RC 20085 6/1/95
Computer Science/Mathematics

IBM Research Report

Analysis of an MPI Program using UTE on the IBM SP2

Joefon Jann and Hubertus Franke

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Analysis of an MPI Program using UTE on the IBM SP2

Joefon Jann and Hubertus Franke
email: joefon@watson.ibm.com frankeh@watson.ibm.com

June 6, 1995

ABSTRACT

We describe an implementation of a 2D-FFT (Complex-Complex) program in MPI-F on the SP2 and show its actual performance. The purpose of this paper is to illustrate how we use the new tracing utility UTE/MPI provided in MPI-F to verify the correctness of our algorithm, to provide timing statistics summaries, and to unravel other system activities, often unexpected by the user, that affect the total elapsed time of the program.

1 Introduction

Message Passing Interface (MPI) is a library of standardized functions and macros that can be used in FORTRAN, C, and C++ programs to achieve parallel processing via message passing on distributed memory machines [2]. MPI-F is the IBM Research implementation of the MPI standard on the IBM SP2 (Scalable POWERParallel) systems and workstation clusters. It provides a full and efficient implementation of the MPI standard and runs under POE (Parallel Operating Environment) under AIX [3]. Recently, MPI-F also provides a performance tracing facility that we feel is quite useful to the parallel user. This is the UTE/MPI (Unified Trace Environment) library which allows us to trace MPI activities of the program, as well as other system events e.g. context switches, page faults, system calls, and I/O events occurring during the life of a program [1]. Since these system activities affect the total elapsed time of the program, an item of importance to the parallel user, this tool provides very useful information often not provided by other tools. UTE/MPI is an IBM Research prototype that is built on top of the AIX trace facility and the upshot utility from Argonne National Laboratory.

In this paper, we take a 2D FFT (Complex-Complex) program written by us in MPI-F, and use the UTE tracing facility to analyze its performance.

In section 2, we describe the particular implementation of our 2D FFT program.

In section 3, we describe the actual performance of the program using MPI-F on an SP2.

In section 4, we describe various ways to use the tracing facility in MPI-F, and an anatomy of the activities in the system during the life of the program. This section sheds light on the observed variance in the performance of a program running on an SP2 even in single-user mode. This variance can be primarily attributed to page faults, context switches, and hidden library loads that occur, often unexpected by the user.

2 Description of the 2D-FFT algorithm

- Let P be the number of processors (preferably a power of 2).
- We consider an $N \times N$ grid of sampled data points on which we perform the FFT.
- We divide this grid into $N/P \times N/P$ tiles (or subsquares).
- Let each processor work on its $(N/P) \times 1$ column of tiles, or equivalently, on its vertical strip of $N \times (N/P)$ points.
- The algorithm for each processor is as follows:

1. Perform a 1D-FFT on each of its N/P columns.
2. Transpose the data.
   (a) For the tile that is a diagonal tile (in the tile matrix), simply perform a transpose of its component points.
   (b) Copy each of its non-diagonal $P-1$ tiles to linear arrays to be sent out to $P-1$ other processors. Each outgoing buffer (or linear array) is of size $(N/P) \times (N/P)$.
   (c) Post a non-blocking receive from each of the other $P-1$ processors for the $P-1$ incoming buffers.
   (d) Post a non-blocking send of each of its $P-1$ outgoing buffers.
   (e) Wait for any receive to complete. On a receive completion, copy its incoming buffer to its position in the $N \times N$ grid.
3. Do a 1D-FFT on each of its N/P columns again.

3 Performance results of the program on the IBM SP2

In performing FTs, FFT algorithms reduce an $O(N^2)$ solution to an $O(N \times logN)$ solution. Hence FFTs offer a significant advantage over many other solution techniques. With the increasing availability of parallel computers like the SP2, which offers very good scalability on a large class of problems, the solution time can be further reduced by a factor of close to P (P being the number of processors).

A good deal of work has been done to implement nD FFTs in parallel machines of various architectures, e.g. hypercubes, rings, etc. This implementation takes advantage of the following strengths of the IBM SP2, namely:

1. The existence of a serial library subroutine that performs 1D-FFT very efficiently – namely ESSL's SCFT subroutine.

2. A parallel computer architecture that allows all processors to simultaneously exchange data, and to do so with a constant delay factor.

The table below shows our experimental results of our implementation of the algorithm described in section 2. The timings were obtained on an SP2 with thin 66.7 MHz nodes, each with 128 MB memory, 64 KB of data cache, and 32 KB instruction cache.

