

Towards the proper “step” command in parallel debuggers

Alexey Kalinov, Konstantin Karganov and Konstantin Khorenko

Institute for System Programming
of the Russian Academy of Sciences

ka@ispras.ru

Parallel versus distributed

Parallel computing means solving of one big problem with essential synchronizations between processes of the parallel program

Distributed computing means solving of one big problem that can be divided or is initially divided into set of independent or weakly synchronized tasks

Tools for parallel computing

Communication libraries: MPI, PVM, ...

Programming languages: HPF, Split-C, mpC, ...

Master-slave paradigm – using parallel programming tools for distributed computing

Synchronization is the main source of errors

In parallel programs logical synchronization errors are the most difficult to detect

It is necessary to step synchronizing processes concurrently

Advanced schemes of stepping is being wanted

Model MPI program

```
1 > > /* 0*/#include "mpi.h"
/* 1*/int main(int ac, char **av) {
/* 2*/ int rank, i;
/* 3*/ MPI_Init(&ac, &av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if (rank == 1) {
/* 6*/     MPI_Status s;
/* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
/* 9*/ if(rank == 0) {
/*10*/     i = 0;
/*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/}
```

Parallel step number, current positions of process with rank 0, rank 1

Model MPI program (step 2)

```
2 > > /* 0*/#include "mpi.h"
/* 1*/int main(int ac, char **av) {
/* 2*/ int rank, i;
/* 3*/ MPI_Init(&ac, &av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if (rank == 1) {
/* 6*/     MPI_Status s;
/* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
/* 9*/ if(rank == 0) {
/*10*/     i = 0;
/*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/}
```

Parallel step number, current positions of process with rank 0, rank 1

Model MPI program (step 3)

```
3 > > /* 0*/#include "mpi.h"
/* 1*/int main(int ac, char **av) {
/* 2*/ int rank, i;
/* 3*/ MPI_Init(&ac, &av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if (rank == 1) {
/* 6*/     MPI_Status s;
/* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
/* 9*/ if(rank == 0) {
/*10*/     i = 0;
/*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/}
```

Parallel step number, current positions of process with rank 0, rank 1

One sequential step – several parallel steps

```
/* 0*/#include "mpi.h"
/* 1*/int m
/* 2*/ int
/* 3*/ MPI_Init(&
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if(rank == 1) {
/* 6*/     MPI_Status s;
4 > /* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
4 > /* 9*/ if(rank == 0) {
/*10*/     i = 0;
/*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return
/*15*/ }
```

Process with rank 1 cannot finish the next step until process with rank 0 call MPI_Send in line 11

It will take two additional steps so that process with rank 0 call MPI_Send in line 11

Parallel step number, current positions of process with rank 0, rank 1

The main schemes of parallel step

Synchronous

Asynchronous

Synchronous scheme

The command is applied to all processes of a group, all processes of the group are expected to complete the step command, the possibility to interrupt the step command execution is available

Advantage:

program state is clear before and after the step

Disadvantage:

it is necessary to perform additional actions to manage the situation “one sequential step – several parallel steps”

Asynchronous scheme

The command is applied to all processes of a group that have already accomplished previous step, the debugger control is returned to user without waiting the command accomplishing

Advantage:

- flexibility in management of the situation
- “one sequential step – several parallel steps”

Disadvantage:

- state of the parallel program is not clear (it is not known either parallel step is accomplished or not)

Trivial parallel causes

The “trivial parallel cause” of process step command incompleteness is a waiting for another process action, for which it is known that it has not been performed and will not be performed during the current step of a parallel program

If the debugger has a model of parallel program execution it can support “smart” synchronous step with handling of “trivial parallel causes”

