MPI

Message Passing Interface Quick Reference in C

```
#include <mpi.h>
```

Blocking Point-to-Point

Send a message to one process. (§3.2.1)

```
int MPI_Send (void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)
```

Receive a message from one process. (§3.2.4)

```
int MPI_Recv (void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)
```

Count received data elements. (§3.2.5)

```
int MPI_Get_count (MPI_Status *status,
    MPI_Datatype datatype, int *count)
```

Wait for message arrival. (§3.8)

```
int MPI_Probe (int source, int tag,
    MPI_Comm comm, MPI_Status *status)
```

Related Functions: MPI_Bsend, MPI_Ssend, MPI_Rsend, MPI_Buffer_attach, MPI_Buffer_detach, MPI_Sendrecv, MPI_Sendrecv_replace, MPI_Get_elements

Non-blocking Point-to-Point

Begin to receive a message. (§3.7.2)

```
int MPI_Irecv (void *buf, int count,
    MPI_Datatype, int source, int tag,
    MPI_Comm comm, MPI_Request *request)
```

Complete a non-blocking operation. (§3.7.3)

```
int MPI_Wait (MPI_Request *request,
    MPI_Status *status)
```

Check or complete a non-blocking operation. (§3.7.3)

```
int MPI_Test (MPI_Request *request, int
    *flag, MPI_Status *status)
```

Check message arrival. (§3.8)

```
int MPI_Iprobe (int source, int tag,
    MPI_Comm comm, int *flag, MPI_Status
    *status)
```

Related Functions: MPI_Isend, MPI_Ibsend, MPI_Issend, MPI_Irsend, MPI_Request_free, MPI_Waitany, MPI_Testany, MPI_Waitall, MPI_Testall, MPI_Waitsome, MPI_Testsome, MPI_Cancel, MPI_Test_cancelled

Persistent Requests

Related Functions: MPI_Send_init, MPI_Bsend_init, MPI_Ssend_init, MPI_Rsend_init, MPI_Recv_init, MPI_Start, MPI_Startall

Derived Datatypes

Create a strided homogeneous vector. (§3.12.1)

```
int MPI_Type_vector (int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

Save a derived datatype (§3.12.4)

```
int MPI_Type_commit (MPI_Datatype
    *datatype)
```

Pack data into a message buffer. (§3.13)

```
int MPI_Pack (void *inbuf, int incount,
    MPI_Datatype datatype, void *outbuf,
    int outsize, int *position, MPI_Comm
    comm)
```

Unpack data from a message buffer. (§3.13)

```
int MPI_Unpack (void *inbuf, int insize,
    int *position, void *outbuf, int
    outcount, MPI_Datatype datatype,
    MPI_Comm comm)
```

Determine buffer size for packed data. (§3.13)

```
int MPI_Pack_size (int incount,
    MPI_Datatype datatype, MPI_Comm comm,
    int *size)
```

Related Functions: MPI_Type_contiguous, MPI_Type_hvector, MPI_Type_indexed, MPI_Type_hindexed, MPI_Type_struct, MPI_Address, MPI_Type_extent, MPI_Type_size, MPI_Type_lb, MPI_Type_ub, MPI_Type_free

Collective

Send one message to all group members. (§4.4)

```
int MPI_Bcast (void *buf, int count,
    MPI_Datatype datatype, int root,
    MPI_Comm comm)
```

Receive from all group members. (§4.5)

```
int MPI_Gather (void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

Send separate messages to all group members. (§4.6)

```
int MPI_Scatter (void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

Combine messages from all group members. (§4.9.1)

```
int MPI_Reduce (void *sendbuf, void
    *recvbuf, int count, MPI_Datatype
    datatype, MPI_Op op, int root, MPI_Comm
    comm)
```

Related Functions: MPI_Barrier, MPI_Gatherv, MPI_Scatterv, MPI_Allgather, MPI_Allgatherv, MPI_Alltoall, MPI_Alltoallv, MPI_Op_create, MPI_Op_free, MPI_Allreduce, MPI_Reduce_scatter, MPI_Scan

Groups

Related Functions: MPI_Group_size, MPI_Group_rank, MPI_Group_translate_ranks, MPI_Group_compare, MPI_Comm_group, MPI_Group_union, MPI_Group_intersection, MPI_Group_difference, MPI_Group_incl, MPI_Group_excl, MPI_Group_range_incl, MPI_Group_range_excl, MPI_Group_free

Basic Communicators

Count group members in communicator. (§5.4.1)

```
int MPI_Comm_size (MPI_Comm comm, int
    *size)
```

Determine group rank of self. (§5.4.1)

```
int MPI_Comm_rank (MPI_Comm comm, int
    *rank)
```

Duplicate with new context. (§5.4.2)

```
int MPI_Comm_dup (MPI_Comm comm, MPI_Comm
    *newcomm)
```

Split into categorized sub-groups. (§5.4.2)

```
int MPI_Comm_split (MPI_Comm comm, int
    color, int key, MPI_Comm *newcomm)
```

Related Functions: MPI_Comm_compare, MPI_Comm_create, MPI_Comm_free, MPI_Comm_test_inter, MPI_Comm_remote_size, MPI_Comm_remote_group, MPI_Intercomm_create, MPI_Intercomm_merge

Communicators with Topology

Create with cartesian topology. (§6.5.1)

```
int MPI_Cart_create (MPI_Comm comm_old,
    int ndims, int *dims, int *periods, int
    reorder, MPI_Comm *comm_cart)
```

Suggest balanced dimension ranges. (§6.5.2)

```
int MPI_Dims_create (int nnodes, int
    ndims, int *dims)
```

Determine rank from cartesian coordinates. (§6.5.4)

```
int MPI_Cart_rank (MPI_Comm comm, int
    *coords, int *rank)
```

Determine cartesian coordinates from rank. (§6.5.4)

```
int MPI_Cart_coords (MPI_Comm comm, int
    rank, int maxdims, int *coords)
```

Determine ranks for cartesian shift. (§6.5.5)

```
int MPI_Cart_shift (MPI_Comm comm, int
    direction, int disp, int *rank_source,
    int *rank_dest)
```

Split into lower dimensional sub-grids. (§6.5.6)

```
int MPI_Cart_sub (MPI_Comm comm, int
    *remain_dims, MPI_Comm *newcomm)
```

Related Functions: MPI_Graph_create, MPI_Topo_test, MPI_Graphdims_get, MPI_Graph_get, MPI_Cartdim_get, MPI_Cart_get, MPI_Graph_neighbors_count, MPI_Graph_neighbors, MPI_Cart_map, MPI_Graph_map

Communicator Caches

Related Functions: MPI_Keyval_create, MPI_Keyval_free, MPI_Attr_put, MPI_Attr_get, MPI_Attr_delete

OHIO SUPERCOMPUTER C·E·N·T·E·R

LAM & MPI Information

1224 Kinnear Rd.
Columbus, Ohio 43212
614-292-8492

lam@tbag.osc.edu

http://www.osc.edu/lam.html
ftp://tbag.osc.edu/pub/lam

Error Handling

Related Functions: MPI_Errhandler_create, MPI_Errhandler_set, MPI_Errhandler_get, MPI_Errhandler_free, MPI_Error_string, MPI_Error_class

Environmental

Determine wall clock time. (§7.4)

```
double MPI_Wtime (void)
```

Initialize MPI. (§7.5)

```
int MPI_Init (int *argc, char ***argv)
```

Cleanup MPI. (§7.5)

```
int MPI_Finalize (void)
```

Related Functions: MPI_Get_processor_name, MPI__Wtick, MPI_Initialized, MPI_Abort, MPI_Pcontrol

Constants

Wildcards (§3.2.4)

```
MPI_ANY_TAG, MPI_ANY_SOURCE
```

Elementary Datatypes (§3.2.2)

```
MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG,
MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT,
MPI_UNSIGNED, MPI_UNSIGNED_LONG,
MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE,
MPI_BYTE, MPI_PACKED
```

Reserved Communicators (§5.2.4)

```
MPI_COMM_WORLD, MPI_COMM_SELF
```

Reduction Operations (§4.9.2)

```
MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD,
    MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND,
    MPI_LOR, MPI_LXOR
```

LAM Quick Reference

LAM / MPI Extensions

Spawn processes.

```
int MPIL_Spawn (MPI_Comm comm, char *app,
    int root, MPI_Comm *child_comm);
```

Get communicator ID.

```
int MPIL_Comm_id (MPI_Comm comm, int *id);
```

Deliver an asynchronous signal.

```
int MPIL_Signal (MPI_Comm comm, int rank,
    int signo);
```

Enable trace collection.

