

Debugging

Inhalt

1. Herkunft und Einführung Debugging
2. Wie funktioniert Debugging?
 1. Dynamisches Debugging
 2. Statisches Debugging vs. Dynamisches Debugging
3. Debugging von parallelen Programmen mit DDT
4. Zusammenfassung & Fazit

Herkunft und Einführung Debugging

Was ist ein „Bug“?

“Ein Softwarebug ist ein **Programmfehler**, welcher erst **bei Ausführung des Programms** auftritt. Das Programm zeigt ab dem Auftreten ein fehlerhaftes Verhalten im weiteren Verlauf oder es wird ganz abgebrochen.” [1]

Definition: Debugging

“Debugging bezeichnet den Prozess des **Finden** und **Beheben** von **Fehlern oder Problemen innerhalb eines Computerprogramms**, welche eine Computersoftware oder ein System daran hindern, richtig zu funktionieren.” [2]

Wieso „macht“ man Debugging?

- Finden und Beheben von Fehlern
- Fehlersuche erleichtern
- Softwarequalität verbessern bzw. sicherstellen
- Programm oder Programmablauf nachvollziehen

Wie funktioniert Debugging?

Debuggingprozess

1) Problem reproduzieren

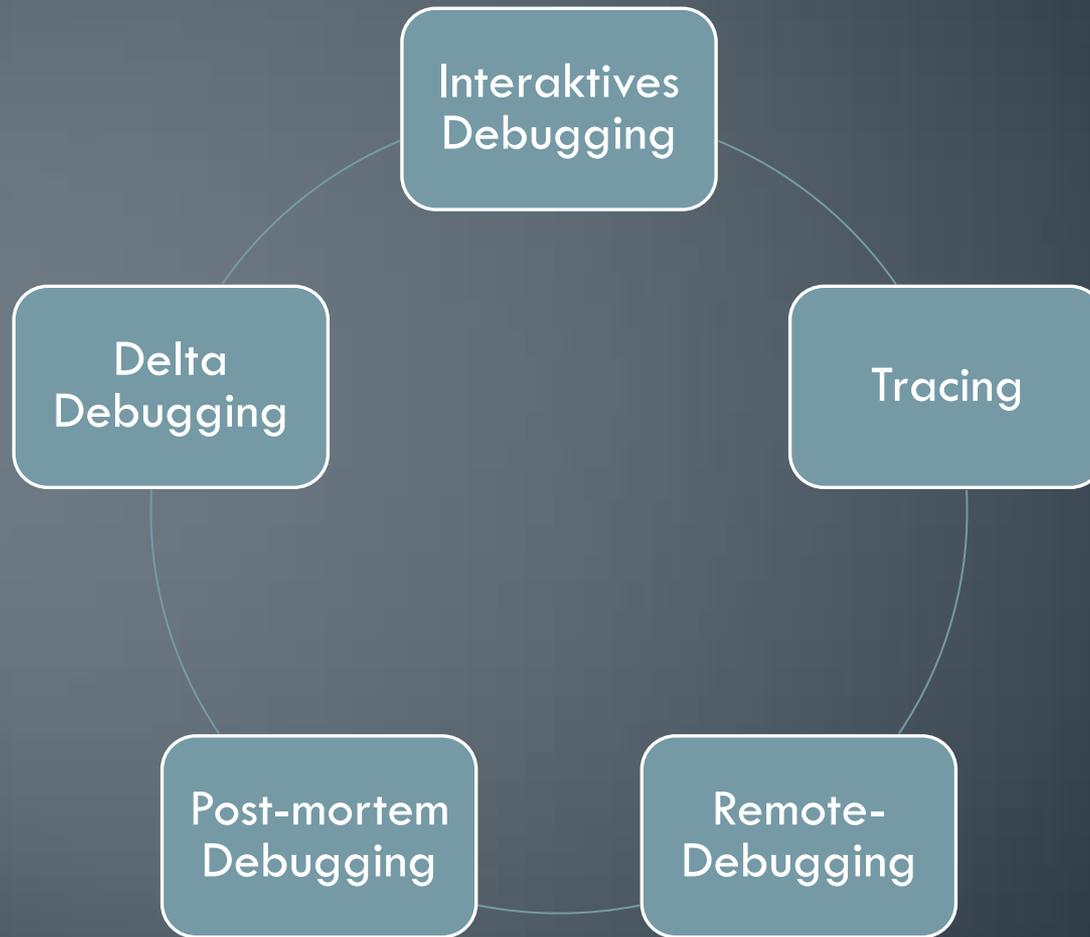
2) Programm evtl. vereinfachen

3) Programmstatus untersuchen

4) Ursprung des Problems suchen

5) Fehler beheben

Debuggingtechniken



Dynamisches Debugging

“**Debugging** of a computer system while it is executing. Optionally a **dynamic** debugger provides a console to interact with the system” [3]

