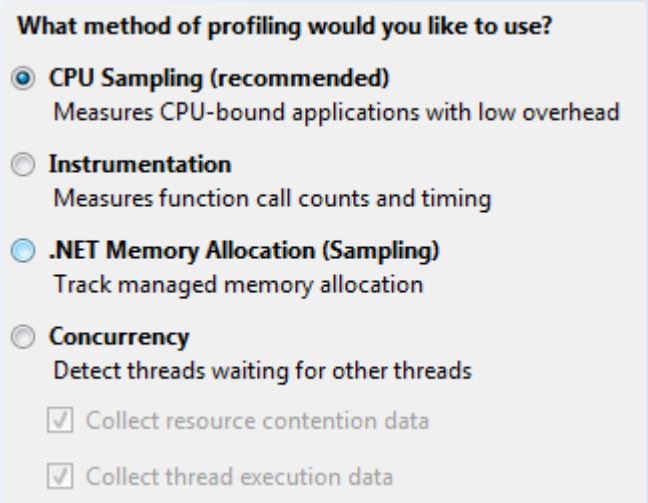
- The executable is created in the directory of the active configuration during the build process.

- Directory structure of a solution:

<top level>	Given user directory
<project name>	Created by VS2005 / VS2008
Debug	Configuration: *Debug*
Release	Configuration: *Release*
x64	Platform: *x64* (64bit for Amd64/Intel64)
Debug	Configuration: *Debug*
Release	Configuration: *Release*

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!

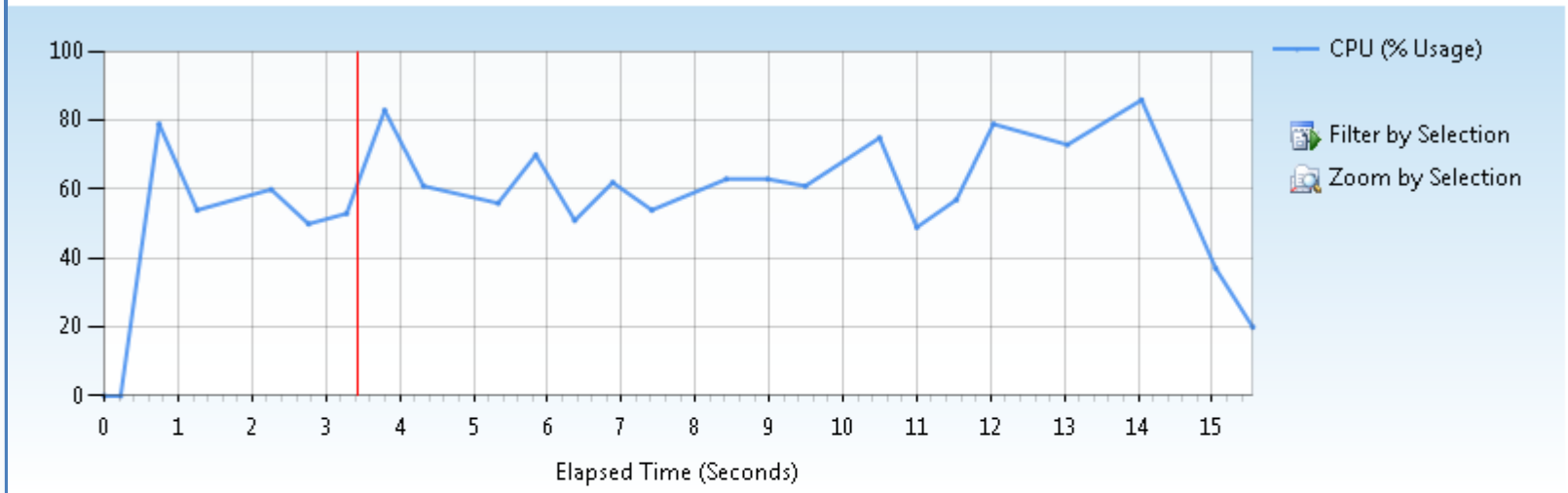
- CPU Sampling:
  - Will run the application under the control of a sampling performance analyzer (snapshots of the program's call tree are taken at regular intervals → non-intrusive, low overhead)

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 %

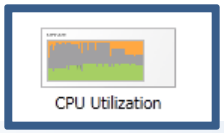
1.0 %
0.7 %
< 0.1 %

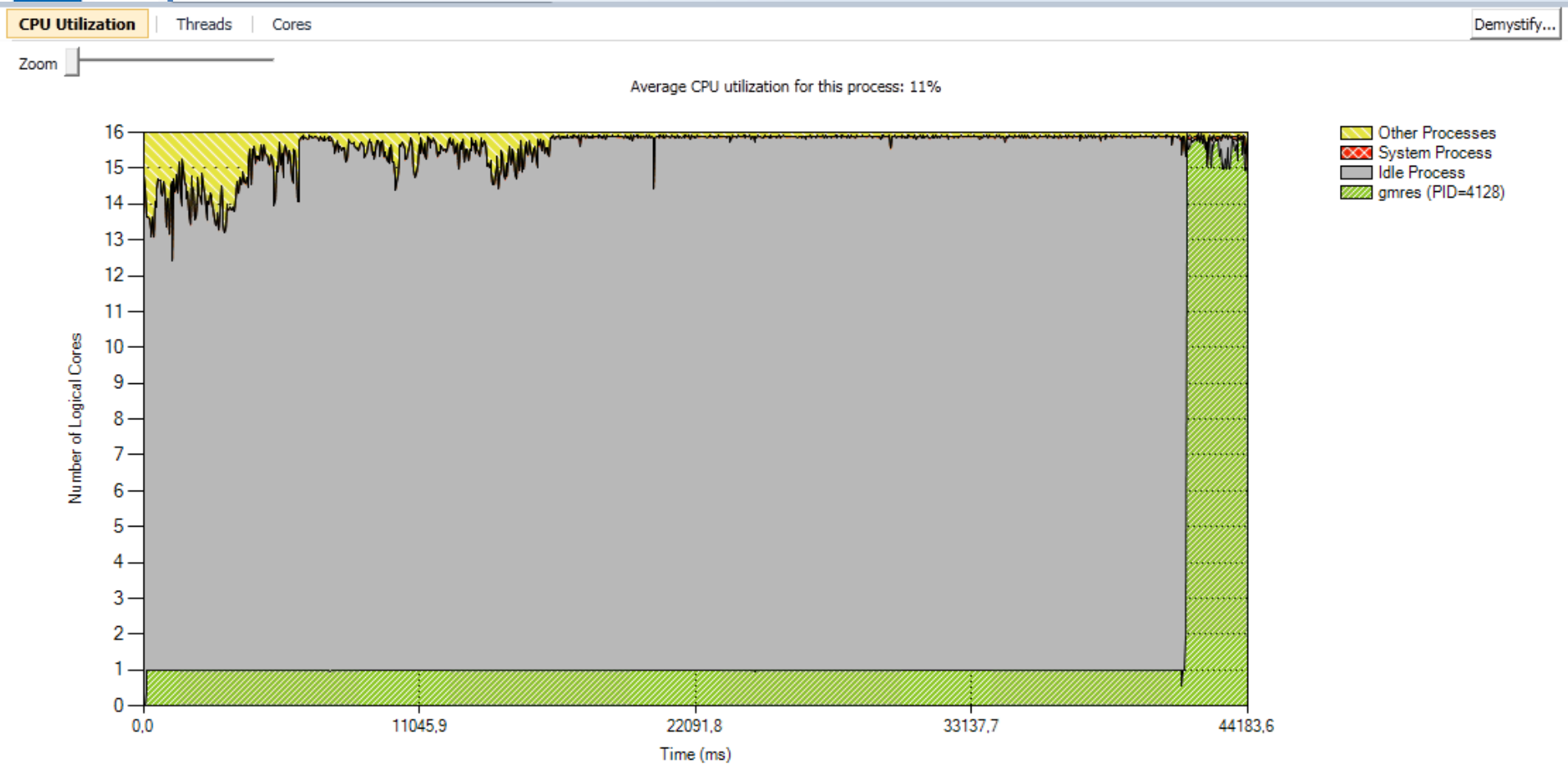
Calling Functions	
__tmainCRTStartup	100.0%

_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 Visualization
Visualizes the behavior of a multi-threaded application





Concurrency (2/3)

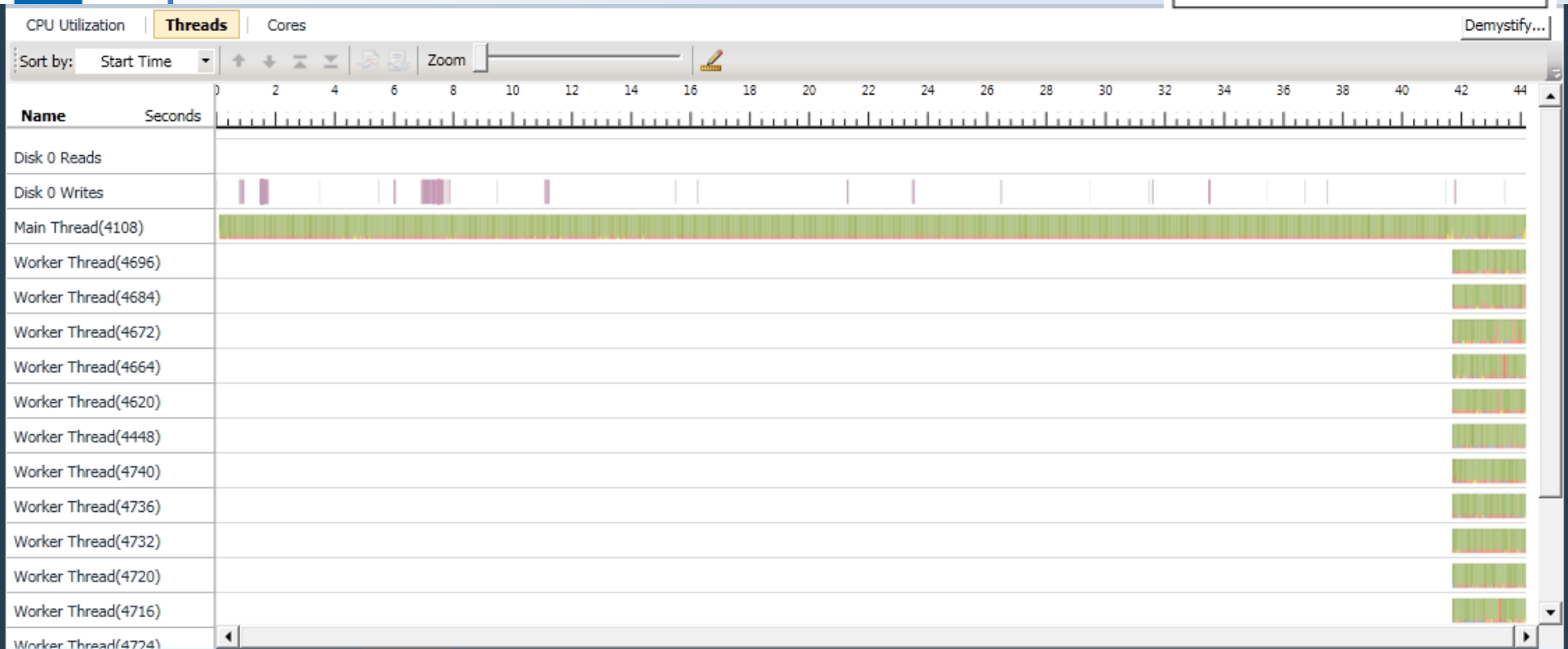
Concurrency Visualization
Visualizes the behavior of a multi-threaded application