```
int MPIL_Trace_on (void);
```

Related Functions: MPIL_Comm_parent, MPIL_Universe_size, MPIL_Type_id, MPIL_Comm_gps, MPIL_Trace_off

Session Management

Confirm a group of hosts.

```
recon -v <hostfile>
```

Start LAM on a group of hosts.

```
lamboot -v <hostfile>
```

Terminate LAM.

```
wipe -v <hostfile>
```

Hostfile Syntax

```
# comment
<hostname> <userid>
<hostname> <userid>
...etc...
```

Compilation

Compile a program for LAM / MPI.

```
hcc -o <binary> <source> -I<incdir>
    -L<libdir> -l<lib> -lmpi
```

Processes and Messages

Start an SPMD application.

```
mpirun -v -s <src_node> -c <copies>
    <nodes> <program> -- <args>
```

Start a MIMD application.

```
mpirun -v <appfile>
```

Appfile Syntax

```
# comment
<program> -s <src_node> <nodes> -- <args>
<program> -s <src_node> <nodes> -- <args>
...etc...
```

Examine the state of processes.

```
mpitask
```

Examine the state of messages.

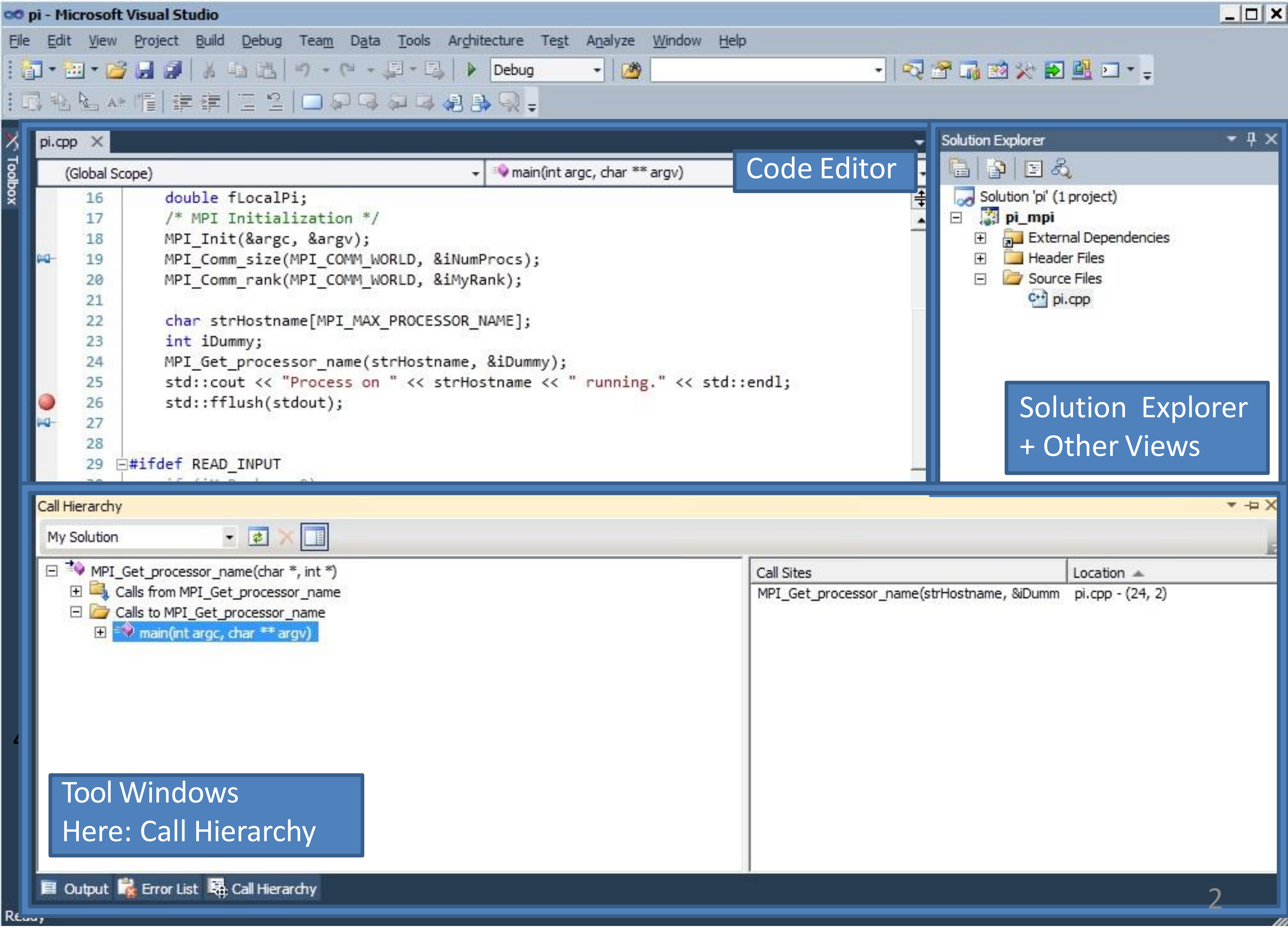
```
mpimsg
```

Cleanup all processes and messages.