Dynamisches Debugging (2)

- Statisches Debugging vs. dynamisches Debugging
- Häufige Fehlerquellen
- Werkzeuge des dynamischen Debugging
 - a) Einzelne Schritte
 - b) Inspektion des Zustands des Programms
 - c) Haltepunkte

Statisches Debugging vs. Dynamisches Debugging

Statisches Debugging

- Automatisiert Fehler finden
- Überprüfung auf Konformität von Programmiertechniken
- Laufzeitfehler zu finden ist mit sehr großem Aufwand verbunden

Dynamisches Debugging

- Erleichtert finden und beheben von Laufzeitfehlern
- GUI erleichtert oft Nutzung
- Debugging von parallelen Programmen

Debugging von parallelen Programmen

Debugging von parallelen Programmen ist:

- Vor allem schwierig, weil:
 - 1) Zu viel „auf einmal“ passiert
 - 2) Viele Debugger nicht für paralleles Debugging gemacht wurden
- Wichtig, um die genauen Vorgänge im Programm nachvollziehen zu können

Distributed Debugging Tool (DDT)

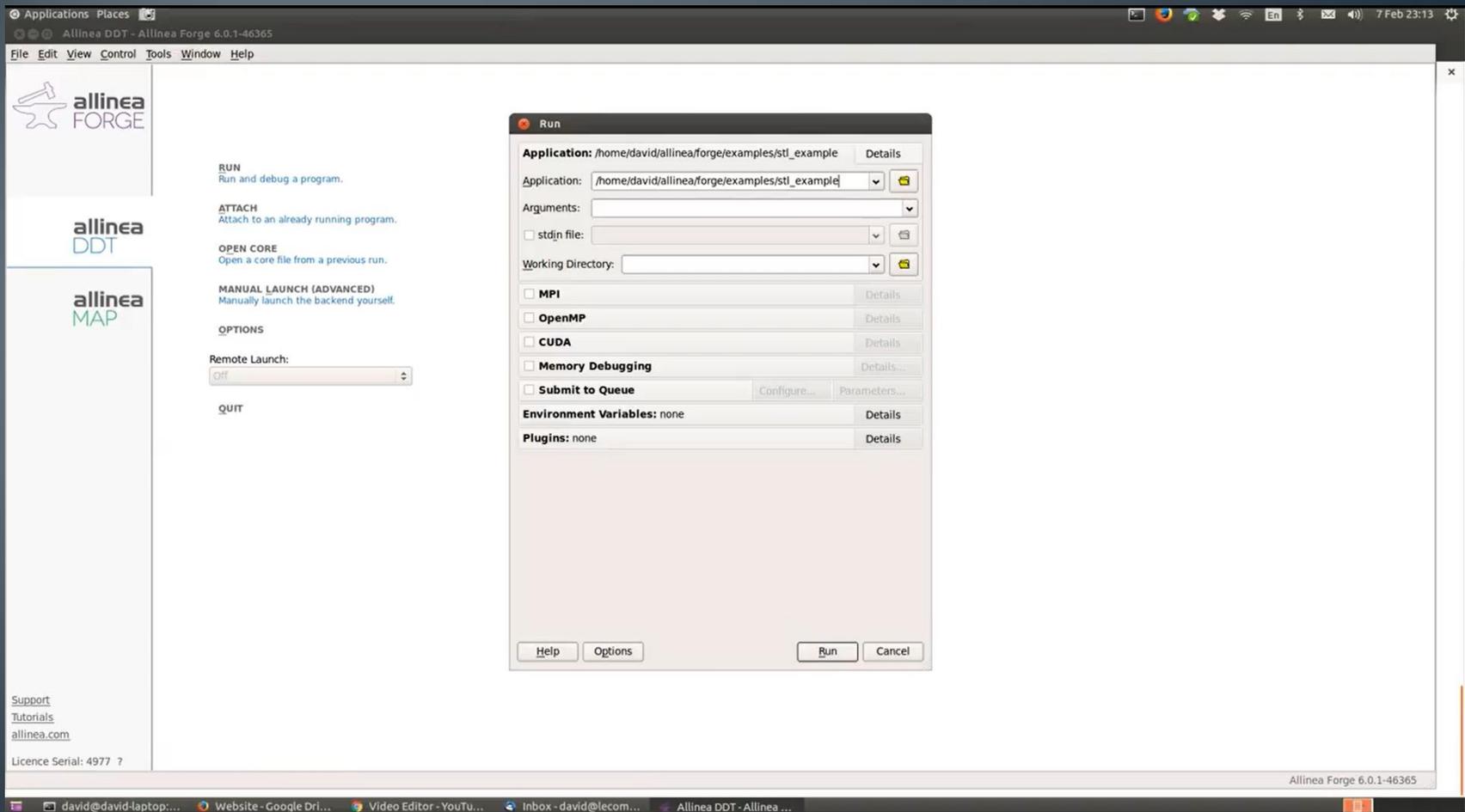
- In der Lage sehr viele verschiedene Anwendungen, darunter vor Allem parallele Programme, zu debuggen
- Besitzt eine Vielzahl an verschiedenen Features
- Bietet grafische Oberfläche