Visible Timeline Profile

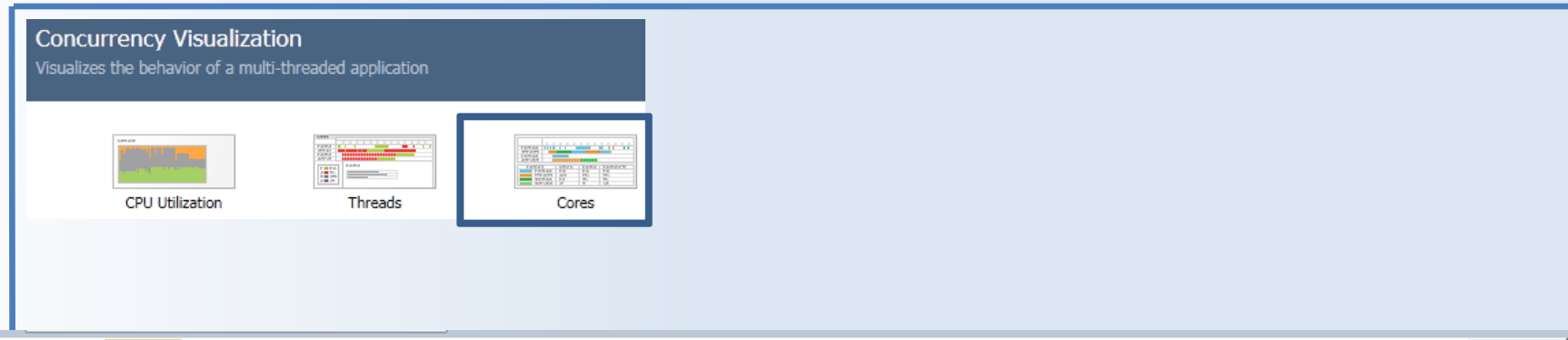
98%	Execution
1%	Synchronization
0%	I/O
0%	Sleep
0%	Memory Management
0%	Preemption
0%	UI Processing

Per Thread Summary

File Operations



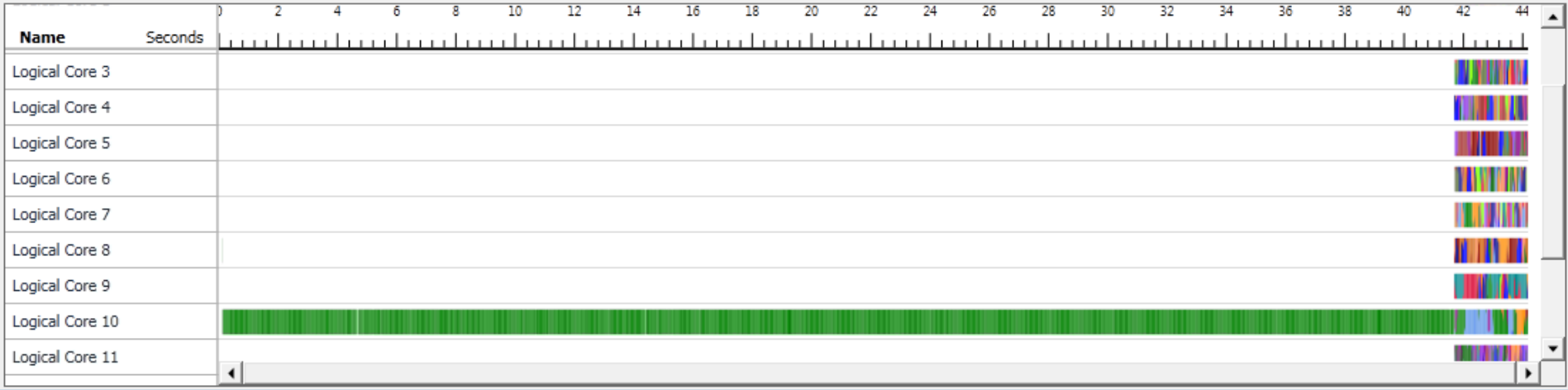
Concurrency (3/3)



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)

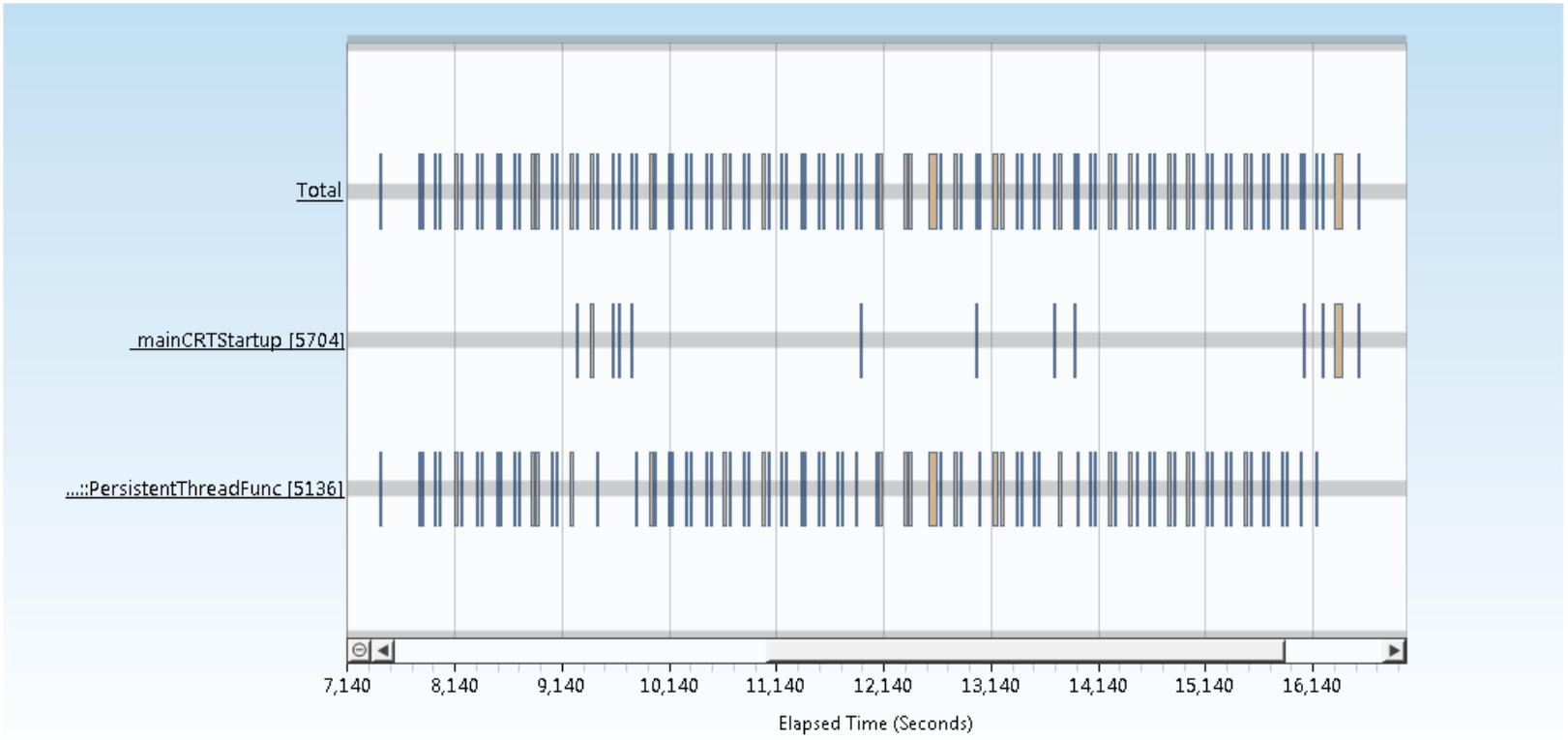
- If threads compete for resources, they can get stalled:

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

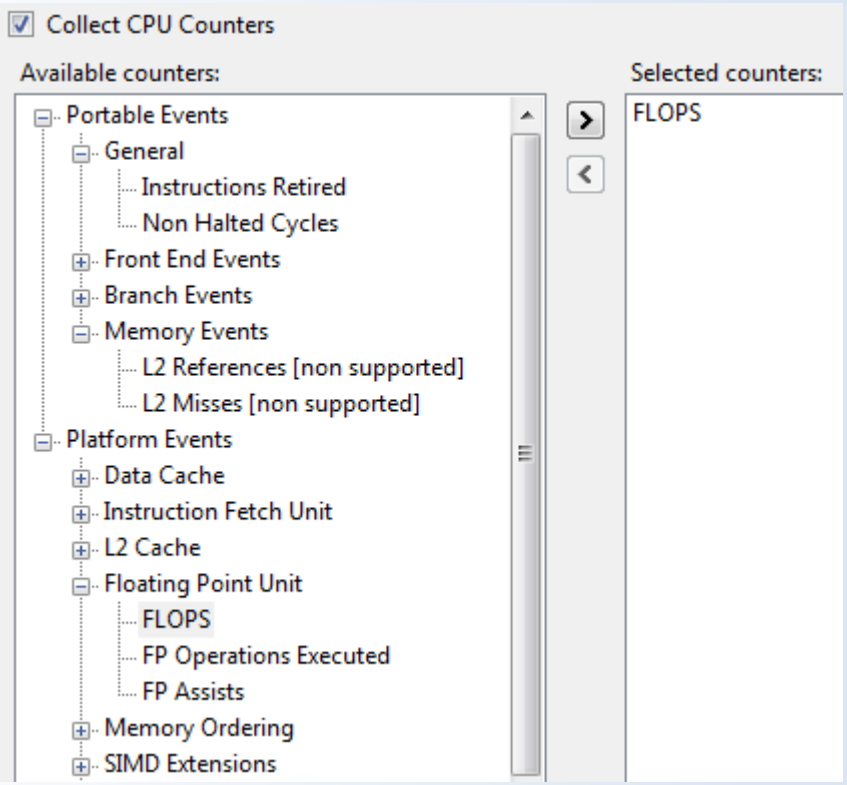
Function Name	Module Name	Inclusive Contentions	Exclusive Contentions	Inclusive Blocked Time	Exclusive Blocked Time
jacobi_aut.exe		97	0	1.518	0
Jacobiomp1	jacobi_aut.exe	84	0	1.346	0
__vcomp_barrier	VCOMP90D.DLL	84	84	1.346	1.346
_mainCRTStartup	jacobi_aut.exe	13	0	172	0
Unknown Frame(s)	UNKNOWN	13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

- This analysis is crucial if you do your own synchronization!