```
lamclean -v
```

Visual Studio for parallel programming - Agenda

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

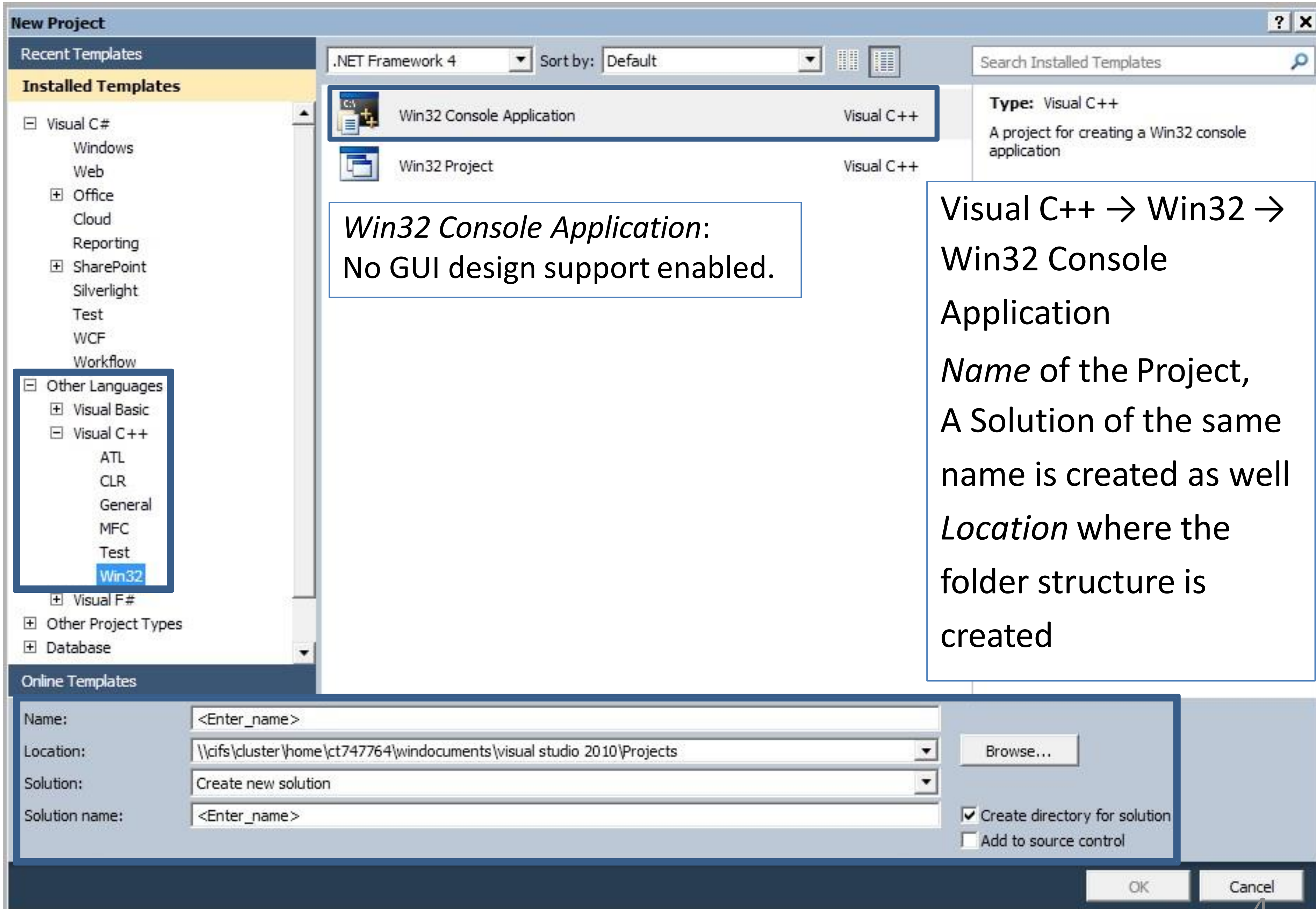

pi - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
pi.cpp
(Global Scope)
main(int argc, char ** argv)
Code Editor
double fLocalPi;
/* MPI Initialization */
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &iNumProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &iMyRank);
char strHostname[MPI_MAX_PROCESSOR_NAME];
int iDummy;
MPI_Get_processor_name(strHostname, &iDummy);
std::cout << "Process on " << strHostname << " running." << std::endl;
std::fflush(stdout);
#ifdef READ_INPUT
Toolbox
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

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…*



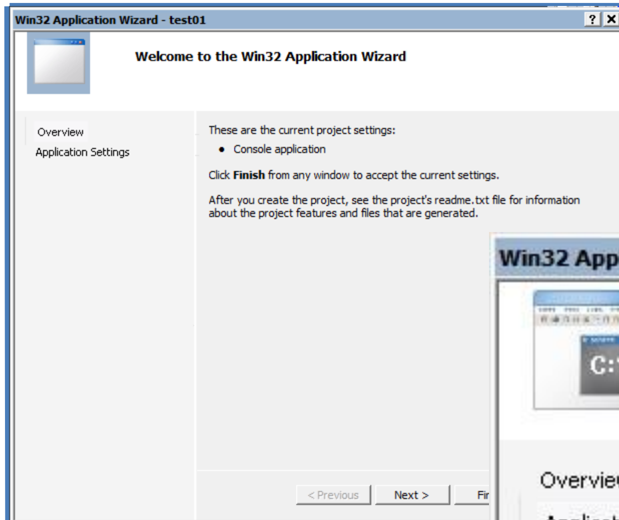
Win32 Console Application:
No GUI design support enabled.

Visual C++ → Win32 → Win32 Console Application

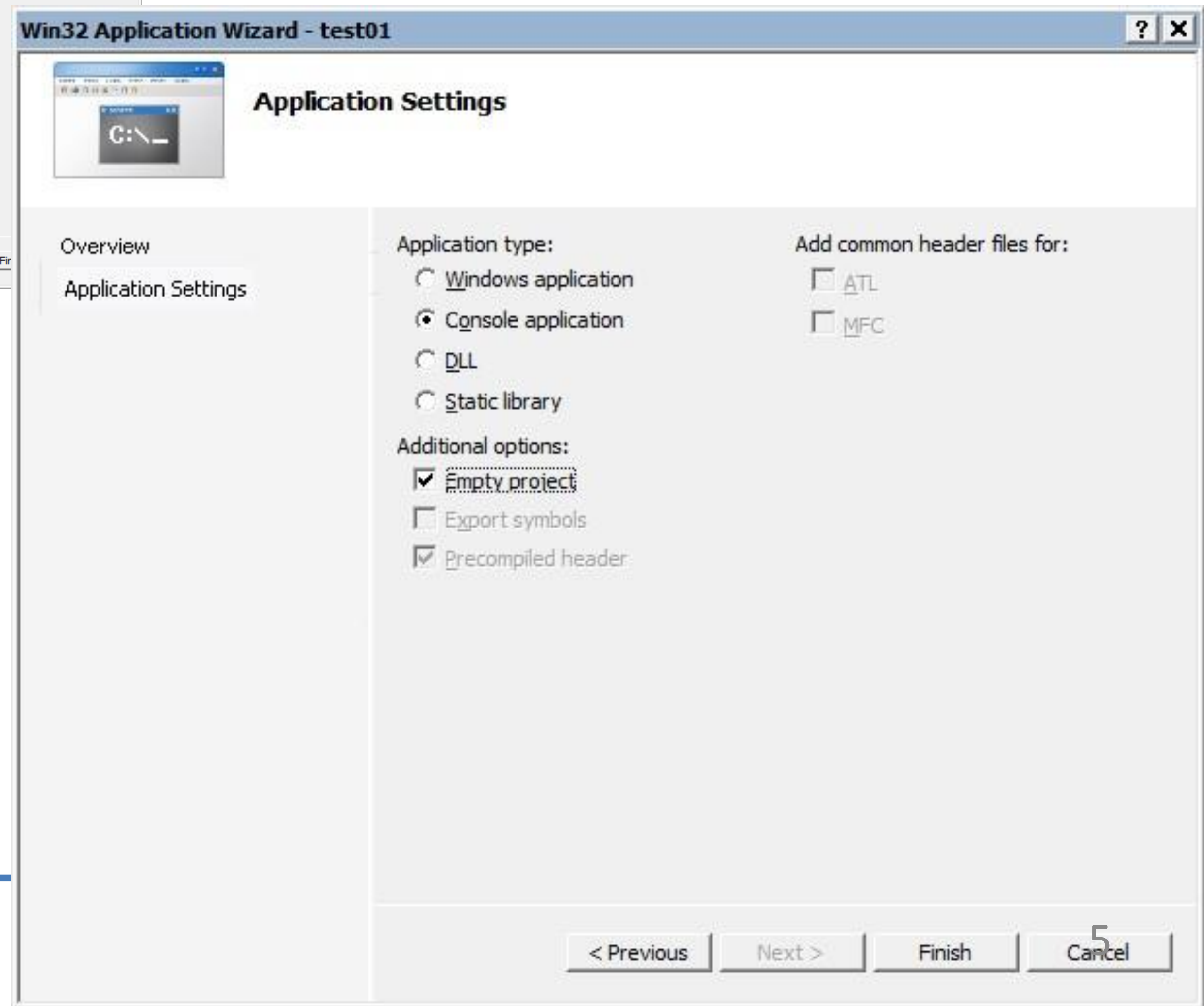
Name of the Project, A Solution of the same name is created as well

Location where the folder structure is created