DDT – Technische Details

- Kann Anwendungen mit über 200.000 gleichzeitigen Prozessen debuggen
- Unterstützt alle Programmiersprachen, die im Gebiet des Hochleistungsrechnen vorkommen
- Extrem schnelle Performanz

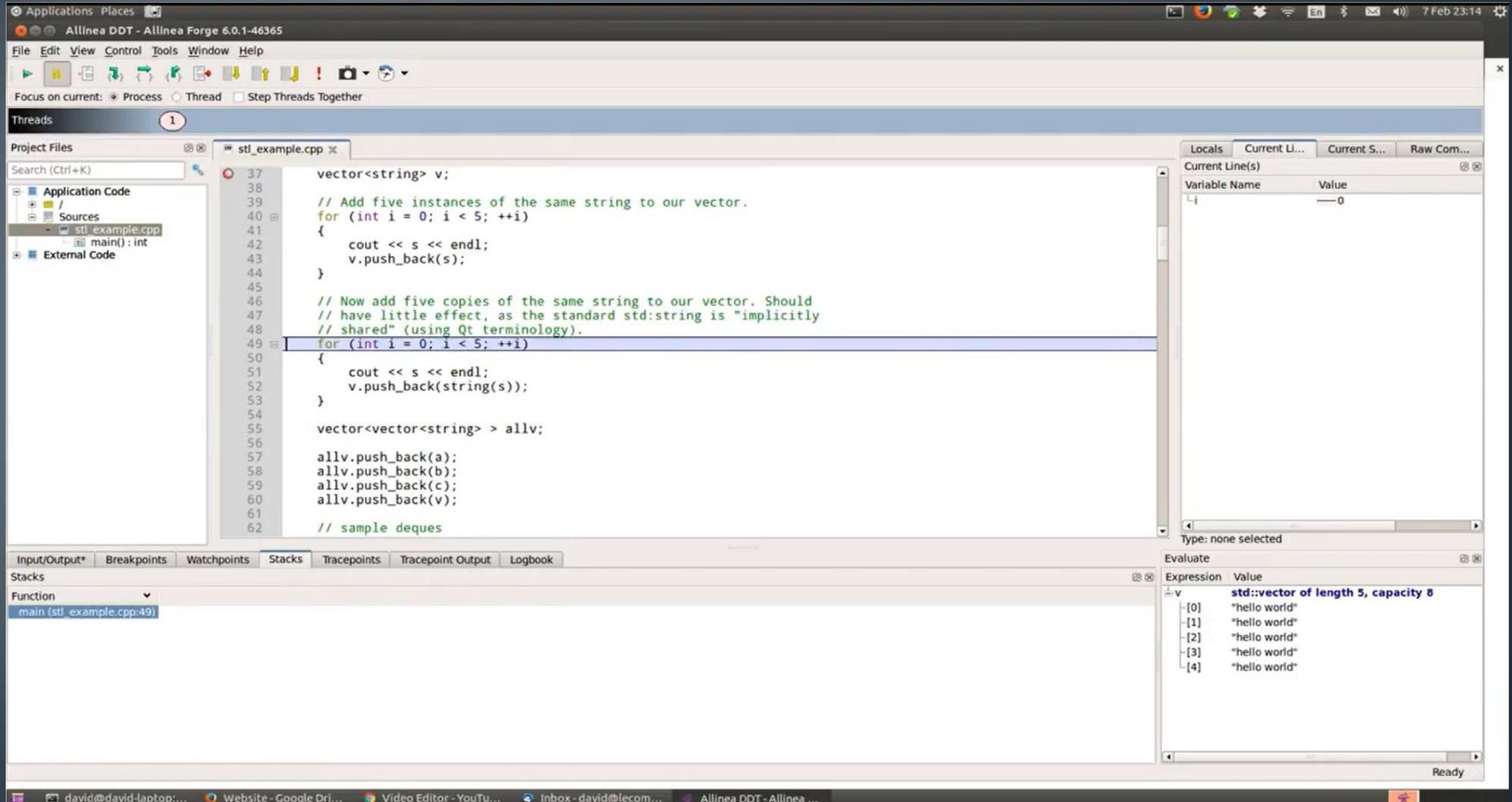
Vorführung – Wie funktioniert DDT?