Performance Analyzer: More future 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!

...

Agenda

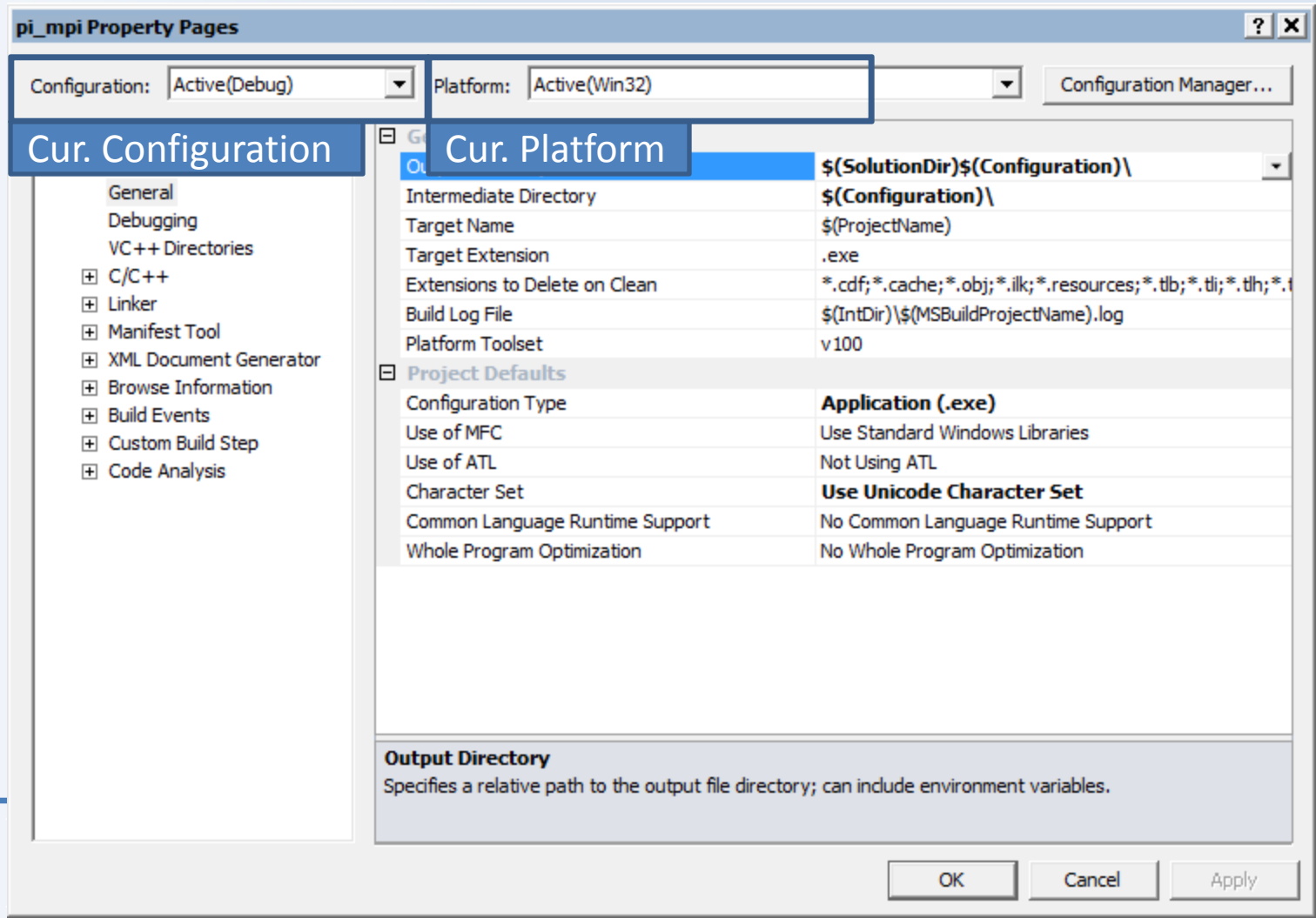
- Overview and Project Management
- The Microsoft and Intel compilers
- Using MPI and OpenMP
- Debugging OpenMP programs
- Debugging MPI programs

Visual Studio Configurations (1/3)

- The set of compiler options is managed in a *Configuration*.
- There are two configurations pre-defined: *Debug* and *Release*.
  - Debug: typical options for debugging, no optimization.
  - Release: debugging still possible, some optimization options.
- The compile process can be triggered by right-clicking on the project and choosing *Build*. Or from the menu: *Build → Build <projectname>*.
- *Build → Build Solution* builds all projects in the solution.
- During and after the compile process compiler output (informational messages, warnings, errors) is displayed in the tool windows *Output* or *Error List*.
- By double-clicking on such a message, the cursor jumps to the corresponding place in the code.

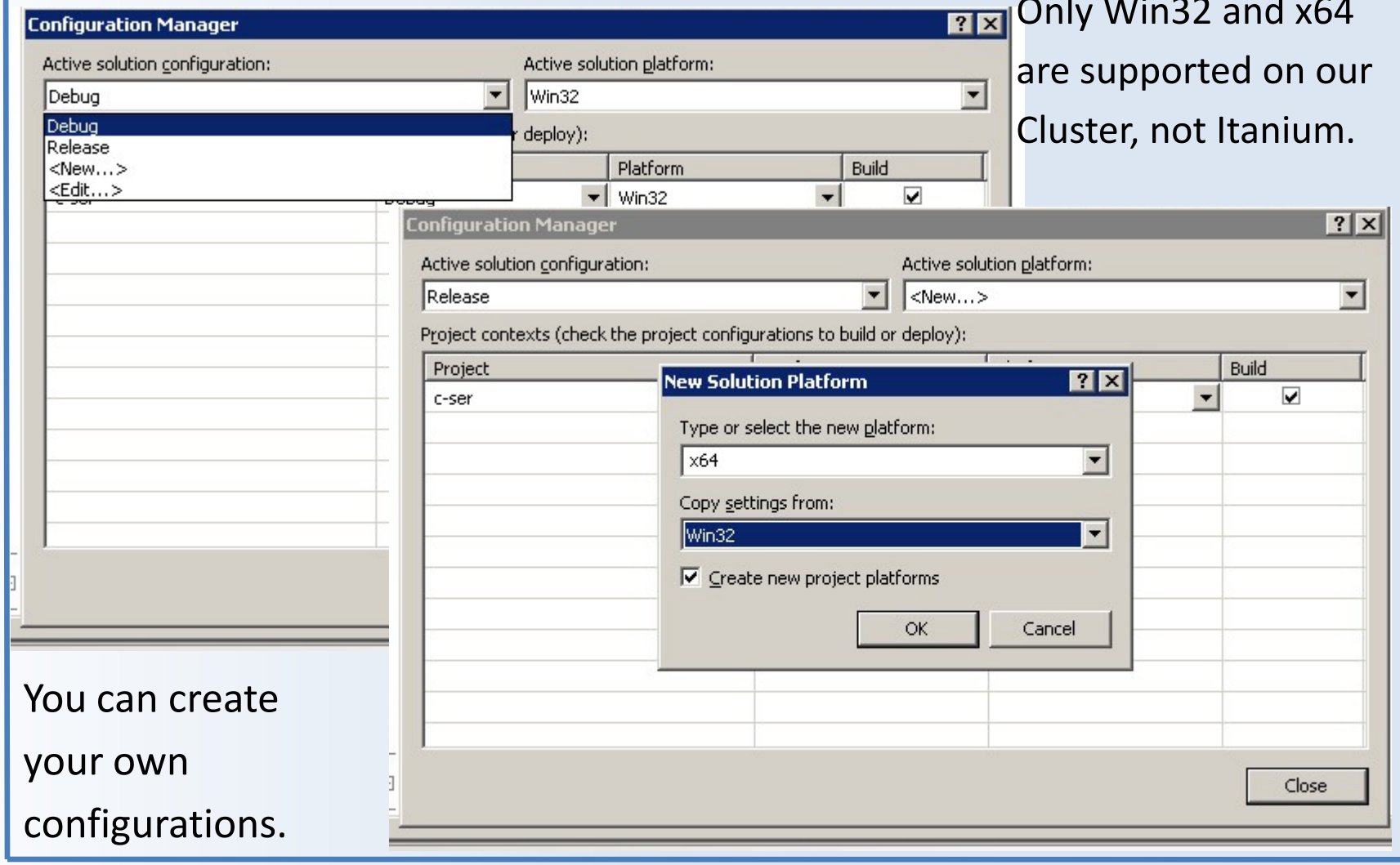
Visual Studio Configurations (2/3)

- Right-clicking on a project and choosing Properties leads to the project configuration dialog.



Visual Studio Configurations (3/3)

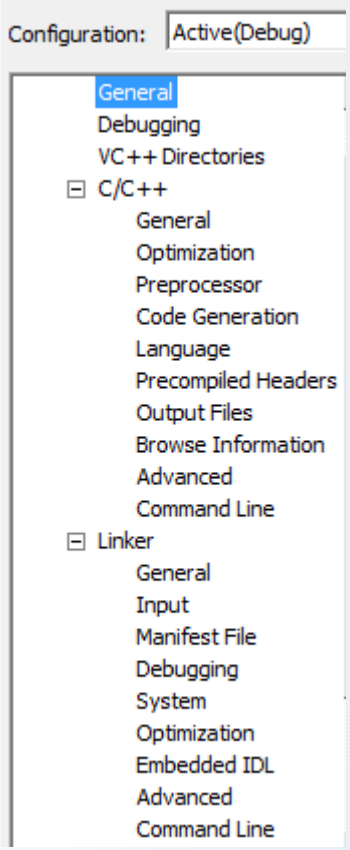
Build → Configuration Manager:



Only Win32 and x64 are supported on our Cluster, not Itanium.

You can create your own configurations.