Visual Studio: Project Management (3/5)

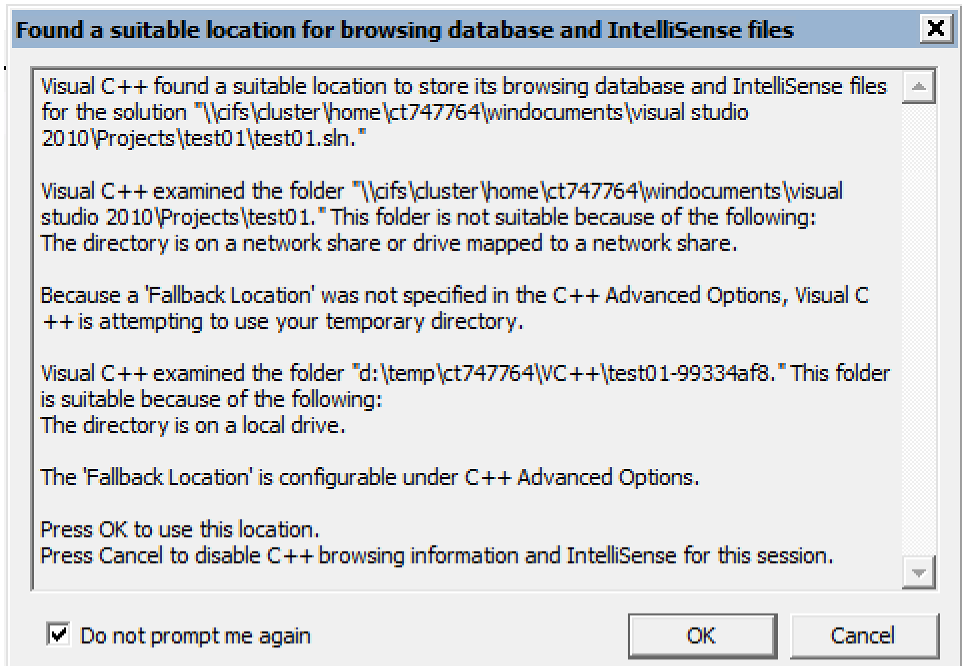


Choose *Empty project* if you already have source files.

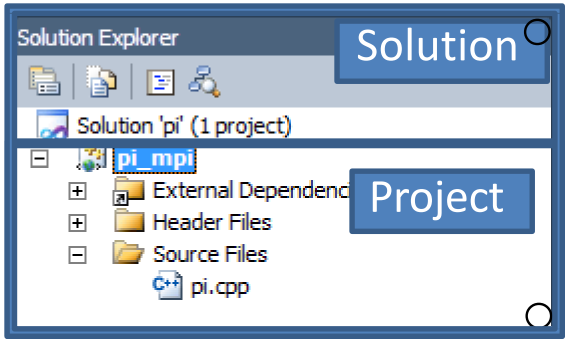


Visual Studio: Project Management (4/5)

- 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)

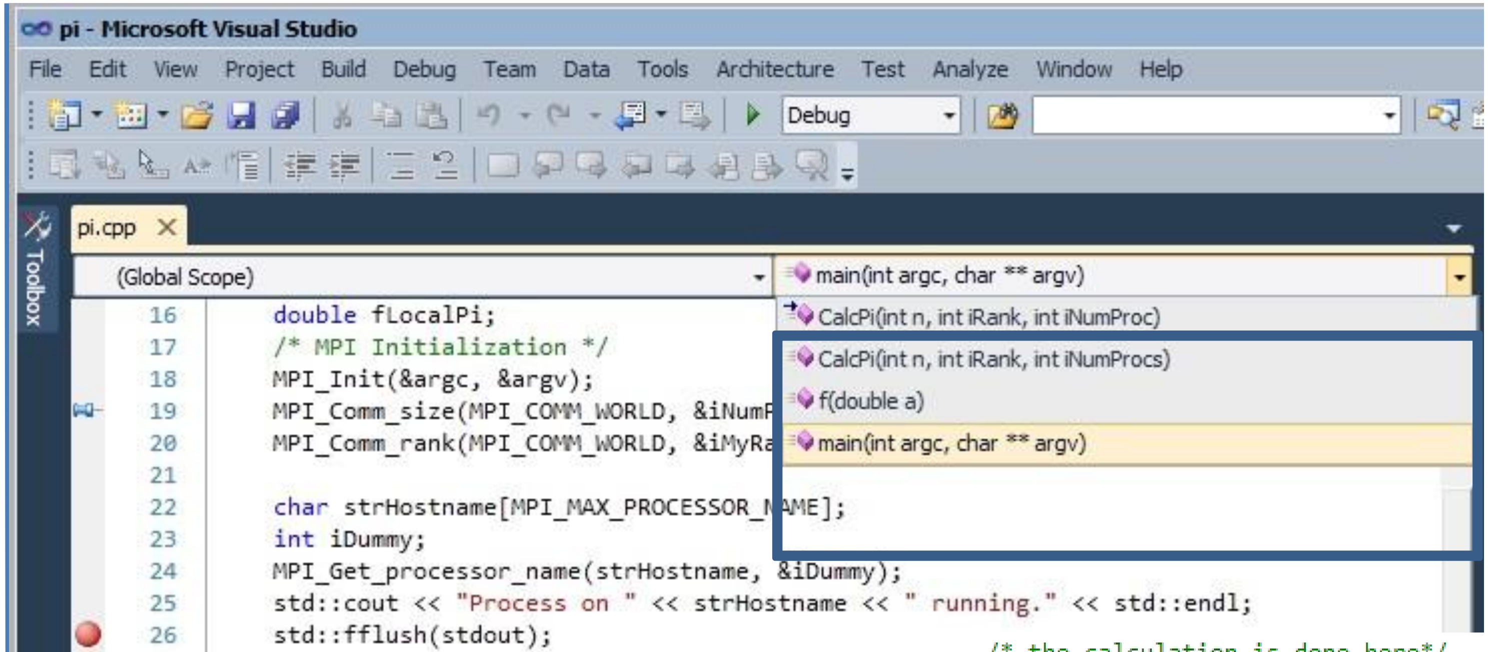


- In many cases, the shortest way to a desired operation can be found by right-clicking on a GUI element and using the context menu.

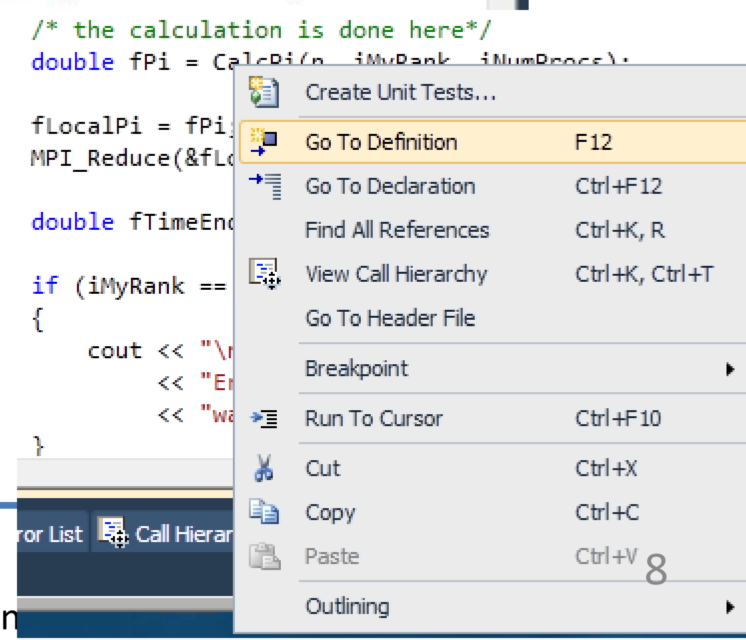
- Adding existing source code items (files) to a project: right-click on the Project (not the Solution !) and *Add → Existing Item…*
- Adding new items: right-click on the Project and *Add → New Item…*

- 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)

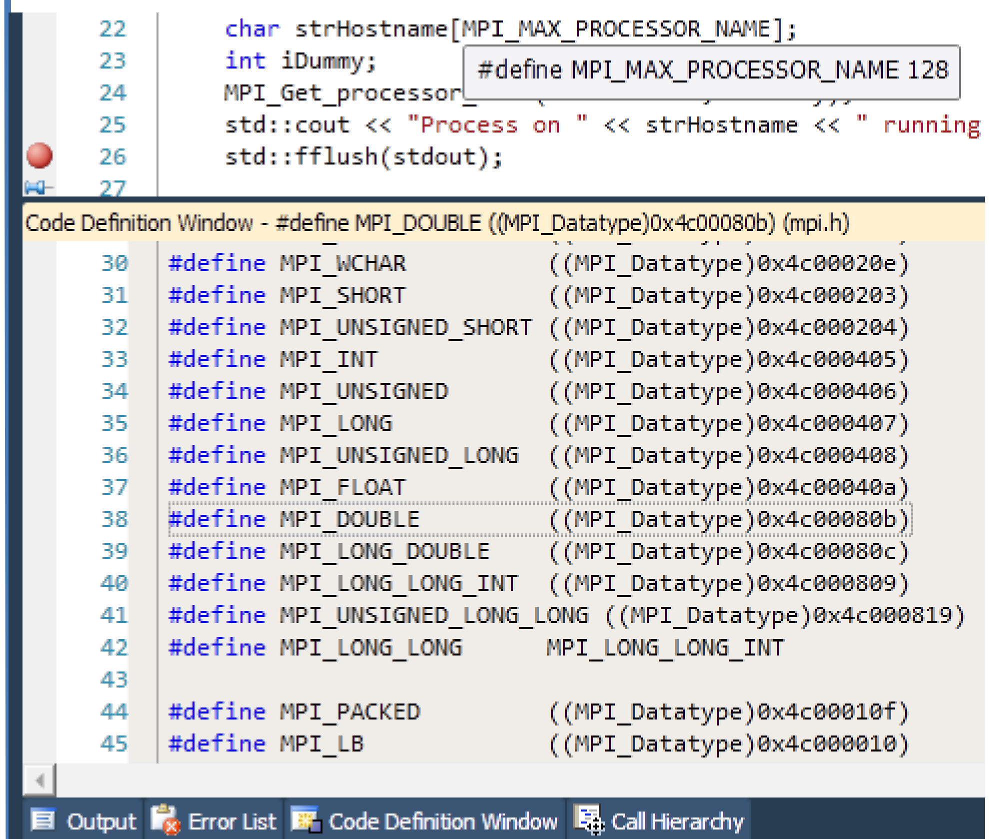


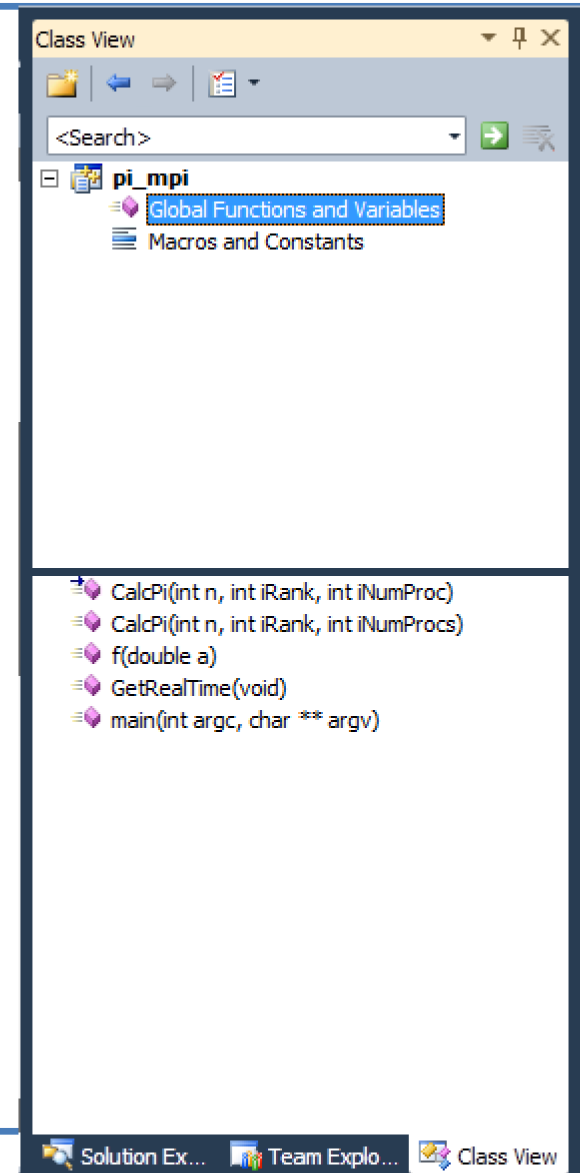
- Selecting a scope + function to navigate right into it‘s implementation.
- Right-clicking a symbol opens up a corresponding context menu:



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*.





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 / VS2010