Einzelprozessanwendungen



UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Einzelprozessanwendungen (2)



UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Paralleles Debugging für MPI

Beispiel 1:

The screenshot shows the Allinea Distributed Debugging Tool v2.6 interface. The main window displays the source code of 'helloworld.c' with a breakpoint set at line 48. The 'Locals' panel on the right shows the current state of variables: argc=1, argv=0x7fffeb5bc8, message=0, numprocs=4, rank=0, status=10, tag=10, and x=32767. The 'Breakpoints' panel at the bottom shows a breakpoint for 'helloworld.c' at line 48. The status bar at the bottom right indicates '3 processes busy'.

```
36 MPI_Recv(&message, /*buffer for message */
37         MPI_INT, /*MAX count to recv */
38         MPI_INT, /*type to recv */
39         0, /*recv from 0 only */
40         tag, /*tag of message */
41         MPI_COMM_WORLD, /*communicator to use */
42         &status); /*status object */
43 printf("Hello from process %d!\n",rank);
44 }
45 else{
46 /* rank 0 ONLY executes this */
47 printf("MPI_COMM_WORLD is %d processes big!\n", numprocs);
48 int x;
49 for(x=1; x<numprocs; x++){
50 MPI_Send(&x, /*send x to process x */
51         1, /*number to send */
52         MPI_INT, /*type to send */
53         x, /*rank to send to */
54         tag, /*tag for message */
55         MPI_COMM_WORLD); /*communicator to use */
56 }
57 } /* end else */
58
59
60 /* always call at end */
61 MPI_Finalize();
62
63 return 0;
```

Variable Name	Value
argc	1
argv	0x7fffeb5bc8
message	0
numprocs	4
rank	0
status	10
tag	10
x	32767

Processes	Threads	File	Line	Function	Condition	Full path
<input checked="" type="checkbox"/>	All	all	helloworld.c	48		/home/brockp/cac-mpi-code/helloworld.c

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Paralleles Debugging für MPI

Beispiel 1:

The screenshot shows the Allinea Distributed Debugging Tool v2.6 interface. The main window displays the source code of 'helloworld.c' with a breakpoint set at line 48. The 'Locals' panel on the right shows the current state of variables: argc=1, argv=0x7fffeb5bc8, message=0, numprocs=4, rank=0, status=10, tag=10, and x=32767. The 'Breakpoints' panel at the bottom shows a breakpoint for 'All' at line 48 of 'helloworld.c'. The status bar at the bottom right indicates '3 processes busy'.

```
36 MPI_Recv(&message, /*buffer for message */
37         MPI_INT, /*MAX count to recv */
38         MPI_INT, /*type to recv */
39         0, /*recv from 0 only */
40         tag, /*tag of message */
41         MPI_COMM_WORLD, /*communicator to use */
42         &status); /*status object */
43 printf("Hello from process %d!\n",rank);
44 }
45 else{
46 /* rank 0 ONLY executes this */
47 printf("MPI_COMM_WORLD is %d processes big!\n", numprocs);
48 int x;
49 for(x=1; x<<numprocs; x++){
50 MPI_Send(&x, /*send x to process x */
51         1, /*number to send */
52         MPI_INT, /*type to send */
53         x, /*rank to send to */
54         tag, /*tag for message */
55         MPI_COMM_WORLD); /*communicator to use */
56 }
57 } /* end else */
58
59
60 /* always call at end */
61 MPI_Finalize();
62
63 return 0;
```

Variable Name	Value
argc	1
argv	0x7fffeb5bc8
message	0
numprocs	4
rank	0
status	10
tag	10
x	32767

Processes	Threads	File	Line	Function	Condition	Full path
All	all	helloworld.c	48			/home/brockp/cac-mpi-code/helloworld.c

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Paralleles Debugging für MPI

Beispiel 1:

The screenshot shows the Allinea Distributed Debugging Tool v2.6 interface. The main window displays the source code of a C program named 'helloworld.c'. The code is as follows:

```
36 MPI_Recv(&message, /*buffer for message */
37         1, /*MAX count to recv */
38         MPI_INT, /*type to recv */
39         0, /*recv from 0 only */
40         tag, /*tag of message */
41         MPI_COMM_WORLD, /*communicator to use */
42         &status); /*status object */
43 printf("Hello from process %d!\n",rank);
44 }
45 else{
46     /* rank 0 ONLY executes this */
47     printf("MPI_COMM_WORLD is %d processes big!\n", numprocs);
48     int x;
49     for(x=1; x<numprocs; x++){
50         MPI_Send(&x, /*send x to process x */
51                 1, /*number to send */
52                 MPI_INT, /*type to send */
53                 x, /*rank to send to */
54                 tag, /*tag for message */
55                 MPI_COMM_WORLD); /*communicator to use */
56     }
57 } /* end else */
58
59
60 /* always call at end */
61 MPI_Finalize();
62
63 return 0;
```

The 'Locals' panel on the right shows the current state of variables:

Variable Name	Value
argc	1
argv	0x7fffeb5bc8
message	0
numprocs	4
rank	0
status	10
tag	10
x	32767

The 'Breakpoints' panel at the bottom shows a breakpoint for 'All' at line 48 of 'helloworld.c'.