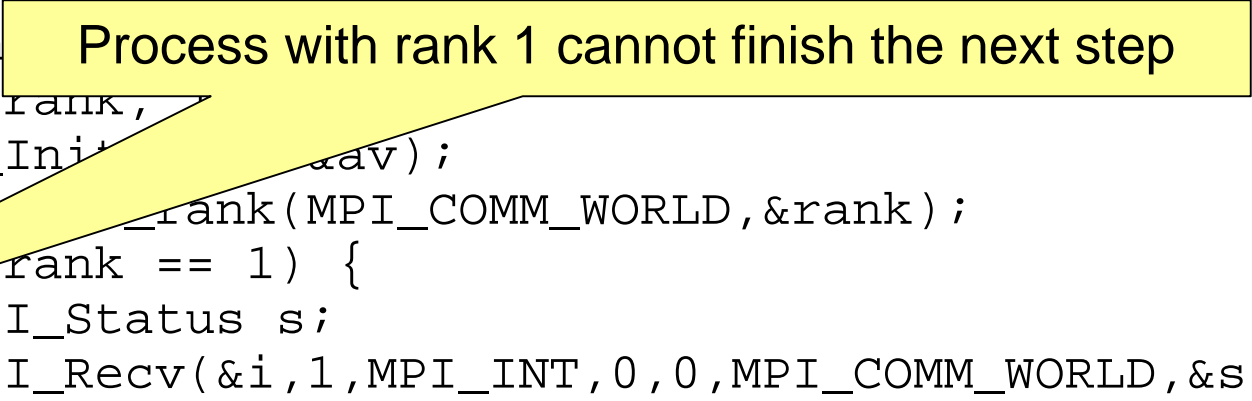
Smart synchronous step

The parallel step of the program is considered to be accomplished if for each process of the group one of the following is true:

- the process has completed its sequential step initiated by the current parallel step and stopped
- the process has completed its sequential step initiated by one of the previous parallel steps and stopped
- the process cannot complete the execution of its sequential step because of “trivial parallel cause”

Smart stepping

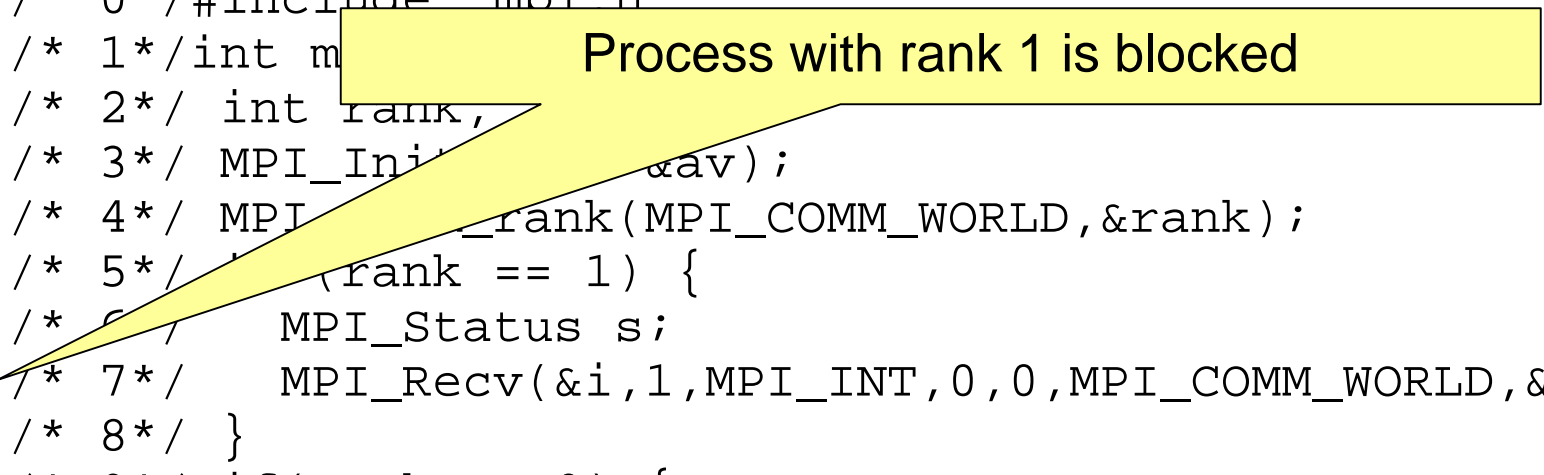
```
/* 0*/#include "mpi.h"
/* 1*/int m
/* 2*/ int rank,
/* 3*/ MPI_Init(&av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if(rank == 1) {
/* 6*/     MPI_Status s;
4 > /* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
4 > /* 9*/ if(rank == 0) {
/*10*/     i = 0;
/*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/}
```



Parallel step number, current positions of process with rank 0, rank 1

Smart stepping (ctd)

```
/* 0*/#include "mpi.h"
/* 1*/int m
/* 2*/ int rank,
/* 3*/ MPI_Init(&av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if(rank == 1) {
/* 6*/     MPI_Status s;
5 > /* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
/* 9*/ if(rank == 0) {
5 > /*10*/     i = 0;
/*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/}
```




Process with rank 1 is blocked

Parallel step number, positions of process with rank 0, rank 1, blocked

Smart stepping (ctd)