- Column 3 labeled (2x1DFFT) corresponds to steps 1 and 3 of the algorithm.

- Column 4 corresponds to step 2 of the algorithm. This includes message passing communication time, plus time for data manipulation before and after the communication.

- The last column is the total time for the 2D-FFT, and equals the sum of the previous 2 columns.

N	P	secs(2x1DFFT)	secs(transpose step)	secs(2DFFT total)
512	2	.091339	.077036	.168375
1024	2	.435713	.335173	.770886
2048	2	2.208154	1.677079	3.885233
512	4	.047234	.051490	.098724
1024	4	.218839	.203398	.422236
2048	4	1.122635	1.031558	2.154193
4096	4	5.251962	4.271225	9.523187
512	8	.023838	.031321	.055159
1024	8	.110853	.116663	.227516
2048	8	.568217	.484477	1.052694
4096	8	2.662863	2.297205	4.960068
512	16	.012153	.018327	.030480
1024	16	.055953	.065634	.121587
2048	16	.281510	.255120	.536630
4096	16	1.295190	1.211026	2.506215
512	32	.006215	.009307	.015522
1024	32	.028530	.037341	.065872
2048	32	.140045	.132845	.272890
4096	32	.643114	.583696	1.226810

Shown below is a plot of the N=1024 case of the data from the above table. Note that the x-axis is in log scale.

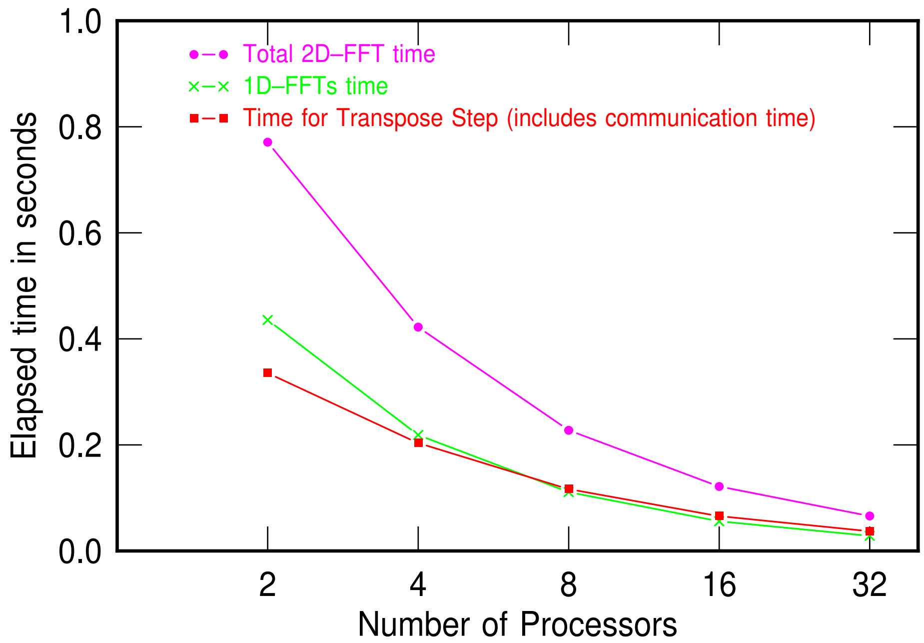

Figure 1: Single precision complex 2D-FFT on 1Kx1K points using SP2 nodes.

4 Using UTE/MPI tracing facility to analyse the program

The UTE/MPI tracing facilty provides us with several advantages. These are are best explained by referring to figures 2 through 5 at the end of the paper. Figures 2 and 3 are zoomed-in views of figure 4 which is the trace of one POE job running the 2D-FFT 4 times on 8 processors. Figure 5 is the overview of a trace on 4 processors. In all the figures, the x-axis denotes the elapsed time in seconds, and the y-axis denotes the processor numbers.

1. UTE/MPI enables us to verify the correctness of the communication patterns in our algorithm. For example, in figures 2 or 3, we can verify that data is being sent from each processor to the correct destination processor by viewing the directions of the arrows at each message exchange step. Also shown are the 7 message exchange steps, at the end of which each processor has exchanged data with every other processor. The red lines between the boxes mean that the process is not in MPI communication and is not swapped out by other processes, thus is probably doing useful computation. One can also display the source code line associated with a process state by simply clicking (with CTL + mouse button 1) on a process state box in the figure.

2. UTE/MPI gives us insight into other system events that slow down parts of the program, and the extent of their impact. In each figure, the "Other Processes" displayed can be page fault events, process dispatch (or context switching) events, SCSI disk drive events e.g. reads and writes, and hidden library loads.