Debug	Configuration: *Debug*
Release	Configuration: *Release*
x64	Platform: *x64* (64bit for Amd64/Intel64)
Debug	Configuration: *Debug*
Release	Configuration: *Release*

Visual Studio Configurations (1/3)

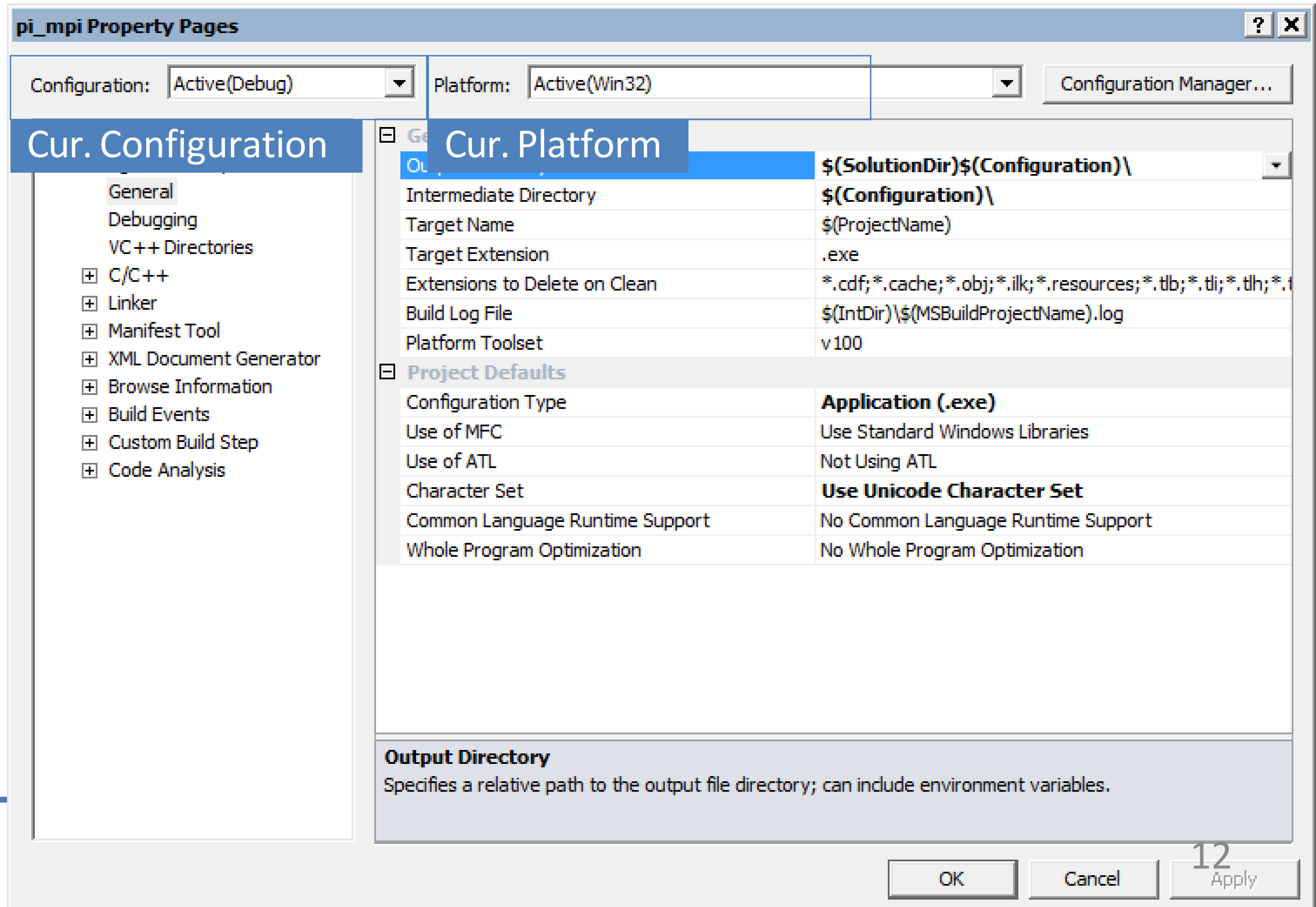
- The set of compiler options is managed in a *Configuration*.
- There are two configurations pre-defined: *Debug* and *Release*.
  - Debug: typical options for debugging, no optimization.
  - Release: debugging still possible, some optimization options.

- 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.

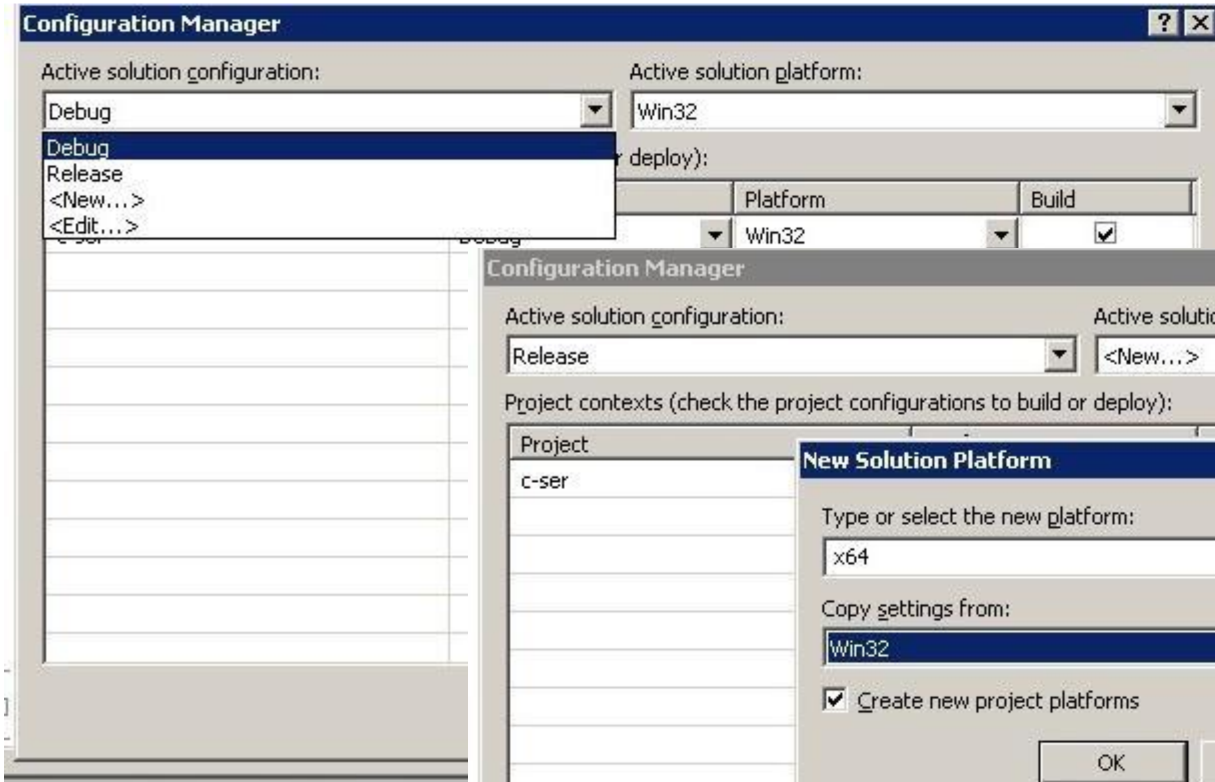
Visual Studio Configurations (2/3)

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

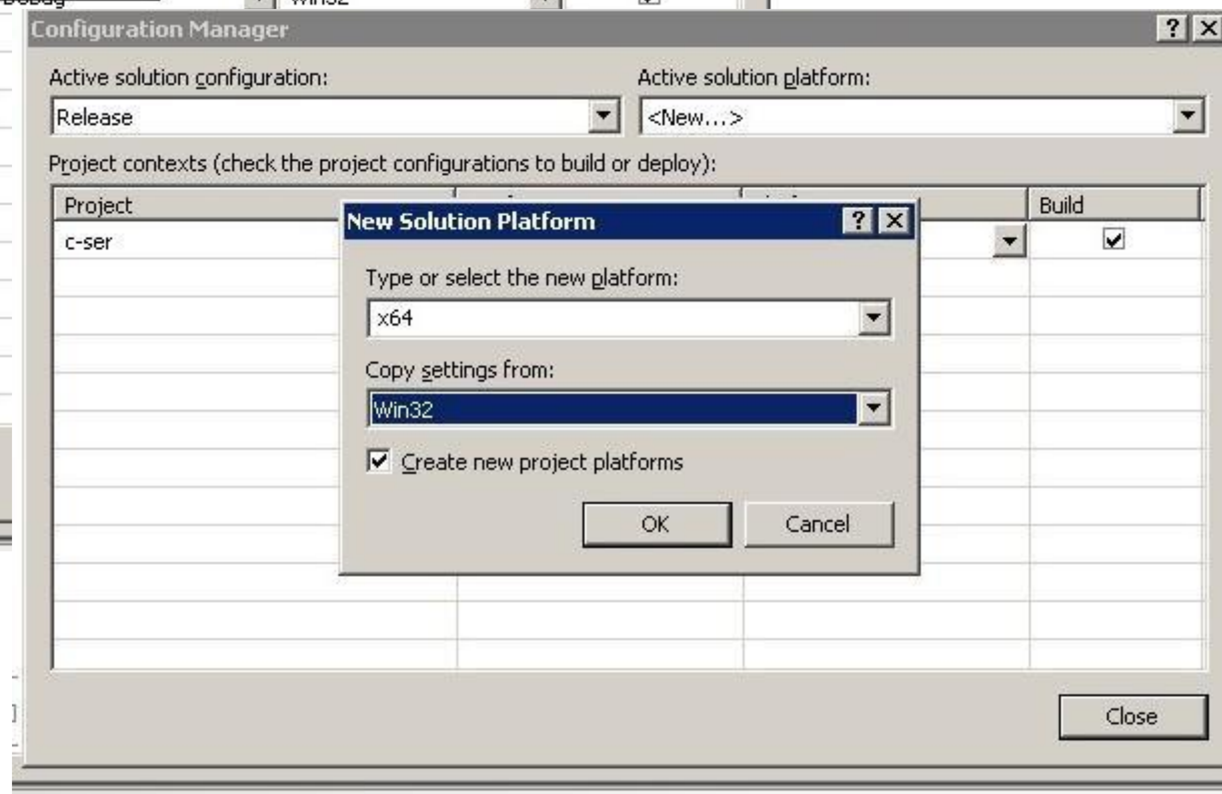


Visual Studio Configurations (3/3)

Build → Configuration Manager:

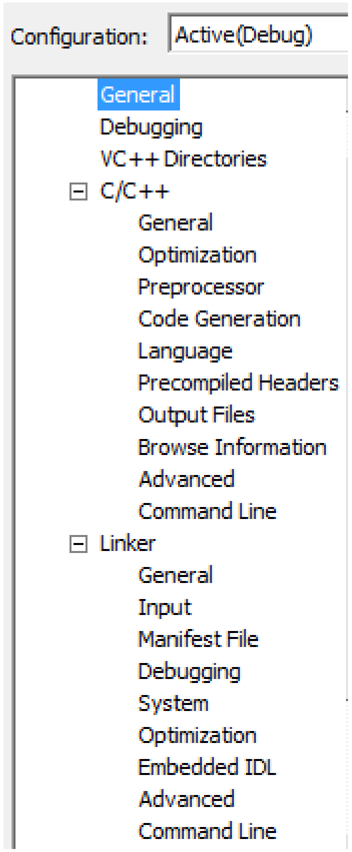


Only Win32 and x64 are supported on our Cluster, not Itanium.



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

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 (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;
```