Processes	Threads	File	Line	Function	Condition	Full path
✓ All	all	helloworld.c	48			/home/brockp/cac-mpi-code/helloworld.c

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Paralleles Debugging für MPI

Beispiel 1:

The screenshot shows the Allinea Distributed Debugging Tool (DDT) v2.6 interface. The main window displays the source code of a C program named 'helloworld.c'. The code is as follows:

```
36 MPI_Recv(&message, /*buffer for message */
37         MPI_INT, /*MAX count to recv */
38         MPI_INT, /*type to recv */
39         0, /*recv from 0 only */
40         tag, /*tag of message */
41         MPI_COMM_WORLD, /*communicator to use */
42         &status); /*status object */
43 printf("Hello from process %d!\n",rank);
44 }
45 else{
46 /* rank 0 ONLY executes this */
47 printf("MPI_COMM_WORLD is %d processes big!\n", numprocs);
48 int x;
49 for(x=1; x<numprocs; x++){
50 MPI_Send(&x, /*send x to process x */
51         1, /*number to send */
52         MPI_INT, /*type to send */
53         x, /*rank to send to */
54         tag, /*tag for message */
55         MPI_COMM_WORLD); /*communicator to use */
56 }
57 } /* end else */
58
59
60 /* always call at end */
61 MPI_Finalize();
62
63 return 0;
```

The 'Input/Output' window shows the following output:

```
Other: MPI_COMM_WORLD is 4 processes big!
Other: Hello from process 2!
Other: Hello from process 1!
Other: Hello from process 3!
```

The 'Current Stack' window shows 'Process 0 has finished.'

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Paralleles Debugging für MPI

Beispiel 2:

The screenshot displays the Allinea Distributed Debugging Tool v2.6 interface. The main window shows the source code for a file named `deadlock.c`. The code is as follows:

```
17 *****/  
18  
19  
20 int main(int argc, char **argv){  
21  
22     int rank;           /* rank of process */  
23     int numprocs;      /* size of COMM_WORLD */  
24     int tag=10;        /* expected tag */  
25     int message;       /* Recv'd message */  
26     MPI_Status status; /* status of recv */  
27  
28     /* call Init, size, and rank */  
29     MPI_Init(&argc, &argv);  
30     MPI_Comm_size(MPI_COMM_WORLD, &numprocs);  
31     MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
32  
33     int source = rank-1; /* source process */  
34     int dest   = rank+1; /* process target */  
35     double *buffer;      /* Send buffer */  
36     buffer = (double *) malloc(sizeof(double)*SIZE);  
37  
38     /* fill buffer */  
39     int i;  
40     for (i=0; i<SIZE; i++){  
41         buffer[i]=i*0.0001;  
42     } /* end for */  
43  
44     /* zero out from last process
```

The interface also shows a 'Current Line(s)' window on the right with the text 'Process 0 is playing.' Below the code editor, there are panels for 'Breakpoints', 'Watches', 'Stacks (All)', and 'Evaluate'. The 'Evaluate' panel is currently empty. The status bar at the bottom right indicates '4 processes busy'.

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Paralleles Debugging für MPI

Beispiel 2:

The screenshot displays the Allinea Distributed Debugging Tool (DDT) v2.6 interface. The main window shows the source code of a C program named `deadlock.c`. The code is as follows:

```
41     buffer[i]=1*0.0001;
42 } /* end For */
43
44 /* zero gets from last process
45    rank numprocs-1 -> 0
46    last process sends to 0 */
47 if(rank == 0){ /* root process only */
48     printf("%i doubles will be sent\n",SIZE);
49     source = numprocs-1;
50 } /* end if */
51 if(rank == numprocs-1){
52     dest = 0;
53 } /* end if */
54
55 /* communicate */
56 MPI_Send(buffer, SIZE, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
57 MPI_Recv(buffer, SIZE, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
58
59 printf("%i rank finished\n",rank);
60
61 free(buffer); /* always free memory */
62 /* always call at end */
63 MPI_Finalize();
64
65 return 0;
66 }
```

The current stack is shown on the right side of the interface:

```
Stack Arguments
#6 main (argc=1, argv=0x7fff5a75bf28) at /home/...
#5 PMPI_Send () at /home/software/src/opens...
#4 mca_pml_ob1_send () at /home/software/s...
#3 ompi_request_wait_completion () at /home...
#2 opal_condition_wait () at /home/software/s...
#1 opal_progress () at /home/software/src/op...
#0 mca_pml_ob1_progress () at /home/softw...
```

The function call stack is shown at the bottom of the interface:

```
function
start_thread
├─bt_openib_async_thread (bt_openib_async.c:343)
│   └─poll
│       └─service_thread_start (bt_openib_fd.c:423)
│           └─select
│               └─main (deadlock.c:56)
│                   └─PMPI_Send (psend.c:67)
│                       └─mca_pml_ob1_send (pml_ob1_isend.c:122)
│                           └─ompi_request_wait_completion (pml_ob1_isend.c:374)
│                               └─opal_condition_wait (pml_ob1_isend.c:98)
```

The interface also shows a "Breakpoints" tab, a "Watches" tab, and a "Stacks (All)" tab. The "Evaluate" tab is active, showing a table with columns for "Expression" and "Value".