Figures 4 and 5 display a loop that runs the program 4 times. This was purposely done in an attempt to unravel the existence of "Other Processes" that lengthen the elapsed time of the program.

- The first set of delays in figures 4 or 5 (before the first MPI_Barriers) can very possibly be due to library load of the POE MPI-F communication libraries and NFS(Network File System) or AMD (AutoMounter Daemon) setup latencies.
- Also in figures 4 and 5, the delay at the end of the trace for processor 0 is due to I/O writes of the results to the output file.
- In figure 2, the zoomed-in view of the first run, the delays at the beginning may be due to load of the dynamically linked, but large, serial ESSL library.
- Subsequent runs shown in figures 3 and 4 do not display these delays.

3. UTE/MPI enables us to easily display timing statistics summaries for the processors.

```
Examples illustrating advantage 3 follow:

a) The command:

     utemerge -n8 tr8. | lsute -p

   merges the 8 trace files (one per node) and gives the components of
   the total elapsed time of the trace, i.e.

   elapsed_time = other_process time + idle_time + main_process time

node   pid elapsed_time other_process idle_time main_process(compute_time)
  0: 17243  3.323241984  0.048410112  1.390130  1.884701696(1.641198848)
  1: 15538  2.697278976  0.027015424  0.716104  1.954159104(1.614765056)
  2: 14594  2.802544384  0.014525440  0.669789  2.118230016(1.611845632)
  3:  6787  2.701424128  0.013748224  0.669876  2.017800192(1.605531392)
  4: 12812  2.706512896  0.013521664  0.652934  2.040057344(1.603267072)
  5: 14682  2.680584192  0.011736832  0.715657  1.953190144(1.602795008)
  6: 14100  2.704121344  0.027019008  0.723690  1.953411840(1.605327360)
  7: 14353  2.751921408  0.015162112  0.725758  2.011001600(1.605473792)
```

b) The command:

```
utemerge -n4 tr8. | lsute -h
```

merges 4 trace files prefixed by tr8., and gives the timings for each MPI call used:

```
              Total.elapsed.time   #calls  Average.elapsed.time.per.call

    112 MPI_Send
node   0     0.225808640      28  0.008064594
node   1     0.255563520      28  0.009127269
node   2     0.247722752      28  0.008847241
node   3     0.256357888      28  0.009155639
    112 MPI_Irecv
node   0     0.000569344      28  0.000020334
node   1     0.000770304      28  0.000027511
node   2     0.000744704      28  0.000026597
node   3     0.000461056      28  0.000016466
    112 MPI_Wait
node   0     0.001842176      28  0.000065792
node   1     0.003654912      28  0.000130533
node   2     0.010813952      28  0.000386213
node   3     0.012900864      28  0.000460745
     16 MPI_Barrier
node   0     0.011752192       4  0.002938048
node   1     0.063664128       4  0.015916032
node   2     0.127234048       4  0.031808512
node   3     0.122425856       4  0.030606464
      4 MPI_Comm_rank
node   0     0.000003328       1  0.000003328
node   1     0.000003328       1  0.000003328
node   2     0.000003584       1  0.000003584
node   3     0.000003072       1  0.000003072
      4 MPI_Init
node   0     0.004122112       1  0.004122112
node   1     0.016136704       1  0.016136704
node   2     0.121886208       1  0.121886208
node   3     0.020936704       1  0.020936704
      4 MPI_Comm_size
node   0     0.000009216       1  0.000009216
node   1     0.000008704       1  0.000008704
node   2     0.000009216       1  0.000009216
node   3     0.000008960       1  0.000008960
```

5 Conclusion

The UTE/MPI utility is very easy to use, and does not require any code changes. Moreover, it is a delight to display the various colorful outputs which can verify the correctness of our algorithm, as well as unravel other system events that affect the overall elapsed time of the program. Work is in progress on the UTE/MPI utility, to report in more details the components of the "Other Processes". We have also shown that our algorithm scales rather linearly for problem sizes from .5K to 4K running on the SP2 with MPI-F under POE.

6 Acknowledgements

We would like to acknowledge C. Eric Wu for providing UTE, and Rick Lawrence for his support and suggestions for improvements.

References

[1] C. Eric Wu and Hubertus Franke, *UTE User's Guide for IBM SP Systems*, 4/95 (cwu@watson.ibm.com).

[2] The MPI Forum, *MPI: A Message Passing Interace Standard*, 5/95.

[3] William Gropp, Ewing Lusk, and Anthony Skjellum, *Using MPI – Portable Parallel Programming with the Message-Passing Interface*, ISBN 0-262-57104-8, MIT Press 1994.