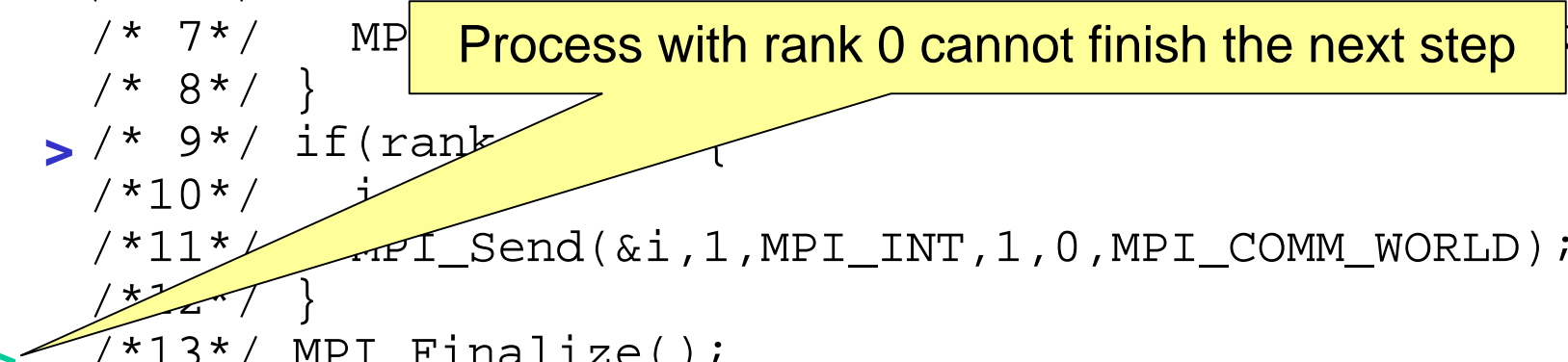
```
/* 0*/#include "mpi.h"
/* 1*/int m
/* 2*/ int rank,
/* 3*/ MPI_Init(&av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if(rank == 1) {
/* 6*/     MPI_Status s;
6 > /* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
/* 9*/ if(rank == 0) {
/*10*/     i = 0;
6 > /*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/}
```



Parallel step number, positions of process with rank 0, rank 1, blocked

Smart stepping (ctd)

```
/* 0*/#include "mpi.h"
/* 1*/int main(int ac, char **av) {
/* 2*/ int rank, i;
/* 3*/ MPI_Init(&ac, &av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD, &rank);
/* 5*/ if (rank == 1) {
/* 6*/     MPI_Status s;
/* 7*/     MP
/* 8*/ }
/* 9*/ if(rank
/*10*/     i
/*11*/     MPI_Send(&i, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/ }
```



Process with rank 0 cannot finish the next step

7

>

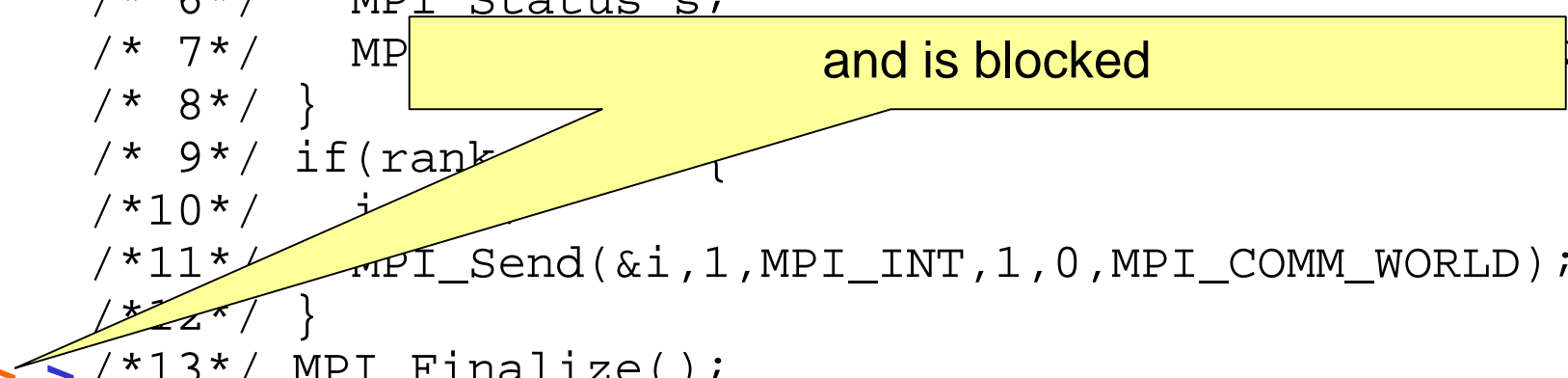
7

>

Parallel step number, positions of process with rank 0, rank 1, blocked

Smart stepping (ctd)

```
/* 0*/#include "mpi.h"
/* 1*/int main(int ac, char **av) {
/* 2*/ int rank, i;
/* 3*/ MPI_Init(&ac, &av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD, &rank);
/* 5*/ if (rank == 1) {
/* 6*/     MPI_Status s;
/* 7*/     MPI_Recv(&i, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &s);
/* 8*/ }
/* 9*/ if (rank == 0) {
/*10*/     i = 1;
/*11*/     MPI_Send(&i, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
/*14*/ return 0;
/*15*/ }
```