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 the cluster in Aachen:
      C:\Program Files\Microsoft HPC Pack 2008 SDK\Include
    - I-MPI on the cluster in Aachen:
      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 the cluster in Aachen:
      C:\Program Files\Microsoft HPC Pack 2008 SDK\Lib\[i386|amd64]
    - I-MPI on the cluster in Aachen:
      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 Trace Analyzer & Collector
  - Sometimes a program does not like a specific MPI, so it is always a good thing to have a second one available…

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.

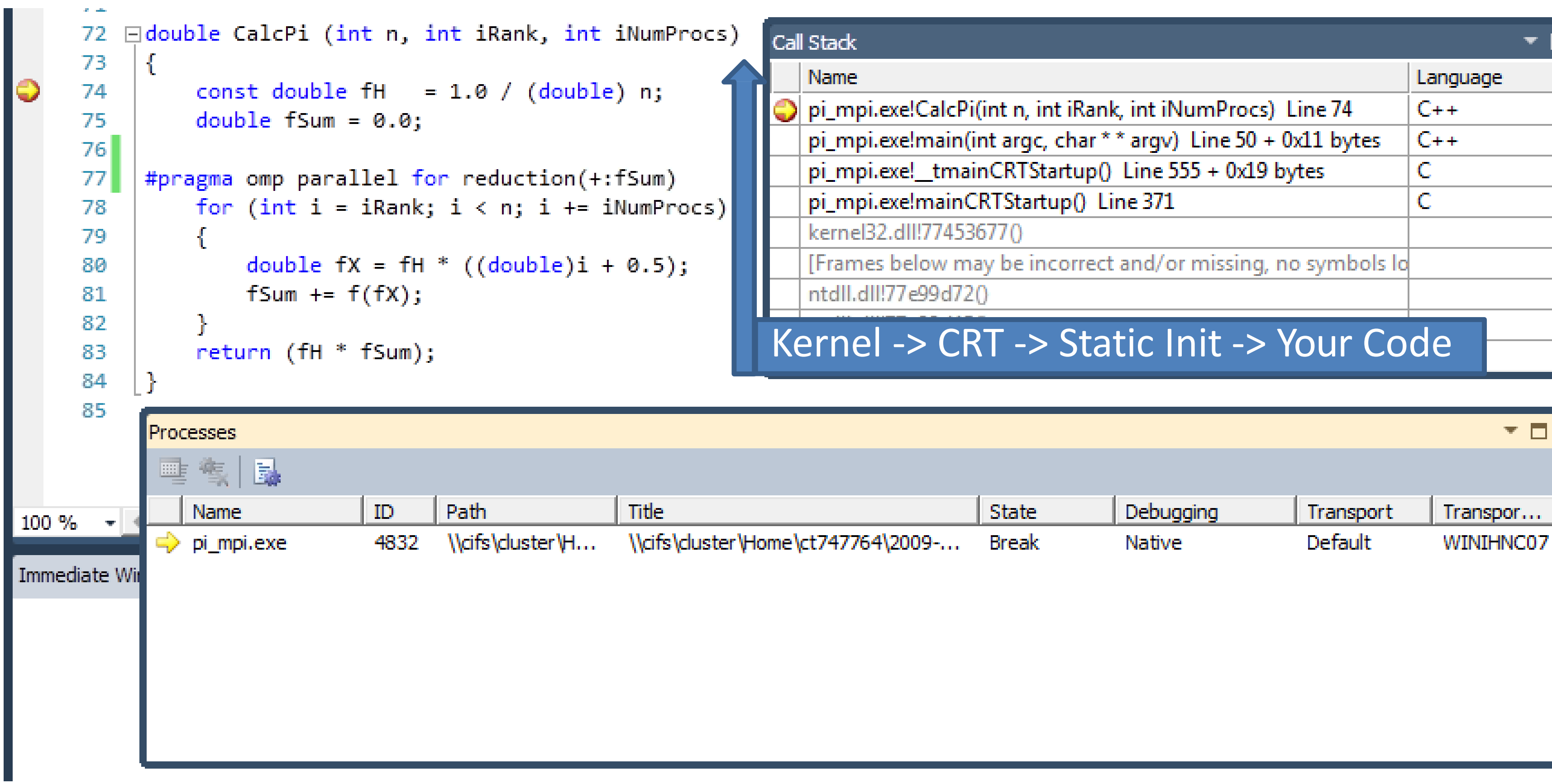
```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &iMyRank);
MPI_Comm_size(MPI_COMM_WORLD, &iNumProcs);
```

iMyRank 0

Local ID in MPI_COMM_WORLD

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 MPI programs (1/6)