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

Paralleles Debugging für MPI

Message Queues:

DDT - Message Queues

Select queues to show:

- Send
- Receive
- Unexpected

Ranks:

- Show local ranks
- Show global ranks

Select communicator:

- MPI_COMM_WORLD
- MPI_COMM_SELF
- MPI_COMM_NULL

Update

Show queues in table

	Text	Communicator	Queue	Pointer	From (local)	From (global)	To (local)	To (global)
1	Send: 0x3e7...	MPI_COMM_...	Send	0x0	1	1	2	2
2	Send: 0x18f1...	MPI_COMM_...	Send	0x0	3	3	0	0
3	Send: 0x1d0...	MPI_COMM_...	Send	0x0	0	0	1	1
4	Send: 0x198...	MPI_COMM_...	Send	0x0	2	2	3	3

Stack Arguments:

- #6 main (argc=1, argv=0x7fff5a75bf28) at /home/...
- #5 PMPI_Send () at /home/software/src/openssl/...
- #4 mca_pml_ob1_send () at /home/software/src/openssl/...
- #3 ompi_request_wait_completion () at /home/software/src/openssl/...
- #2 opal_condition_wait () at /home/software/src/openssl/...
- #1 opal_progress () at /home/software/src/openssl/...
- #0 mca_pml_ob1_progress () at /home/software/src/openssl/...

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

DDT – Paralleles Debugging für MPI

Cross-Process Comparison:

The screenshot displays the Allinea Distributed Debugging Tool (DDT) v2.6 interface. A dialog box titled "DDT - Cross-Process Comparison View" is open, allowing for comparison across processes. The dialog includes options for "Compare across:" (Processes in current group, Threads in current process), an "Expression:" field, and checkboxes for "Array mode" and "Limit comparison to". A table with columns "Process(es)" and "Value" is present but empty. The background shows a code editor with a C program named "deadlock.c" and a stack view showing the current stack frame for "main".

```
41  buffer[i]=i*0.0001
42  } /* end for */
43
44  /* zero gets from last process
45  last process sends
46  if(rank == 0) { /* root
47  printf("%i doubles
48  source = numprocs-1
49  } /* end if */
50  } /* end if */
51  if(rank == numprocs-1
52  dest = 0;
53  } /* end if */
54
55  /* communicate */
56  MPI_Send(buffer, SIZE, MPI_INT, dest, 0, MPI_COMM_WORLD);
57  MPI_Recv(buffer, SIZE, MPI_INT, source, 0, MPI_COMM_WORLD);
58
59  printf("%i rank finished\n", rank);
60
61  free(buffer); /* always call at end */
62  /* always call at end */
63  MPI_Finalize();
64
65  return 0;
66 }
```

Stack Arguments:

- #6 main (argc=1, argv=0x7fff5a75bf28) at /home/...
- #5 MPI_Send () at /home/software/src/opens...
- #4 mca_pml_ob1_send () at /home/software/s...
- #3 ompi_request_wait_completion () at /home...
- #2 opal_condition_wait () at /home/software/s...
- #1 opal_progress () at /home/software/src/op...
- #0 mca_pml_ob1_progress () at /home/softw...

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

DDT – Paralleles Debugging für MPI

Vergleich bestimmter Variablen zwischen Prozessen:

The screenshot displays the Allinea Distributed Debugging Tool (DDT) v2.6 interface. The main window shows the source code for 'deadlock.c' with a breakpoint set at line 56, which is the MPI_Send function call. The 'Locals' window on the right shows the current state of variables: 'buffer' at address 0x3e84d20, 'dest' with value 2, and 'tag' with value 10. The 'Evaluate' window at the bottom shows the expression '-dest' evaluated to 2. The 'Stacks (All)' window shows the call stack, with the current frame being 'main (deadlock.c:56)'. The code in the main window includes a loop that sets 'dest' to 0 for the last process (rank == numprocs-1) and then sends a message to that process.

```
41     buffer[i]=i*0.0001;
42 } /* end for */
43
44 /* zero gets from last process
45    rank numprocs-1 -> 0
46    last process sends to 0 */
47 if(rank == 0) /* root process only */
48     printf("%i doubles will be sent\n",SIZE);
49     source = numprocs-1;
50 } /* end if */
51 if(rank == numprocs-1){
52     dest = 0;
53 } /* end if */
54
55 /* communicate */
56 MPI_Send(buffer, SIZE, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
57 MPI_Recv(buffer, SIZE, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
58
59 printf("%i rank finished\n",rank);
60
61 free(buffer); /* always free memory */
62 /* always call at end */
63 MPI_Finalize();
64
65 return 0;
66 }
```