Microsoft C/C++-specific settings



- Important General Settings:
  - C/C++ → General
    - Addition Include Directories: Include Path
  - Linker → General
    - Additional Library Directories: Library Path
  - Linker → Input
    - Additional Dependencies: Libraries to be used
- Important Optimization Settings:
  - C/C++ → Optimization
    - Optimization: General Optimization Level
    - Inline Function Expansion: Inlining
  - C/C++ → Code Generation
    - Enable Enhanced Instruction Set: Vectorization

Using the Intel C/C++ compiler

1. Create a (Win32) project using the Microsoft C/C++ compiler
2. Right-click on the Solution or on the Project
3. Intel Parallel Studio Installed (`cluster-win-beta` + `-lab`): Intel Parallel Composer → Use Intel C++
4. Intel Parallel Studio Not Installed (`cluster-win`): Use Intel(R) C++

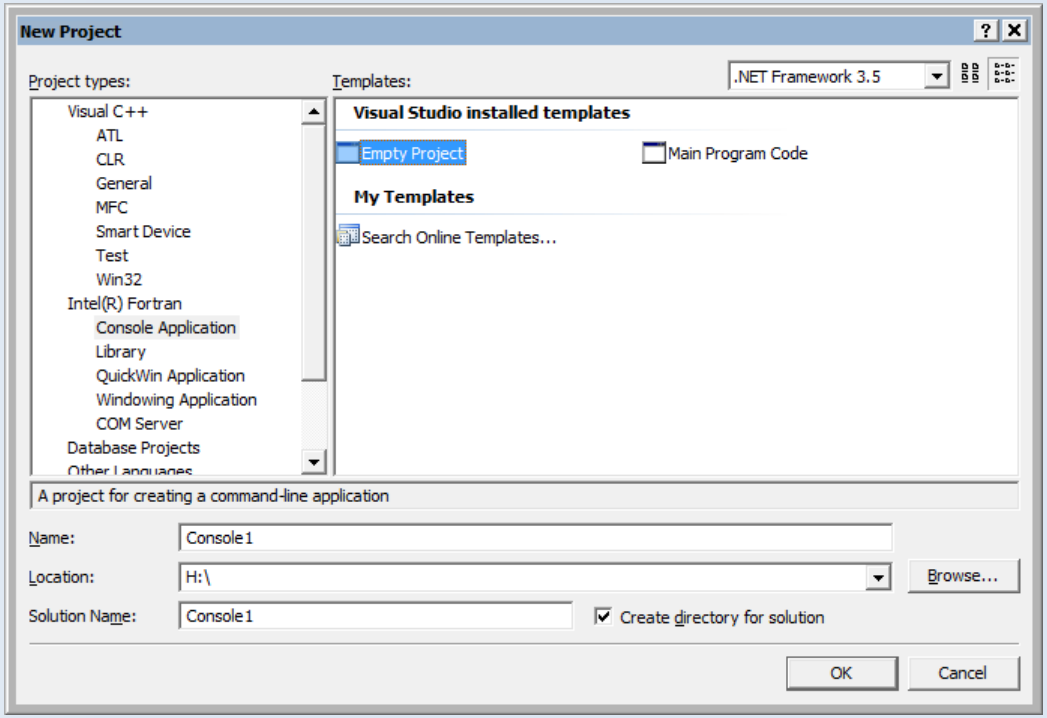
- Solution or Project can be converted back to use Microsoft C/C++ as well.
- Currently only supported in Visual Studio 2008!

Intel C/C++-specific settings

- Most settings are the same as with Microsoft C/C++, but:
- Additional Optimization Settings:
  - C/C++ → Optimization
    - Generate Alternate Code Path and/or Use Intel(R) Processor Extensions: Optimization for specific CPU
    - Parallelization: Ask the compiler for automatic parallelization
  - C/C++ → Language
    - Recognize the Restrict Keyword: Enable C99 `restrict` (Tuning!)
    - Enable C++0x Support: Enable first C++0x features
  - Add to C/C++ → Command Line and Linker → Command Line:
    - /Qtcheck: Enable source instrumentation for Thread Checker
    - /Qtprofile: Enable source instrumentation for Thread Profiler

Using the Intel Fortran compiler

1. Open the Project Creation dialog via *File → New → Project…*
2. Select *Intel(R) Fortran* as project type



3. Typically select *Empty Project* as best-suited option

- Remaining project handling does not differ from C/C++!

Portable Time Measurement (1/3)

- Porting applications from Unix to Windows (or the other way around) can be quite hard … but it was not for most user codes (HPC) we tried on Windows.
  (1) The most common problem was time measurement as `gettimeofday()` is not available on Windows,
  (2) followed by directory management issues where ‚/‘ instead of ‚\‘ had been used before.

- In most cases we attacked (2) using `#ifdefs`.
- Handling (1) depends on the programming language:
  - C++: We have written a version of `double realtime()` for Windows and Unix.
  - FORTRAN: As the library (defined along with the language) already provides time measurement facilities, we used these.

Portable Time Measurement (2/3)

```
#ifdef WIN32
   #include <Windows.h>
   #define Li2Double(x) ((double)((x).HighPart) * 4.294967296E9 + \
      (double)((x).LowPart))
#else
  #include <sys/time.h>
  #include <time.h>
#endif

double realtime (void) {
#ifdef WIN32
   LARGE_INTEGER time, freq;
   double dtime, dfreq;
   if (QueryPerformanceCounter(&time) == 0) { … error … }
   if (QueryPerformanceFrequency(&freq) == 0) { … error … }
   return Li2Double(time) / Li2Double(freq);
#else
   struct timeval tv;
   gettimeofday(&tv, (struct timezone*)0);
   return ((double)tv.tv_sec + (double)tv.tv_usec / 1000000.0 );
}
```

Portable Time Measurement (3/3)

- Taking time the MPI way:

```
#include <mpi.h>
   ...
   double t1, t2, elapsed_seconds;
   t1 = MPI_Wtime();
   ...
   t2 = MPI_Wtime();
   elapsed_seconds = t2 - t1;
```

- Taking time the OpenMP way:

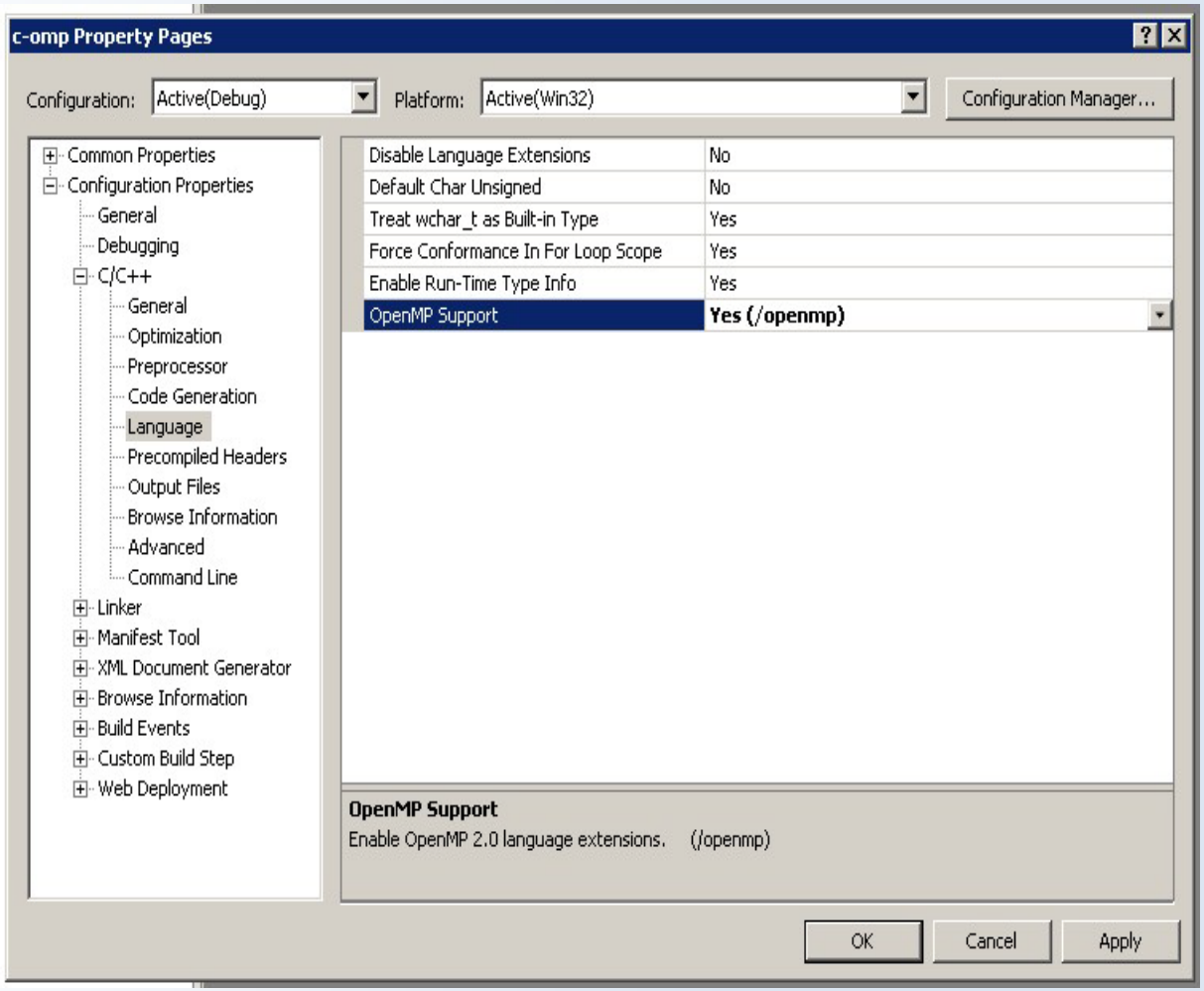
```
#include <omp.h>
   ...
   double t1, t2 elapsed_seconds;
   t1 = omp_get_wtime();
   ...
   t2 = omp_get_wtime();
   elapsed_seconds = t2 - t1;
```

Agenda

- Overview and Project Management
- The Microsoft and Intel compilers
- Using MPI and OpenMP
- Debugging OpenMP programs
- Debugging MPI programs

Enabling OpenMP (1/3)

- OpenMP support has to be enabled in a configuration:



OpenMP 2.0 / 2.5:

- VS2005 C/C++
- VS2008 C/C++
- VS2010 C/C++

OpenMP 3.0:

- Intel C/C++
- Intel FORTRAN

Enabling OpenMP (2/3)