and is blocked

8 >

Parallel step number, positions of process with rank 0, rank 1, blocked

Smart stepping (ctd)

```
/* 0*/#include "mpi.h"
/* 1*/int main(int ac, char **av) {
/* 2*/ int rank, i;
/* 3*/ MPI_Init(&ac, &av);
/* 4*/ MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* 5*/ if (rank == 1) {
/* 6*/     MPI_Status s;
/* 7*/     MPI_Recv(&i,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
/* 8*/ }
/* 9*/ if(rank == 0) {
/*10*/     i = 0;
/*11*/     MPI_Send(&i,1,MPI_INT,1,0,MPI_COMM_WORLD);
/*12*/ }
/*13*/ MPI_Finalize();
9 > > /*14*/ return 0;
/*15*/}
```

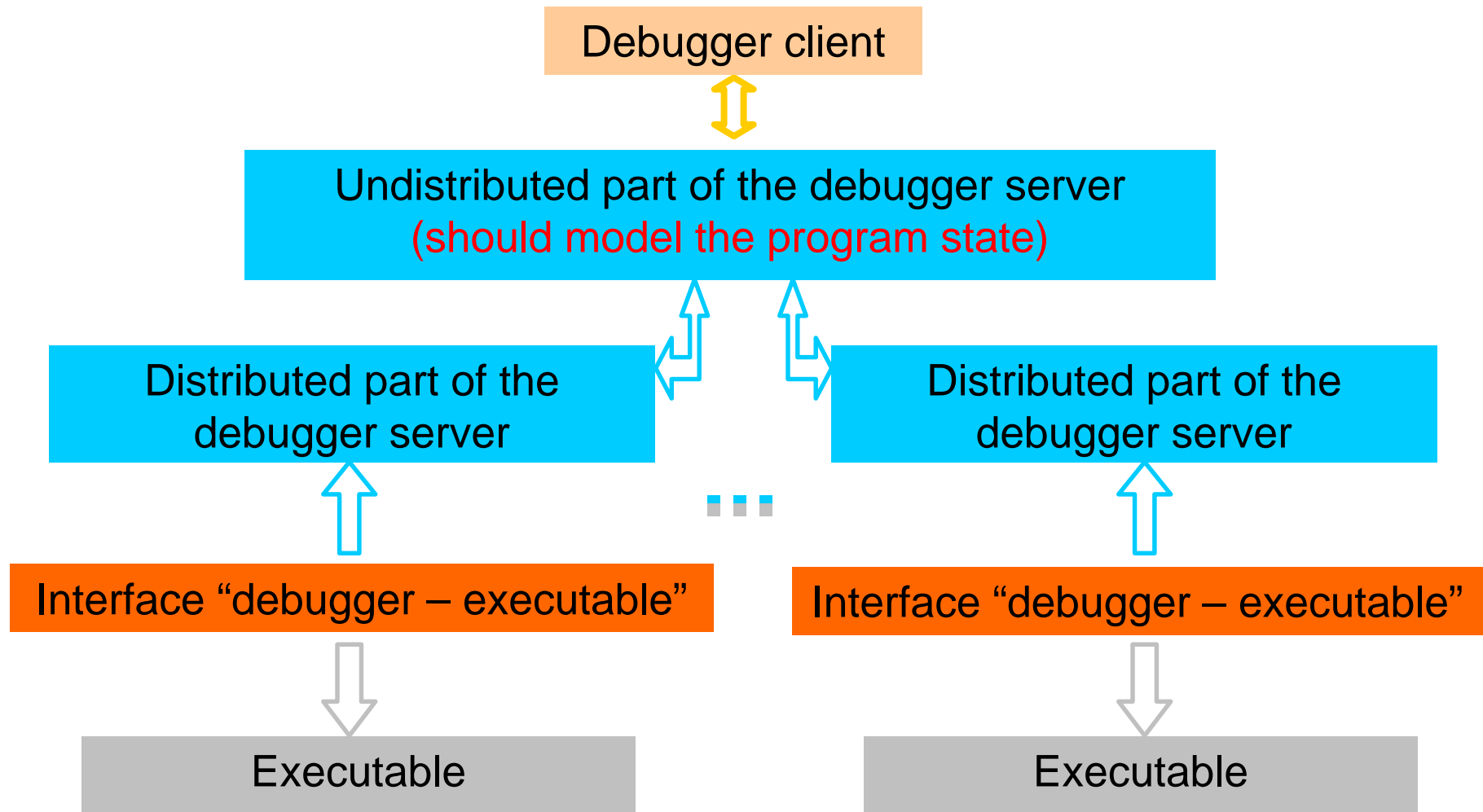
Parallel step number, positions of process with rank 0, rank 1, blocked