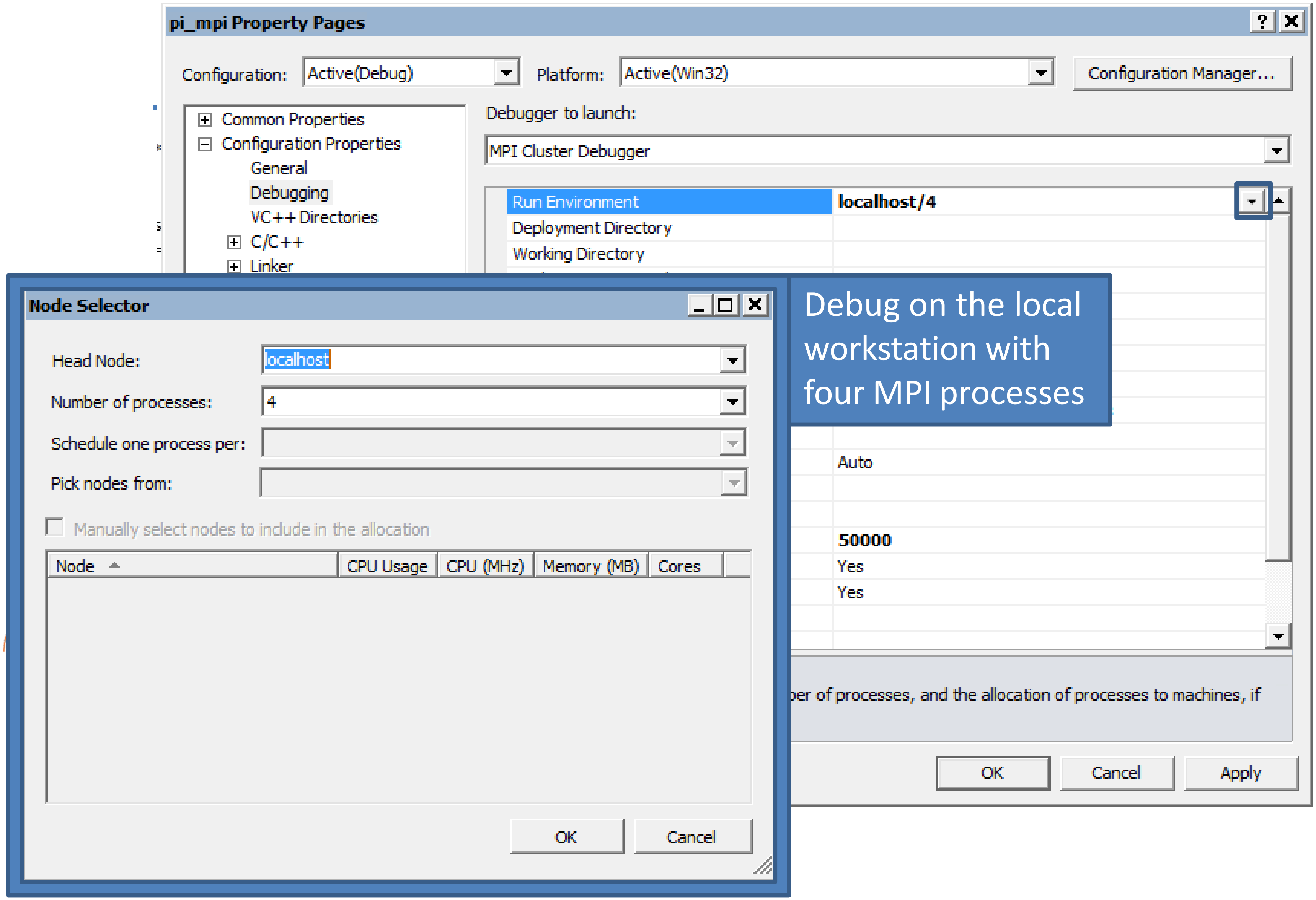
- MS-MPI works best, but you should be able to use I-MPI as well. At least the following instructions work for both.
- Visual Studio supports debugging of MPI programs using the *Cluster Debugger*. As far as I know – or was able to verify – the cluster debugger only works with the Microsoft C/C++ compiler 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)

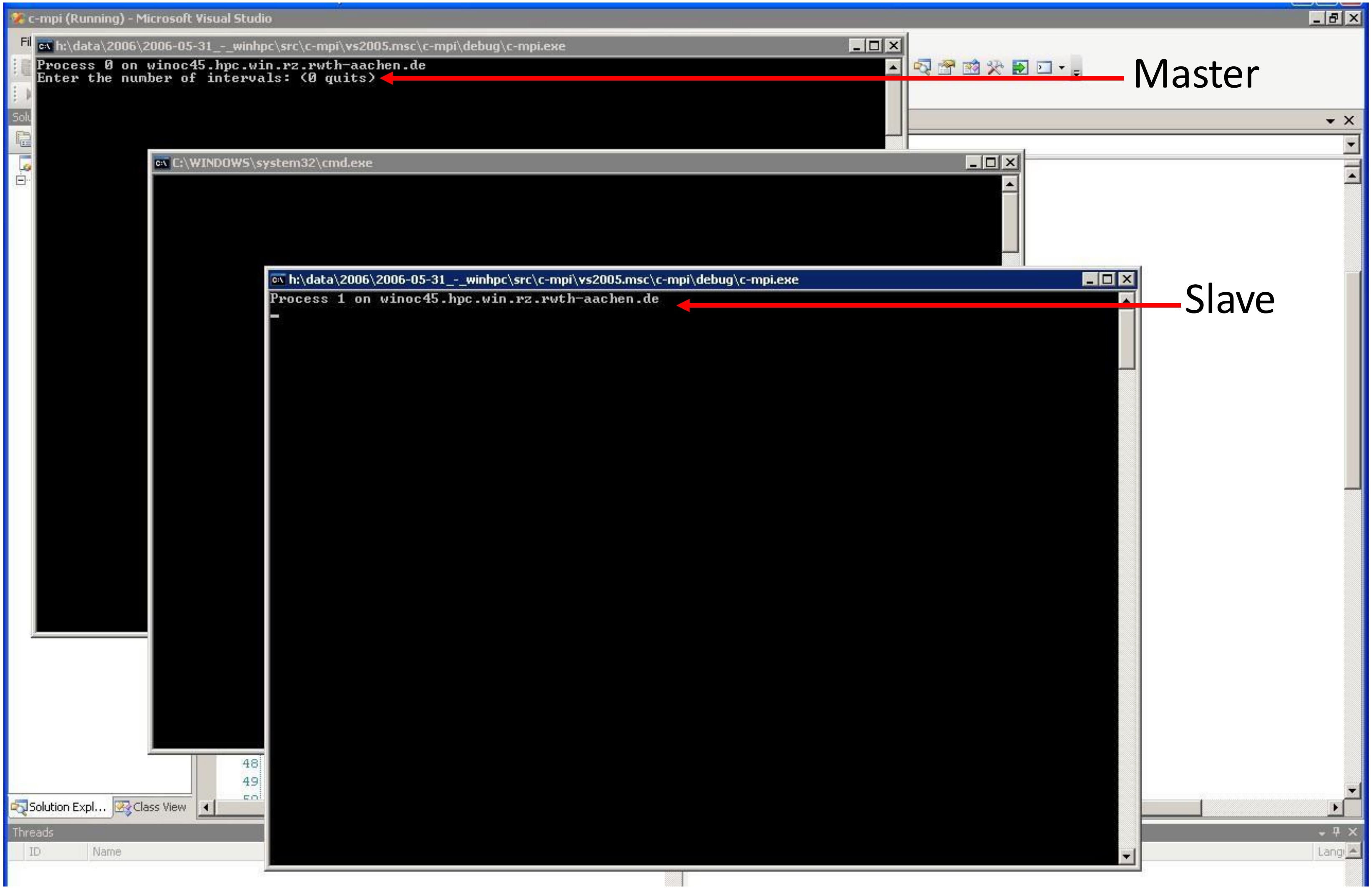
- In order to stop all processes at a breakpoint, please check for the following option: In *Tools* → *Options* → *Debugging* → *General* the checkbox *Break all processes when one process breaks* has to be activated.
- Select the current process using the *Processes* register.