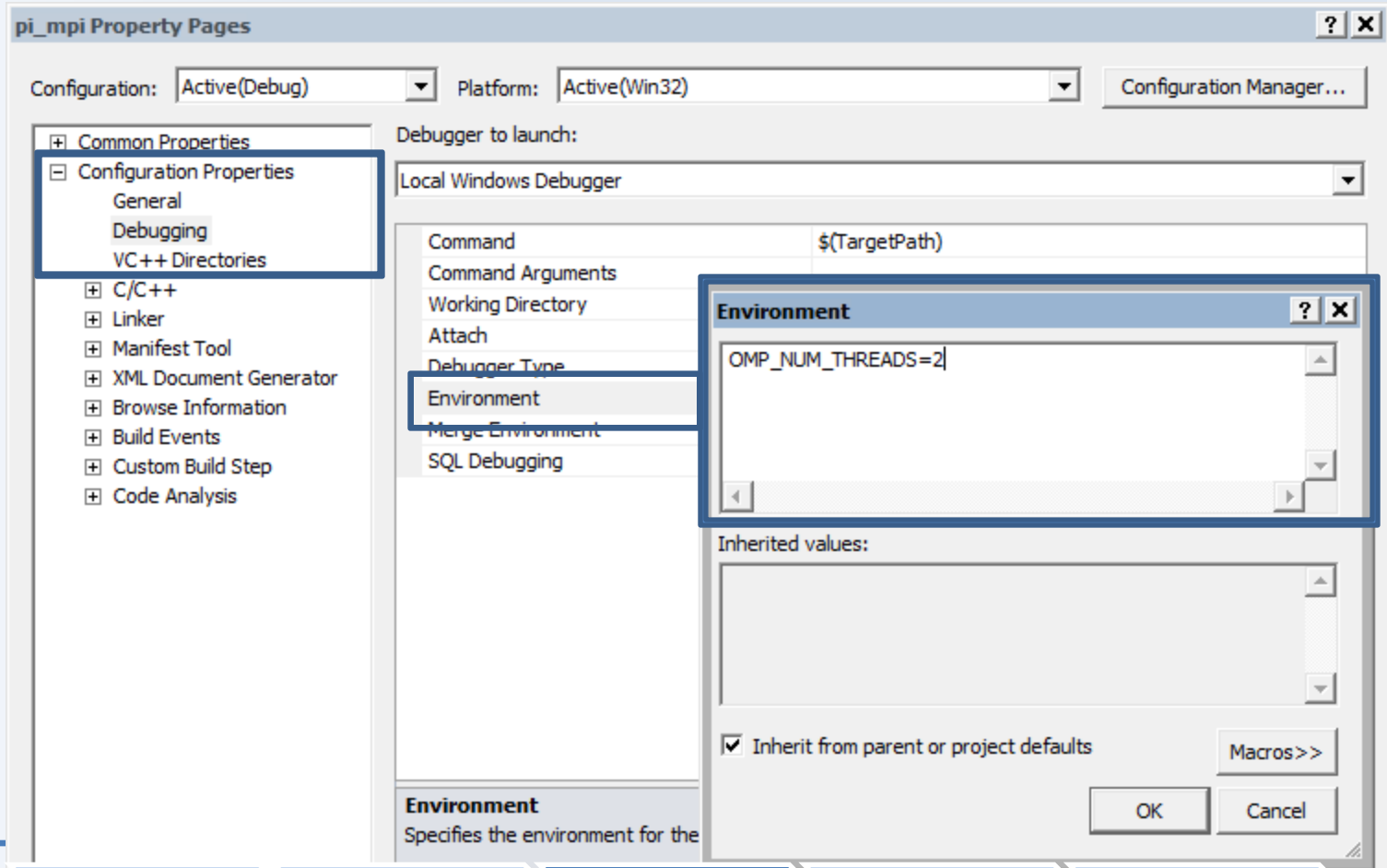
- Known problem with Visual Studio and OpenMP:

- The message appears if an OpenMP program has been compiled with OpenMP support enabled, but `omp.h` had not been included.

- Solution: include `omp.h` in at least one file per project.

Enabling OpenMP (3/3)

- Setting the number of threads for debugging of OpenMP programs: set environment variable `OMP_NUM_THREADS`.



Enabling MPI (1/2)

- As MPI is implemented by a library, an application includes a file containing the type and function declarations named `mpi.h` and has to be linked with that library.

- Modify the project properties (1/2):
  - Include Path: *C/C++ → General → Additional Include Directories*
    - MS-MPI 2008 on cluster-win / cluster-win-lab :
      C:\Program Files\Microsoft HPC Pack 2008 SDK\Include
    - I-MPI on cluster-win:
      C:\Program Files (x86)\Intel\ICT\3.1\mpi\3.1\[ia32|em64t]\include

Enabling MPI (2/2)

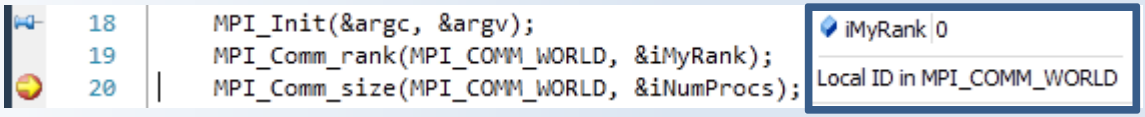
- Modify the project properties (2/2):
  - Library Path: *Linker → General → Additional Library Directories*
    - MS-MPI 2008 on cluster-win / cluster-win-lab :
      C:\Program Files\Microsoft HPC Pack 2008 SDK\Lib\[i386|amd64]
    - I-MPI on cluster-win:
      C:\Program Files (x86)\Intel\ICT\3.1\mpi\3.1\[ia32|em64t]\lib
- No significant performance difference, so our advise:
  - Use MS-MPI with Visual Studio MPI Debugger
  - Use I-MPI with Intel Thread Analyzer & Collector
  - Sometimes a program does not like a specific MPI, so it is always a good thing to have a second one available…

Agenda

- Overview and Project Management
- The Microsoft and Intel compilers
- Using MPI and OpenMP
- Debugging OpenMP programs
- Debugging MPI programs

Debugging Basics (1/2)

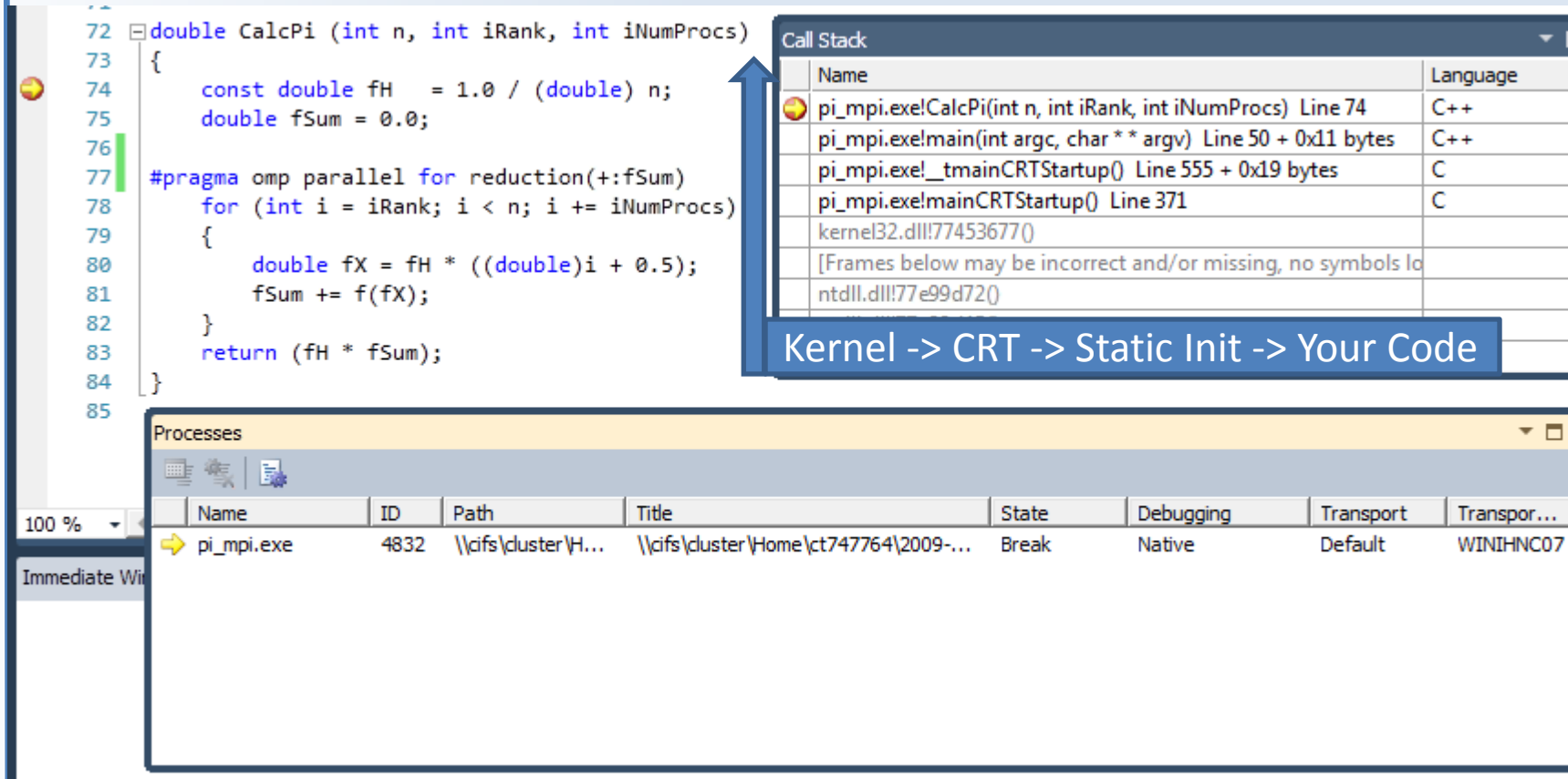
- A breakpoint can be set by clicking in the grey area left of the line number. Clicking again removes the breakpoint.



- Right-clicking a breakpoint opens the context menu with the following functions
  - Disable a breakpoint (temporarily)
  - Set breakpoint trigger conditions
  - Trigger filter for selected threads or processes
  - Define actions to be executed when the breakpoint is triggered
- Just hold the mouse over a variable for a short moment to get the actual value displayed (*hover*). This is also possible for expression (with some limitations).
  - Clicking the *pin* laminates the variable display

Debugging Basics (2/2)

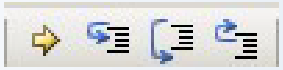
- During a debugging session, the actual program location is marked by a yellow arrow. You can drag this arrow up/down.



Debugging OpenMP Programs (1/4)

- Debugging of OpenMP applications in Visual Studio works with all compilers: the Microsoft C/C++ compiler, the Intel C/C++ compiler and the Intel Fortran compiler.
- Note: If you use one of the Intel compilers or VS21010 and start a program with n threads, you will see n+1 threads (one management thread).
- We advise you to compile without any optimization for debugging, that means use the pre-configured *Debug* configuration and just enable OpenMP.
- Control debugging:

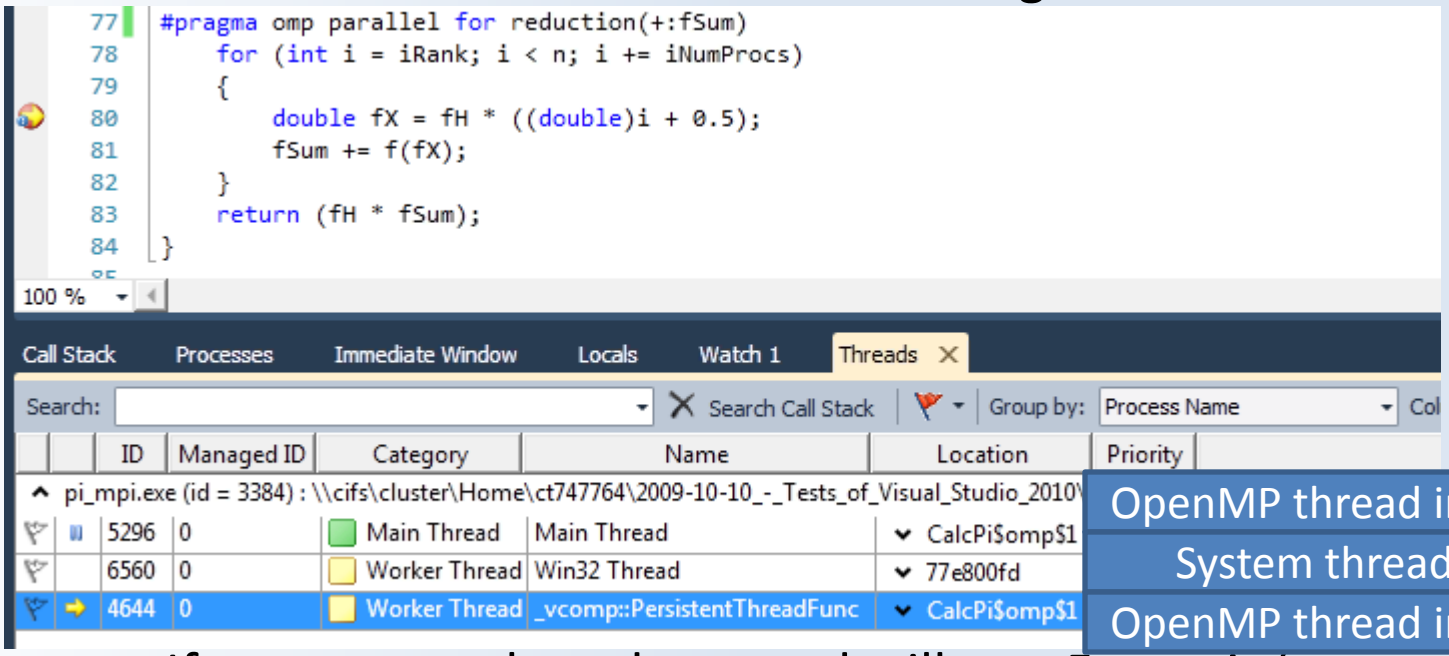
» Start / Continue, Break, Stop, Restart



» Show next statement, Step Into, Step Over, Step Out

Debugging OpenMP Programs (2/4)

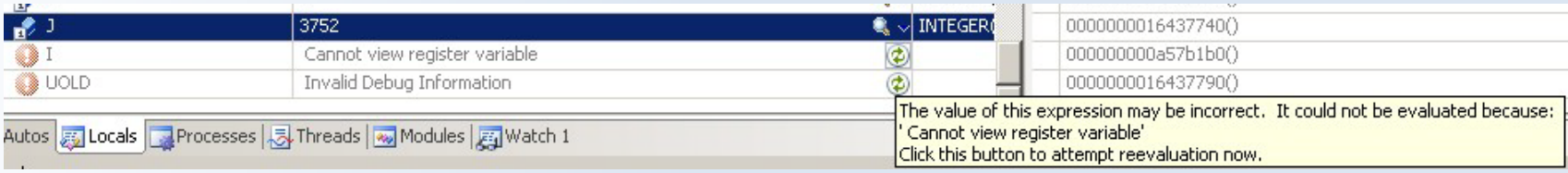
- All threads stop at a breakpoint (first thread encounters it).
  - You can open the *Threads* register from the menu via *Debug* → *Windows* → *Threads*. Double-clicking a thread will select it.



  - If you want a thread to stand still you *Freeze* it (context menu).
  - If you want it to continue *Thaw* it.

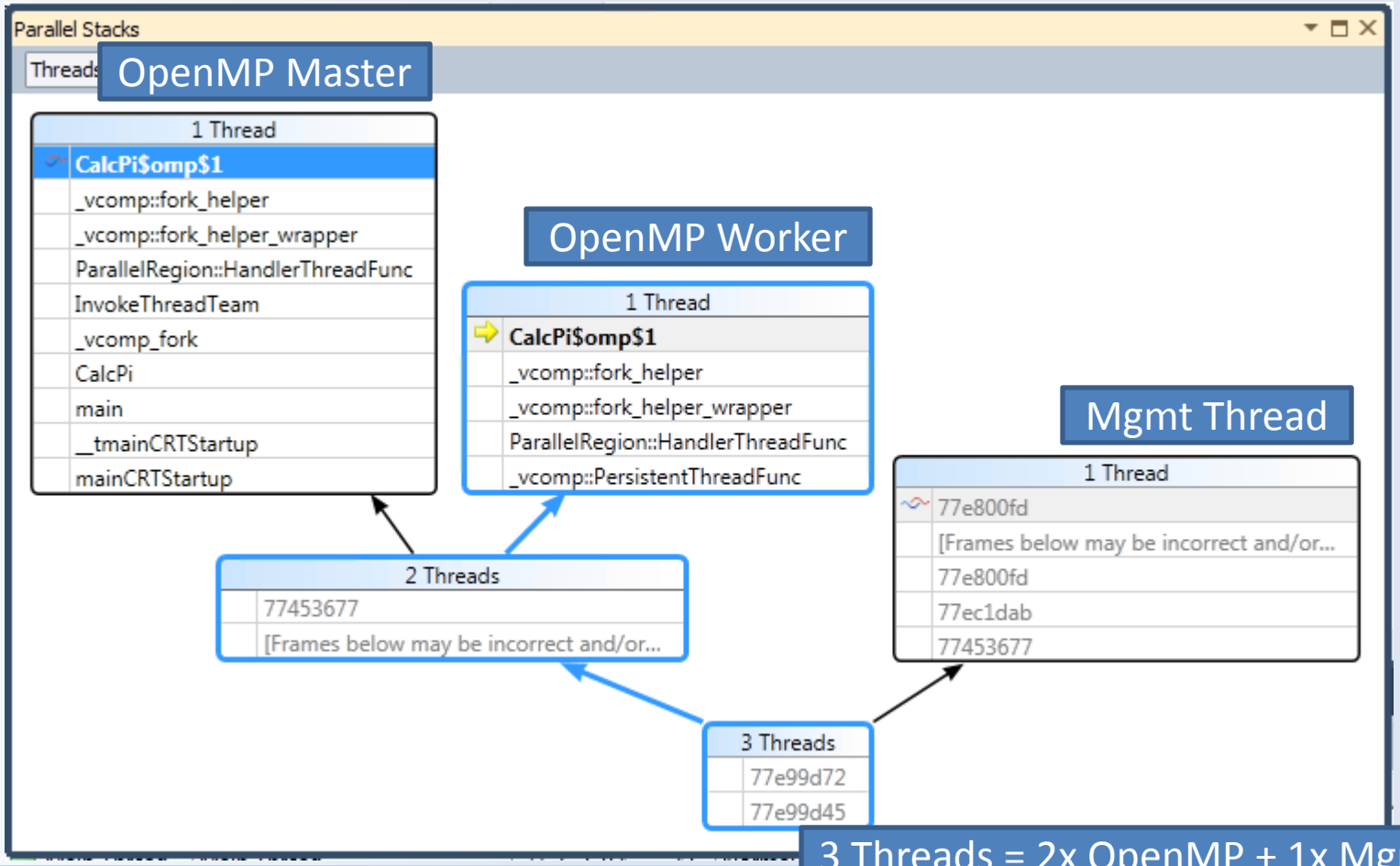
Debugging OpenMP Programs (3/4)

- For all threads, you can view the *local* and *shared* variables.
  - The *Locals* register contains all variables of the current scope.
  - The *Autos* register contains a set of interesting variables guessed by the compiler – remarkably good.
- Some limitations when using the Intel Fortran compiler:
  - The *Autos* register is empty, *Locals* is working fine.
  - One can not (at least sometimes) identify the management thread by the name – it is the one you get an error message of no source code being available if you select it ;-)
  - Sometimes expressions have to be updated because of (possible) compiler optimization.



Debugging OpenMP Programs (4/4)

- The Parallel Stacks window is a new feature of VS2010 and in the menu under *Debug* → *Windows* → *Parallel Stacks*:

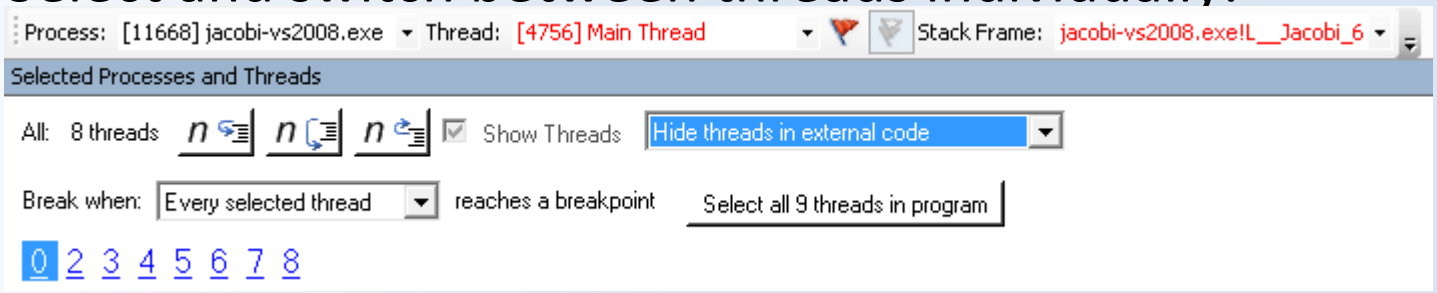


Allinea DDTlite Overview

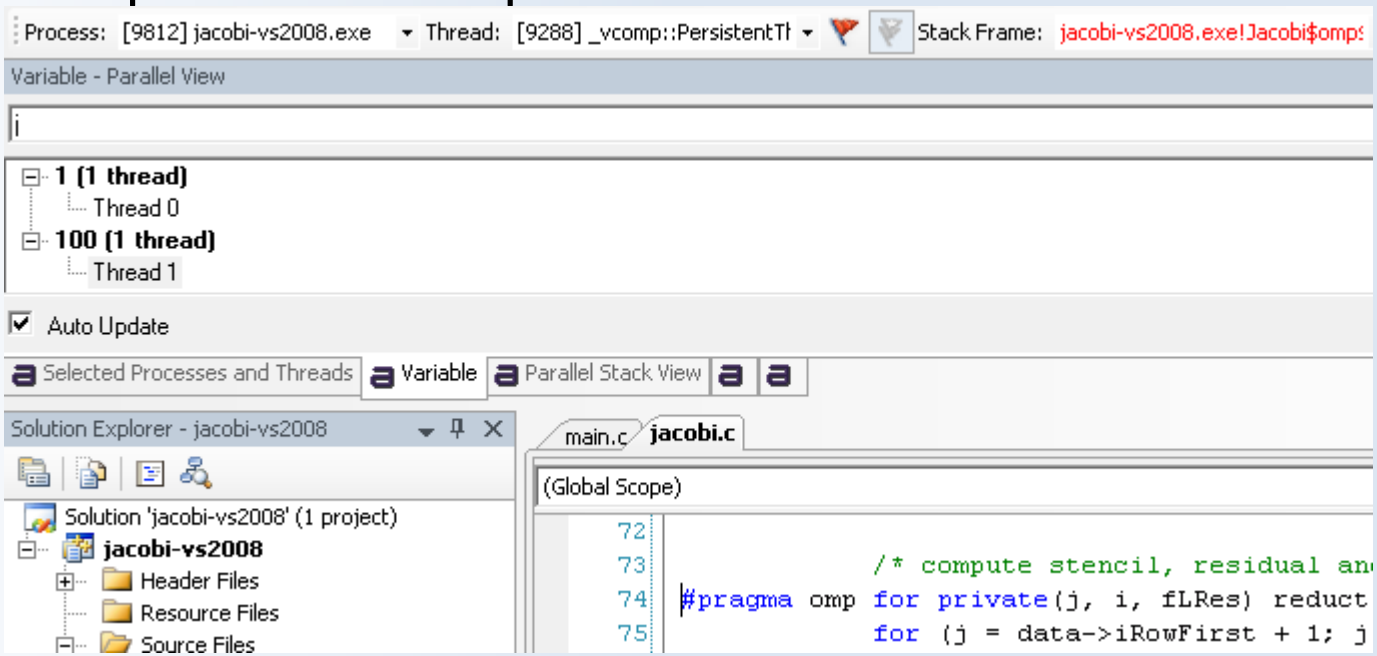
- DDTlite is a plugin for Visual Studio 2008 SP1 (and newer)
  - Mainly marketed for improved MPI debugging experience
  - But also improves OpenMP debugging experience
- Adds the following features:
  - Control processes and threads individually, grouped or together
  - Examine variable values per thread / process
  - Parallel Stack View per thread / process
  - Location View per thread / process
  - Thread / process group management
- For a trial version go to www.allinea.com