Smart stepping advantages

Proposed smart stepping has the following advantages:

- it does not require additional actions for stepping parallel program
- parallel program stepping is as close to the sequential program stepping as possible
- parallel step cannot be finished only due to:
 - sequential cause
 - error in the program

Architecture of parallel debuggers



Problem with MPI_Send

MPI implementation is free to decide whether to buffer the message being sent or not

- in case of message buffering the operation is local and can be completed without the matching receive call
- in other case the operation is not local and cannot be completed before the matching receive call is posted

For smart stepping MPI implementation should provide information about the chosen send mode

mpC Workshop

Integrated development environment for mpC parallel programming language

mpC is a parallel extension of ANSI C targeted to programming heterogeneous networks

mpC relates to C+MPI in parallel programming as C relates to assembly language in sequential programming

mpC Workshop source-level parallel debugger supports smart stepping

mpC analogue of model MPI program

The screenshot displays the mpC Workshop IDE with the file `ex56.mpc` open. The left sidebar shows a project tree with `ex56`, `Target_Debug`, `ex56.exe_(det)`, `Source Files`, `ex56.mpc`, and `Header Files`. The main editor shows the following code:

```
#include "mpc.h"

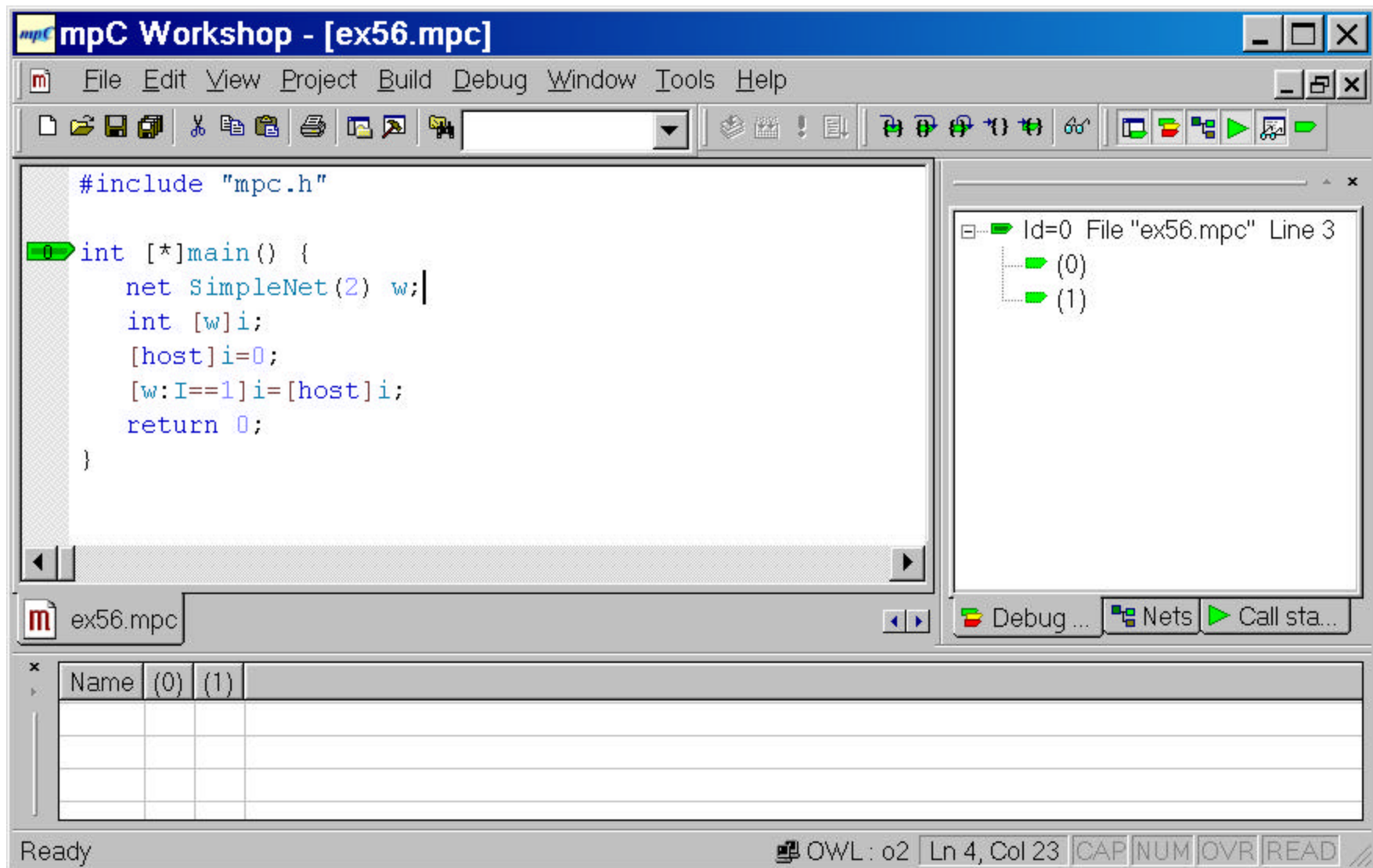
int [*]main() {
    net SimpleNet(2) w;
    int [w]i;
    [host]i=0;
    [w:I==1]i=[host]i;
    return 0;
}
```