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

DDT – Paralleles Debugging für MPI

Vergleich bestimmter Variablen zwischen Prozessen:

The screenshot displays the Allinea Distributed Debugging Tool (DDT) v2.6 interface. The main window shows a C code editor with the following code:

```
250 return return_value;
251 }
252
253 /*
254  * SM component initialization
255  */
256 mca_btl_base_module_t** mca_btl_sm_component_init(
257     int *num_btls,
258     bool enable_progress_threads,
259     bool enable_mpi_threads)
260 {
261     mca_btl_base_module_t **btls = NULL;
262     *num_btls = 0;
263
264     /* lookup/create shared memory pool only when used */
265     mca_btl_sm_component.sm_mpool = NULL;
266     mca_btl_sm_component.sm_mpool_base = NULL;
267
268     #if OMPI_ENABLE_PROGRESS_THREADS == 1
269     /* create a named pipe to receive events */
270     sprintf( mca_btl_sm_component.sm_fifo_path,
271             "%s"OPAL_PATH_SEP"sm_fifo.%lu", orte_process_info.job_session_dir,
272             (unsigned long)ORTE_PROC_MY_NAME->vpid );
273     if(mkfifo(mca_btl_sm_component.sm_fifo_path, 0660) < 0) {
274         opal_output(0, "mca_btl_sm_component_init: mkfifo failed with errno=%d\n",errno);
275     }
276     return NULL;
277 }
```

The stack trace on the left shows the following call stack:

- mca_btl_sm_component_init (bt1_sm_component.c:263)
- mca_btl_sm_component_progress (bt1_sm_component.c:379)
- mca_btl_sm_component_progress (bt1_sm_component.c:388)
- sm_fifo_read (bt1_sm_component.c:263)
- mca_pml_ob1_progress (pml_ob1_progress.c:26)
- opal_progress (opal_progress.c:206)
- opal_condition_wait (pml_ob1_isend.c:98)
- ompi_request_wait_completion (pml_ob1_isend.c:374)
- mca_pml_ob1_send (pml_ob1_isend.c:122)
- PMPI_Send (psend.c:67)
- main (deadlock.c:50)

The Evaluate window on the right shows the following expression and value:

Expression	Value
dest	<No symbol "dest" in current context.>

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

DDT – Paralleles Debugging für MPI

Vergleich bestimmter Variablen zwischen Prozessen:

The screenshot displays the Allinea Distributed Debugging Tool (DDT) v2.6 interface. The main window shows a code editor with the following C code snippet:

```
366 /* First, deal with any pending sends */
367 /* This check should be fast since we only need to check one variable. */
368 if ( 0 < mca_btl_sm_component.num_pending_sends ) {
369
370     /* perform a loop to find the endpoints that have pending sends */
371     /* This can take a while longer if there are many endpoints to check. */
372     for ( peer_smp_rank = 0; peer_smp_rank < mca_btl_sm_component.num_smp_procs; peer_smp_rank++)
373         struct mca_btl_base_endpoint_t* endpoint;
374         if ( peer_smp_rank == my_smp_rank )
375             continue;
376         endpoint = mca_btl_sm_component.sm_peers[peer_smp_rank];
377         if ( 0 < opal_list_get_size(&endpoint->pending_sends) )
378             mca_btl_sm_process_pending_sends(endpoint);
379     }
380 }
381
382 /* poll each fifo */
383 for(j = 0; j < FIFO_MAP_NUM(mca_btl_sm_component.num_smp_procs); j++) {
384     fifo = &(mca_btl_sm_component.fifo[my_smp_rank][j]);
385     recheck_peer:
386     /* acquire thread lock */
387     if(opal_using_threads()) {
388         opal_atomic_lock(&(fifo->tail_lock));
389     }
390
391     hdr = (mca_btl_sm_hdr_t *)sm_fifo_read(fifo);
392
393     /* release thread lock */
```

The interface includes a 'Locals' panel on the right, which is currently empty. Below the code editor is a 'Stacks (All)' panel showing the current call stack:

- main (deadlock.c:50)
- PMPI_Send (psend.c:67)
- mca_pml_ob1_send (pml_ob1_isend.c:122)
- ompi_request_wait_completion (pml_ob1_isend.c:374)
- opal_condition_wait (pml_ob1_isend.c:98)
- opal_progress (opal_progress.c:206)
- mca_btl_sm_component_progress (btl_sm_component.c:379)**
- mca_btl_sm_component_progress (btl_sm_component.c:388)
- sm_fifo_read (btl_sm_component.c:263)
- mca_pml_ob1_progress (pml_ob1_progress.c:26)

The 'Evaluate' panel at the bottom right shows the expression 'dest' with the value '<No symbol "dest" in current context.>'. The status bar at the bottom right indicates '1 process busy'.