Debugging MPI programs (3/6)

- In VS2010 you can just go with the defaults:

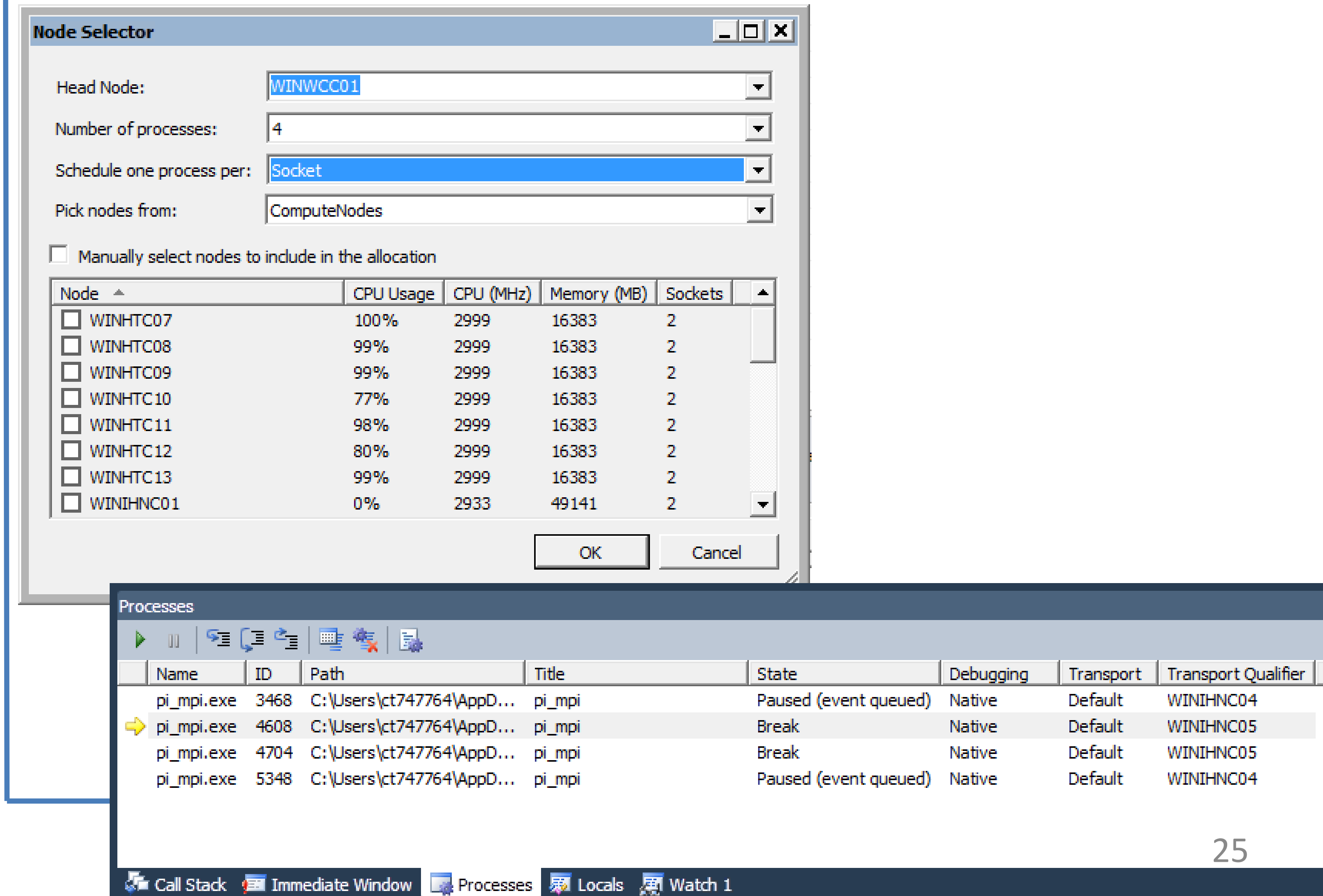


Debugging MPI programs (4/6)



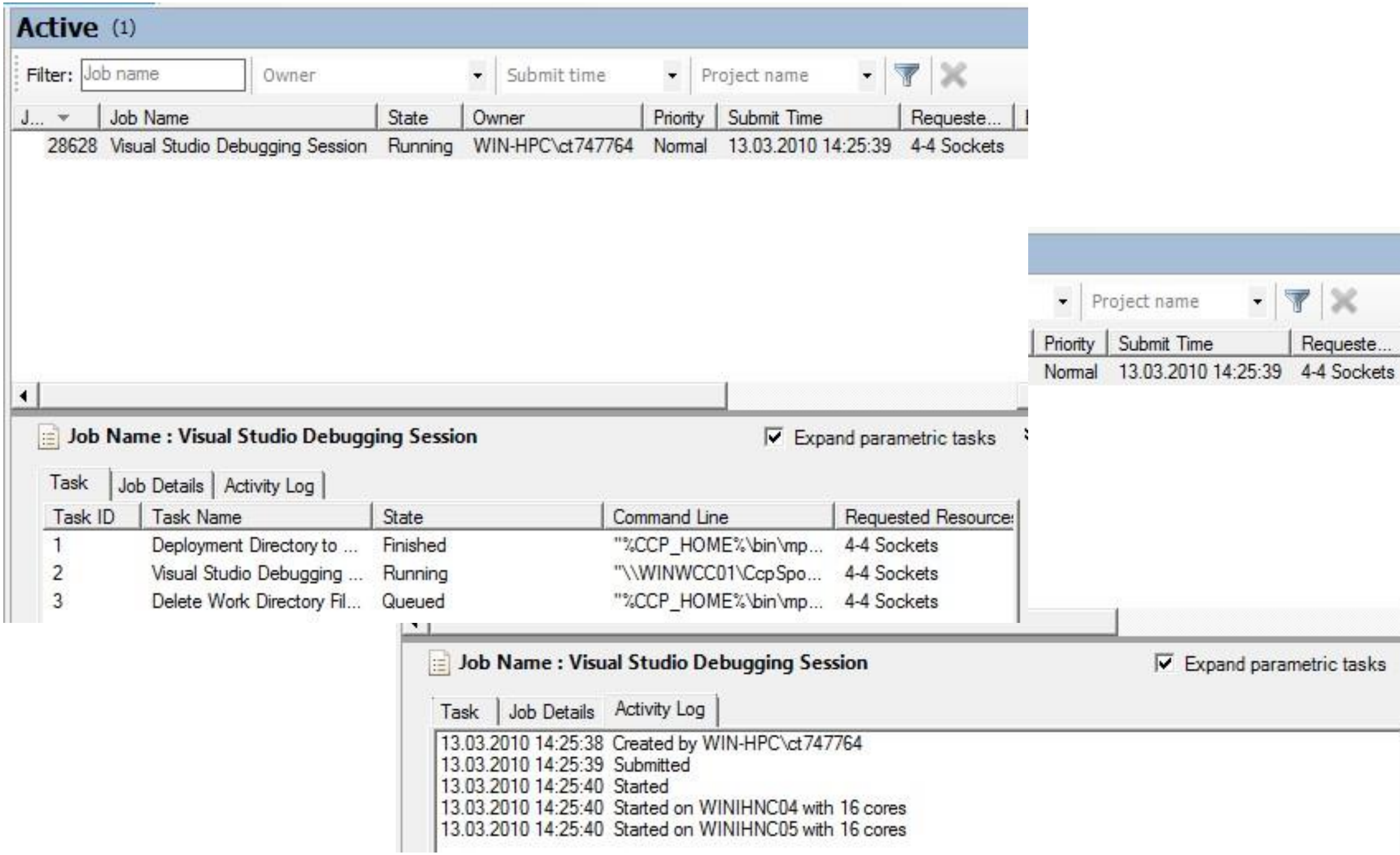
Debugging MPI programs (5/6)

- 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