Annotations explain the code:

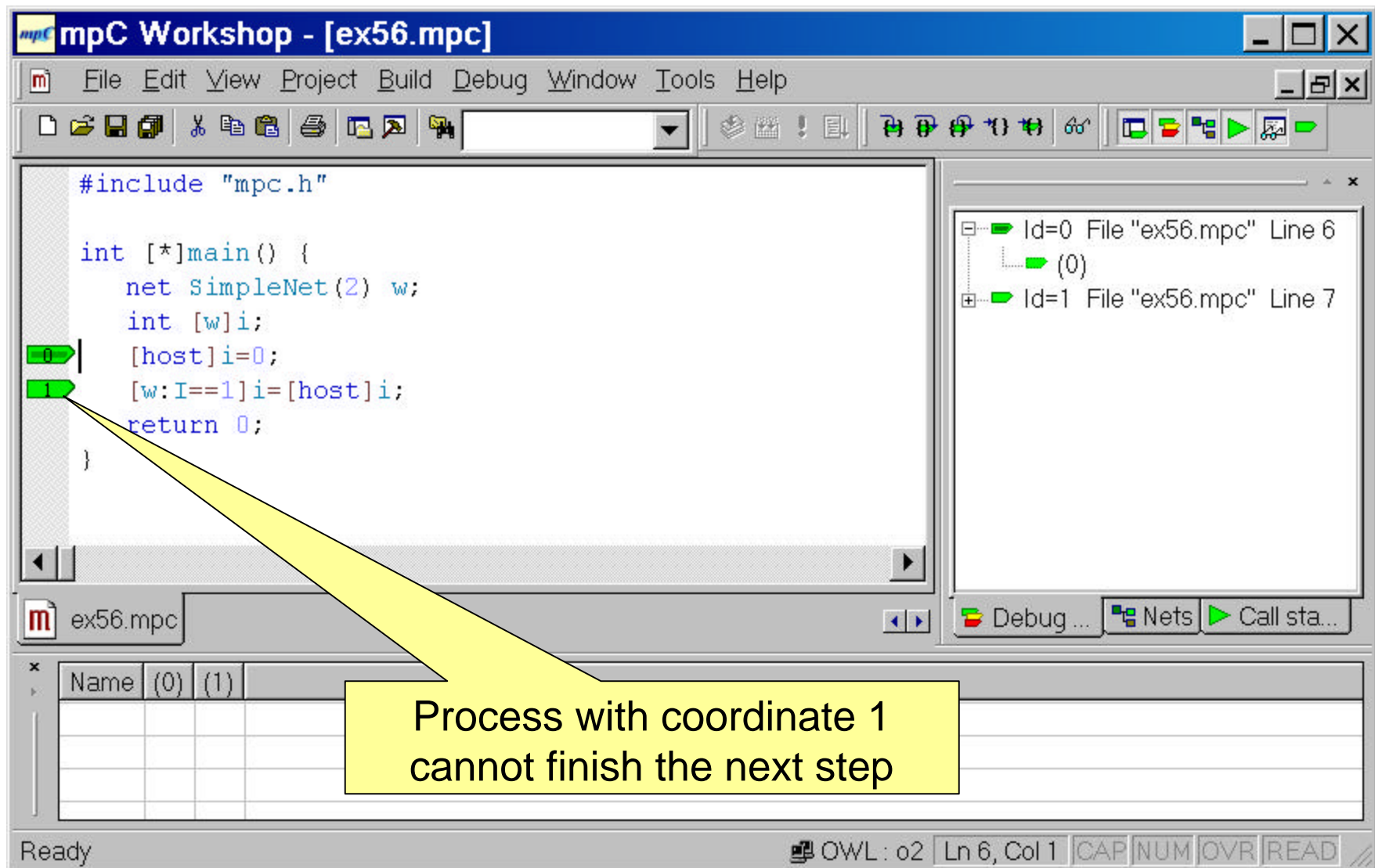
- Definition of network of 2 nodes. host has coordinate 0 in w.** (points to `net SimpleNet(2) w;`)
- Variable `i` is distributed over `w`.** (points to `int [w]i;`)
- 0 is assigned to the component of variable `i` on host.** (points to `[host]i=0;`)
- The value of the component of variable `i` on host is assigned to the component of variable `i` on the node with coordinate 1.** (points to `[w:I==1]i=[host]i;`)