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

DDT – Paralleles Debugging für MPI

Vergleich bestimmter Variablen zwischen Prozessen:

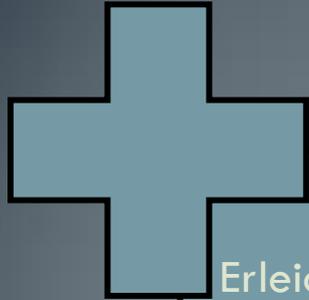
The screenshot displays the Allinea Distributed Debugging Tool (DDT) v2.6 interface. The main window shows a code editor with C code from the file `bt1_sm_component.c`. The code is currently at line 379, which is highlighted in blue. The code includes a loop for finding endpoints with pending sends and a section for polling each fifo. The stack panel on the right shows the current call stack, with `mca_bt1_sm_component_progress (bt1_sm_component.c:379)` at the top. The locals panel is empty. The bottom panel shows the Evaluate window with the expression `dest` and the value `<No symbol "dest" in current context.>`.

```
366 /* This check should be fast since we only need to check one variable. */
367 if ( 0 < mca_bt1_sm_component.num_pending_sends ) {
368
369
370 /* perform a loop to find the endpoints that have pending sends */
371 /* This can take a while longer if there are many endpoints to check. */
372 for ( peer_smp_rank = 0; peer_smp_rank < mca_bt1_sm_component.num_smp_procs; peer_smp_rank++)
373 struct mca_bt1_base_endpoint_t* endpoint;
374 if ( peer_smp_rank == my_smp_rank )
375 continue;
376 endpoint = mca_bt1_sm_component.sm_peers[peer_smp_rank];
377 if ( 0 < opal_list_get_size(&endpoint->pending_sends) )
378 mca_bt1_sm_process_pending_sends(endpoint);
379 }
380 }
381
382 /* poll each fifo */
383 for(j = 0; j < FIFO_MAP_NUM(mca_bt1_sm_component.num_smp_procs); j++) {
384 fifo = &(mca_bt1_sm_component.fifo[my_smp_rank][j]);
385 recheck_peer:
386 /* acquire thread lock */
387 if(opal_using_threads()) {
388 opal_atomic_lock(&(fifo->tail_lock));
389 }
390
391 hdr = (mca_bt1_sm_hdr_t *)sm_fifo_read(fifo);
392
393 /* release thread lock */
```

UMCoECAC - DDT Parallel Debugging for MPI - <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

...und zum Schluss

Ist Debugging immer nützlich?



Erleichtert Fehlersuche

Vielfältige Debuggingmöglichkeiten
und Debuggingtools

Umgang mit Debuggingsoftware
einfach und leicht zu erlernen

Debugging erspart einem viele
Nerven ;)

Evtl. fallen Anschaffungskosten
an

Je nach Methode und Software
ist know-how erforderlich

„Manchmal sieht man den
Fehler vor lauter Debugging
nicht“

Zusammenfassung & Fazit

- Grundsätzlich ist Debugging immer dann sinnvoll, wenn ein Programm einen Fehler hat, der erst zur Laufzeit auftritt und nicht im Vorfeld von der IDE erkannt wird
- Trotz der vielen Vorteile von Debuggingtools, sollte man diese nicht ohne Bedacht benutzen
- Statische Analyse vs. Dynamisches Debugging

Quellen

Textquellen:

[1] Markus Dausch – Das Debugging Dilemma

[2] Economictimes - Definition Debugging -
<https://economictimes.indiatimes.com/definition/debugging>

[3] Boris Veldhuijzen van Zanten - The very first recorded computerbug. Erschienen auf www.thenextweb.com <https://thenextweb.com/shareables/2013/09/18/the-very-first-computer-bug/>

[4] Björn Schmitz – Statische Code Analyse – Analysieren Sie schon oder debuggen Sie noch? – Erschienen auf www.medtech-ingenieur.de <https://medtech-ingenieur.de/statische-code-analyse-analysieren-sie-schon-oder-debuggen-sie-noch/>

[5] UMCoECAC - DDT Parallel Debugging for MPI. Erschienen auf www.youtube.de <https://www.youtube.com/watch?v=Pf4yT3ugT-4>

[6] Wikipedia – Debugging - <https://en.wikipedia.org/wiki/Debugging>

Quellen (2)

Bildquellen:

[1] Boris Veldhuijzen van Zanten - The very first recorded computerbug. Erschienen auf [www.thenextweb.com](https://thenextweb.com/shareables/2013/09/18/the-very-first-computer-bug/) <https://thenextweb.com/shareables/2013/09/18/the-very-first-computer-bug/>

[2] UMCoECAC - DDT Parallel Debugging for MPI. Erschienen auf www.youtube.de <https://www.youtube.com/watch?v=Pf4yT3ugT-4>