Debugging OpenMP Programs w/ DDTlite

- Select and switch between threads individually:



- Compare variables per thread:



Agenda

- Overview and Project Management
- The Microsoft and Intel compilers
- Using MPI and OpenMP
- Debugging OpenMP programs
- Debugging MPI programs

Debugging MPI programs (1/6)

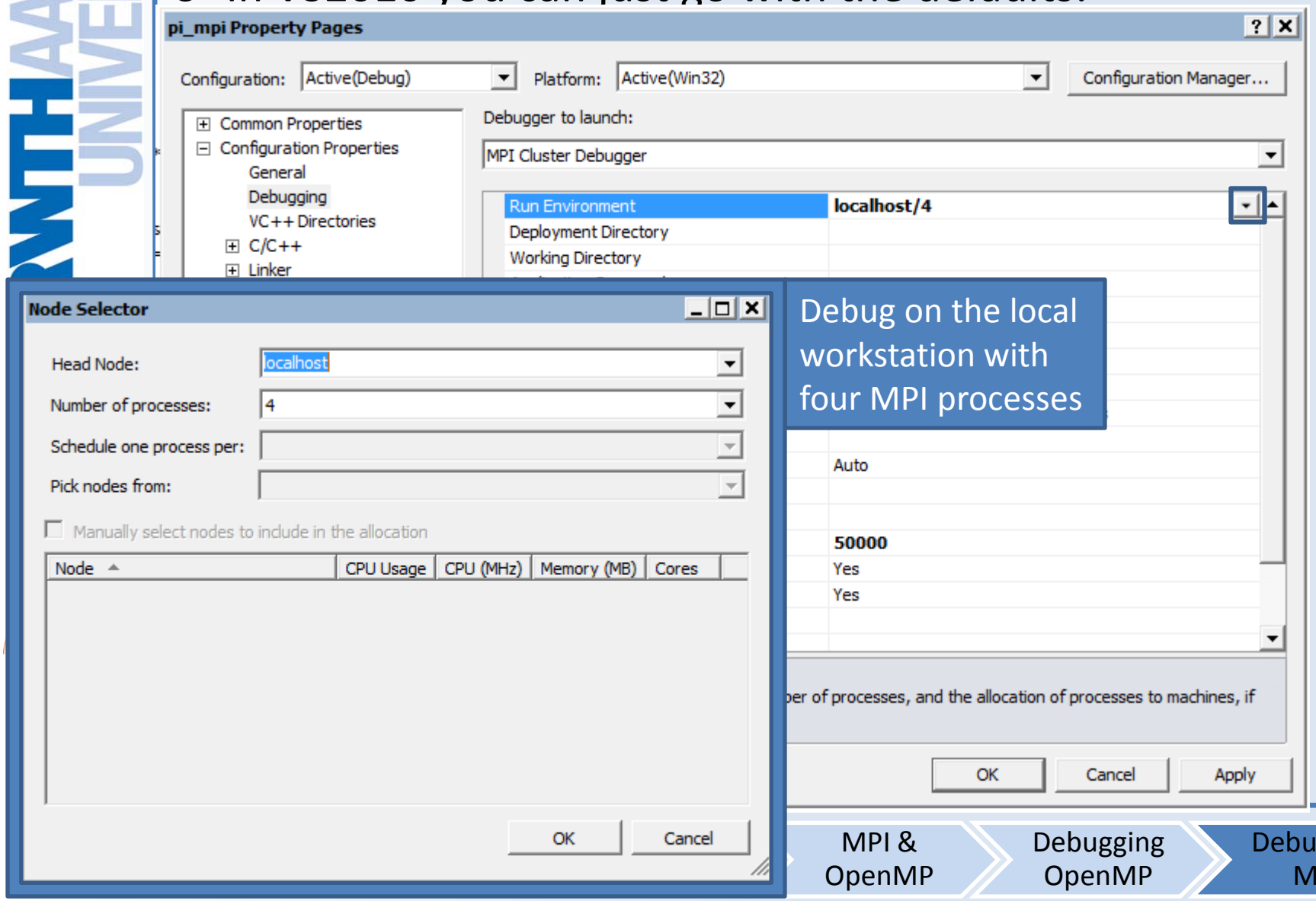
- MS-MPI works best, but you should be able to use I-MPI as well. At least the following instructions work for both.
- Visual Studio supports debugging of MPI programs using the *Cluster Debugger*. As far as I know – or was able to verify – the cluster debugger only works with the Microsoft C/C++ compiler and not with projects using the Intel C/C++ compiler or the Intel FORTRAN compiler.
- In the project properties under *Debugging*, choose the *MPI Cluster Debugger* as *Debugger to launch*. For VS2008 only:
  - MPIRun: *„C:\Program Files\Microsoft HPC Pack 2008 SDK\Bin “*
  - MPIRun Arguments: for example *–n 2*
  - MPIShim Location:
    It is not possible to specify a path containing empty spaces here, so you have to copy MPIShim from *c:\program files[(x86)]\microsoft visual studio 9.0 \common7\ide\RemoteDebugger\x86[or x64]\MPIShim* to a suitable location.

Debugging MPI programs (2/6)

- In order to stop all processes at a breakpoint, please check for the following option: In *Tools* → *Options* → *Debugging* → *General* the checkbox *Break all processes when one process breaks* has to be activated.
- Select the current process using the *Processes* register.
- Problem: if one process does a step, all other processes make a step as well.
  - Solution 1: Be clever setting breakpoints …
  - Solution 2: Detach the second (and third and fourth and …) process from the VS debugger (remember the right-click and context menu), open up another VS instance (maybe more) and attach to the process.
  - Solution 3: **Use DDTlight**

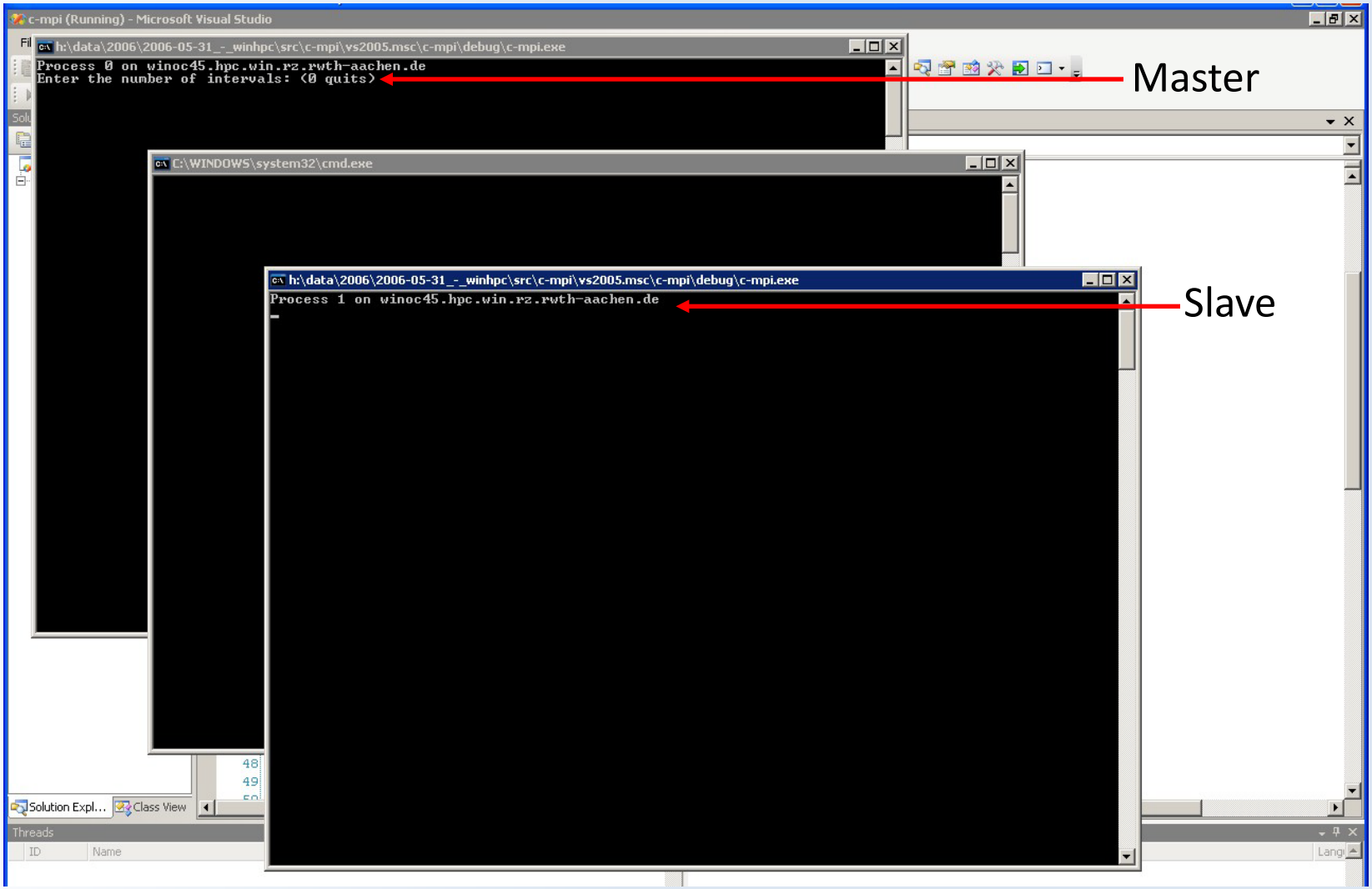
Debugging MPI programs (3/6)

- In VS2010 you can just go with the defaults:



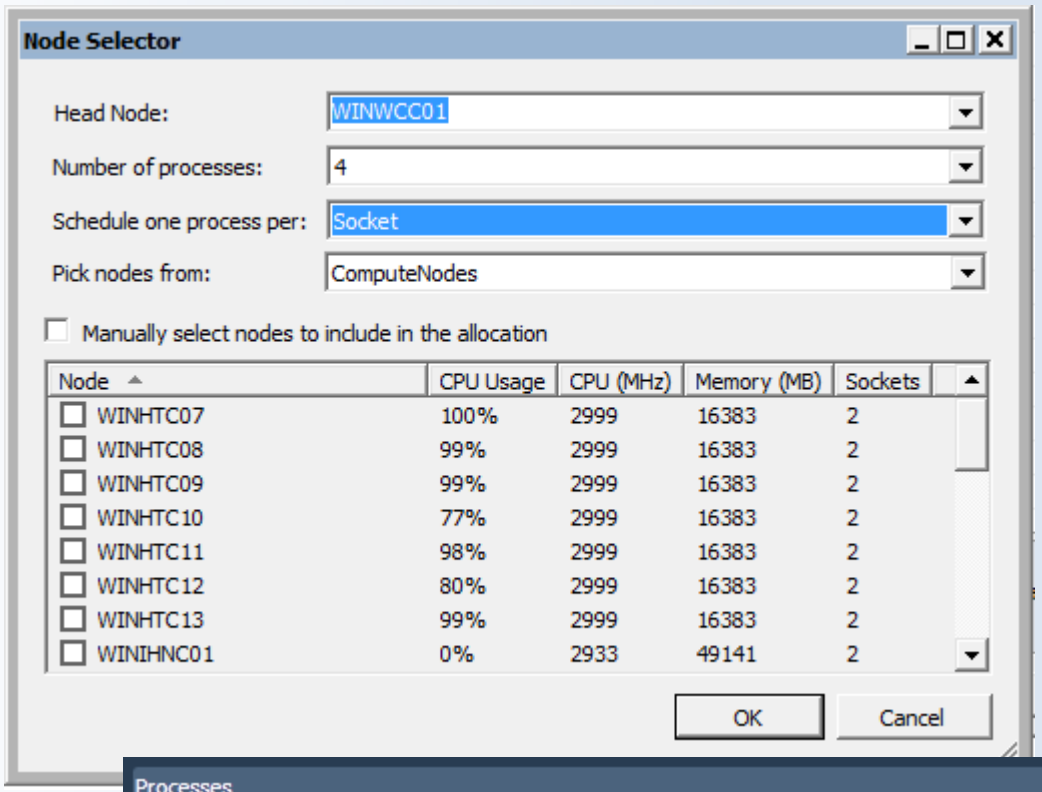
Debug on the local workstation with four MPI processes

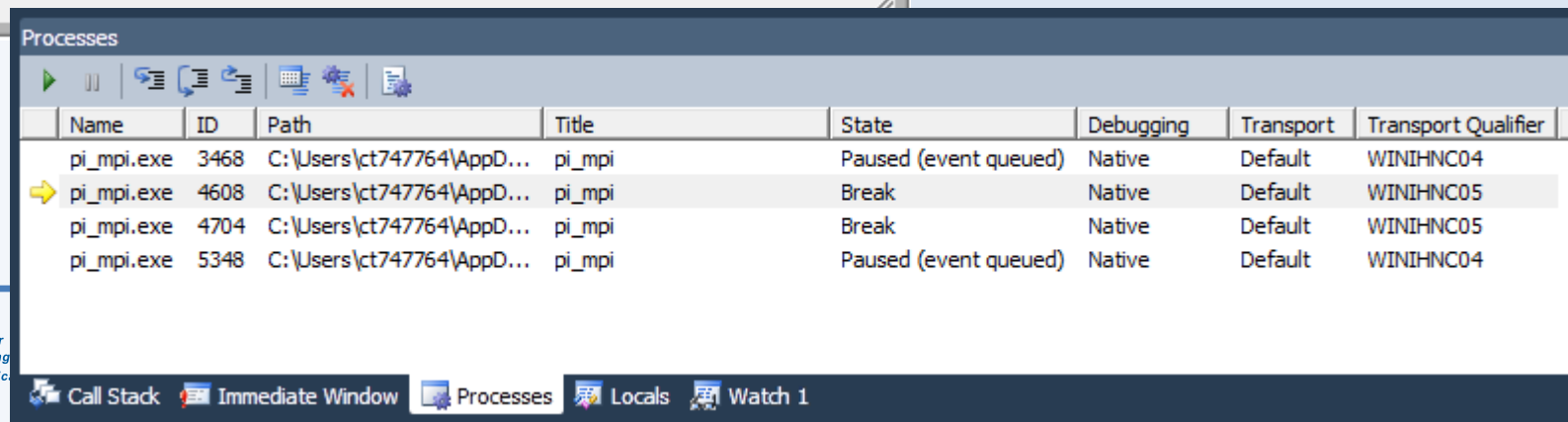
Debugging MPI programs (4/6)



Debugging MPI programs (5/6)

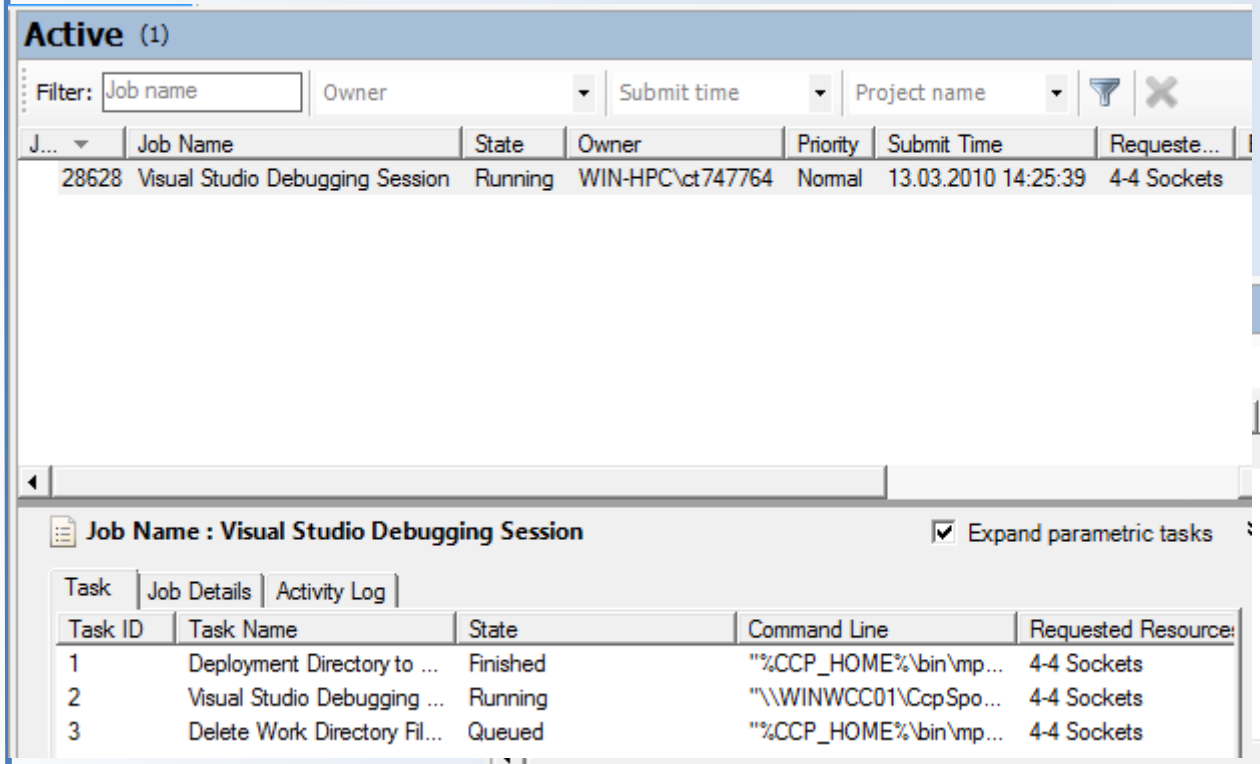
- You can also *F5 to the Cluster*, if there are free slots:

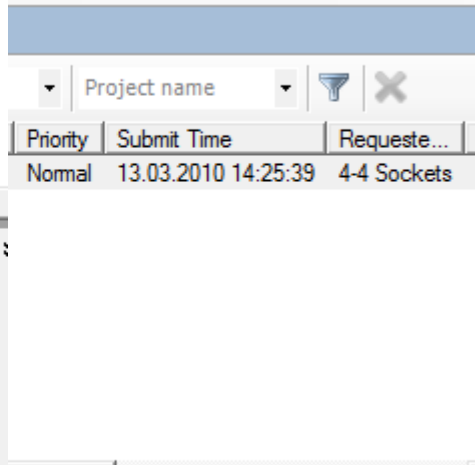


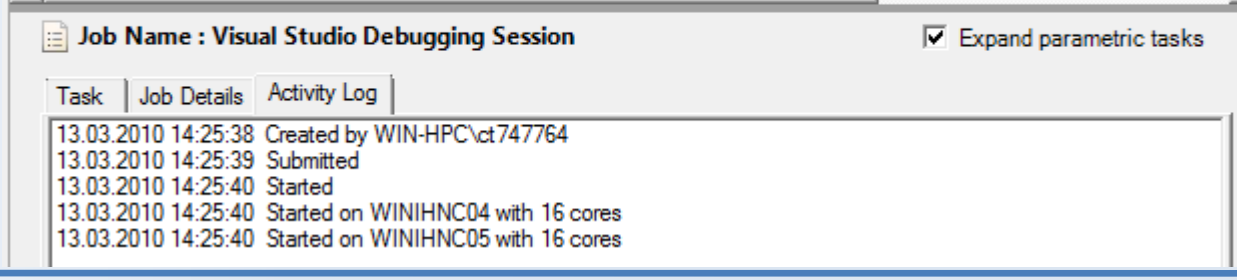


Debugging MPI programs (6/6)

- A VS2010 debug job running on our Cluster:







DDTlite: Overview

- Allinea DDT Lite is an add-in for Visual Studio 2008 SP1
  - Currently an additional patch to VS2008 is required
- Significantly improves the MPI debugging experience
  - Debug / Control MPI processes individually
  - Debug / Control groups of MPI processes individually
  - Display variable values per process side-by-side
  - Display MPI process stacks side-by-side
  - …
- For a trial version go to www.allinea.com

DDTlite: Usage

- Configure project to use the MPI Cluster Debugger
- Tools -> Add-in Manager
  - Select Allinea DDTlite
  - Enable *Startup*
- Start Debugging (e.g. via F5)
- Debug -> Windows -> Show DDTLite Windows
  - Selected Processes and Threads
  - Groups – Parallel View
  - Variable – Parallel View
  - Location – Parallel View
  - Parallel Stack View

The End

Thank you for

your attention!