The bottom status bar shows the output: `C:\mpC\Server\OwnMPI\L`, `Errors: 0`, and `Operation successful.` The status bar also includes tabs for `Build`, `Debug`, and `Find in Files`, and a status indicator `Ready`.

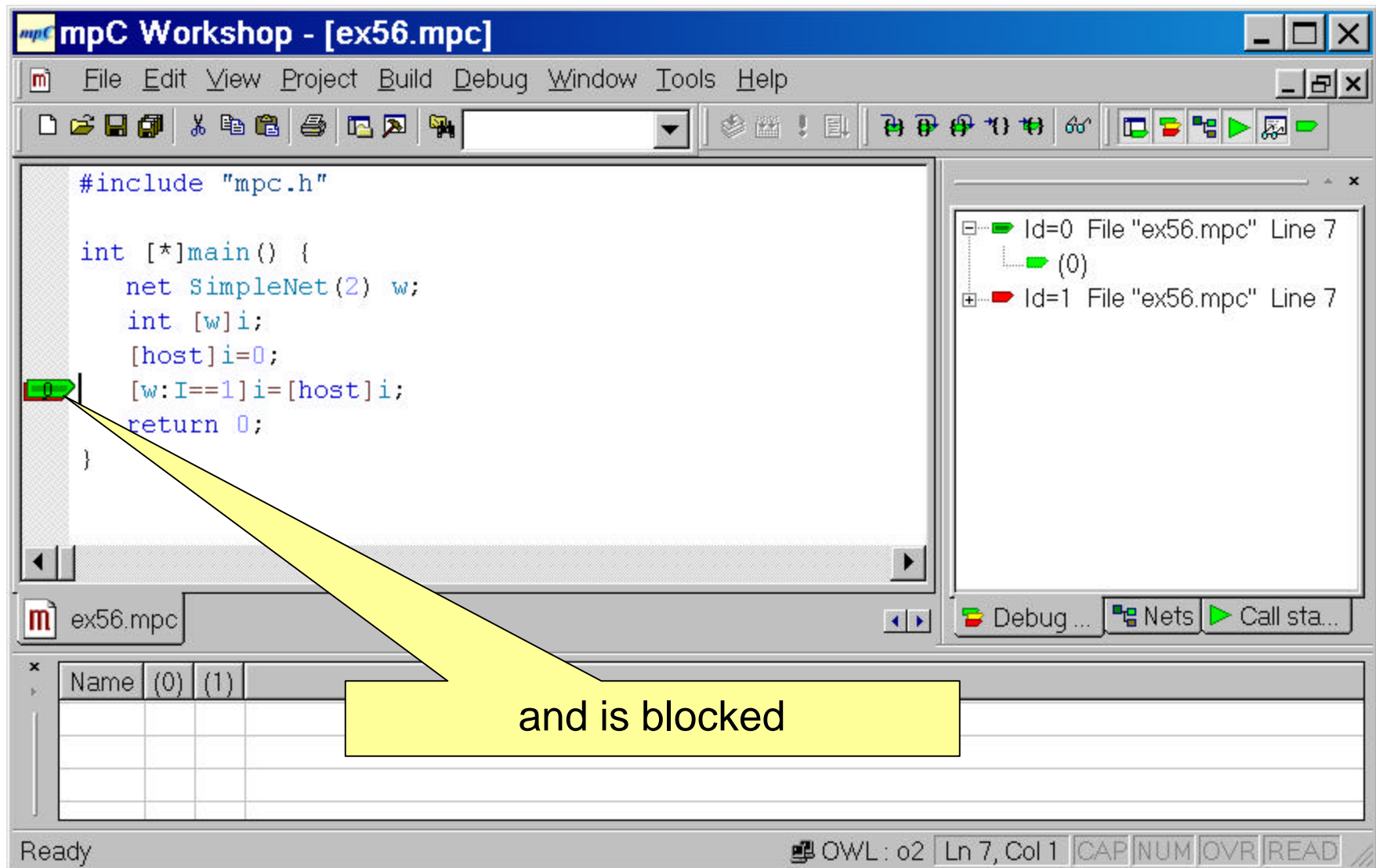
mpC Workshop debug session



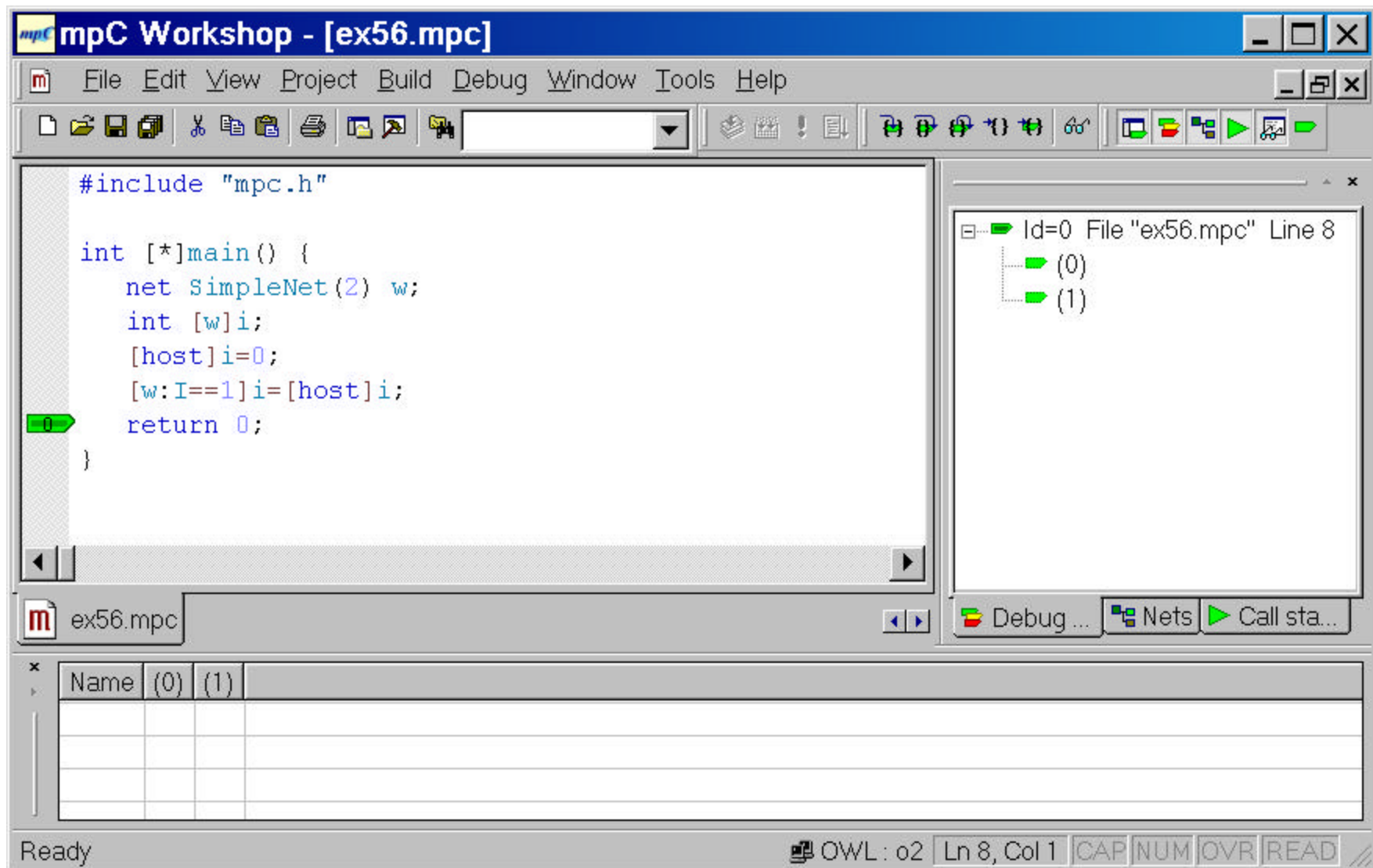
mpC Workshop debug session (ctd)



mpC Workshop debug session (ctd)



mpC Workshop debug session (ctd)



Conclusion

Smart stepping allows user to avoid the routine work and to make the debugging of a parallel program as close as possible to the debugging of a sequential program

Implementation of smart stepping requires the support from communication library or parallel programming language used