[4] Hubertus Franke, *MPI-F Version 1.41 User Guide*, 5/95 (frankeh@watson.ibm.com)

[5] IBM *AIX Parallel Environment Operation and Use, release 2.1 manual*, SH26-7230-01 12/94.

[6] IBM *Engineering and Scientific Subroutine Library Guide and Reference manual*, version 2 release 2 volume 3 SC23-0526-01 1/94.

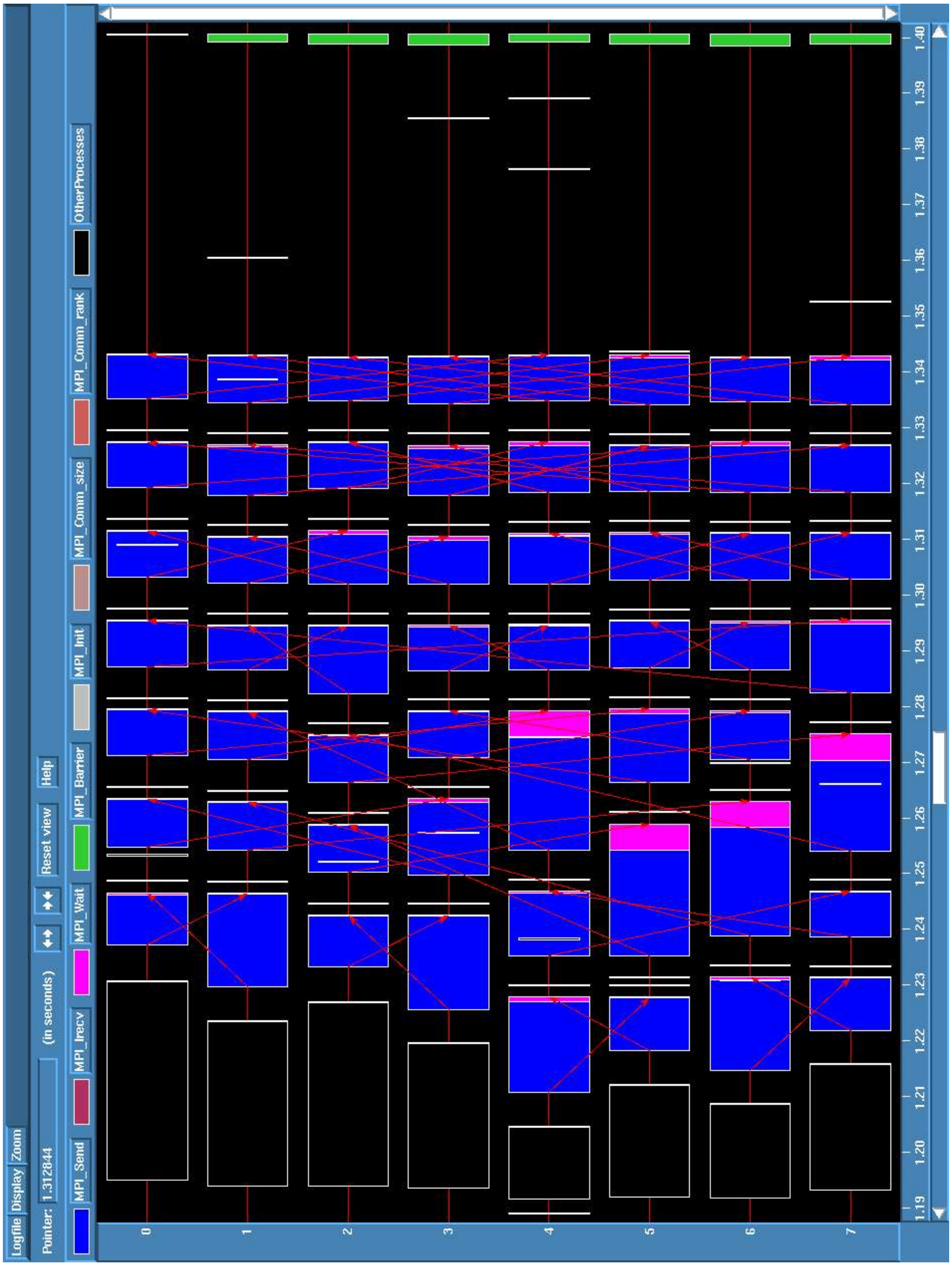

Figure 2: Zoomed-in view of UTE/MPI trace of the 1st run of 2D-FFT on 8 processors

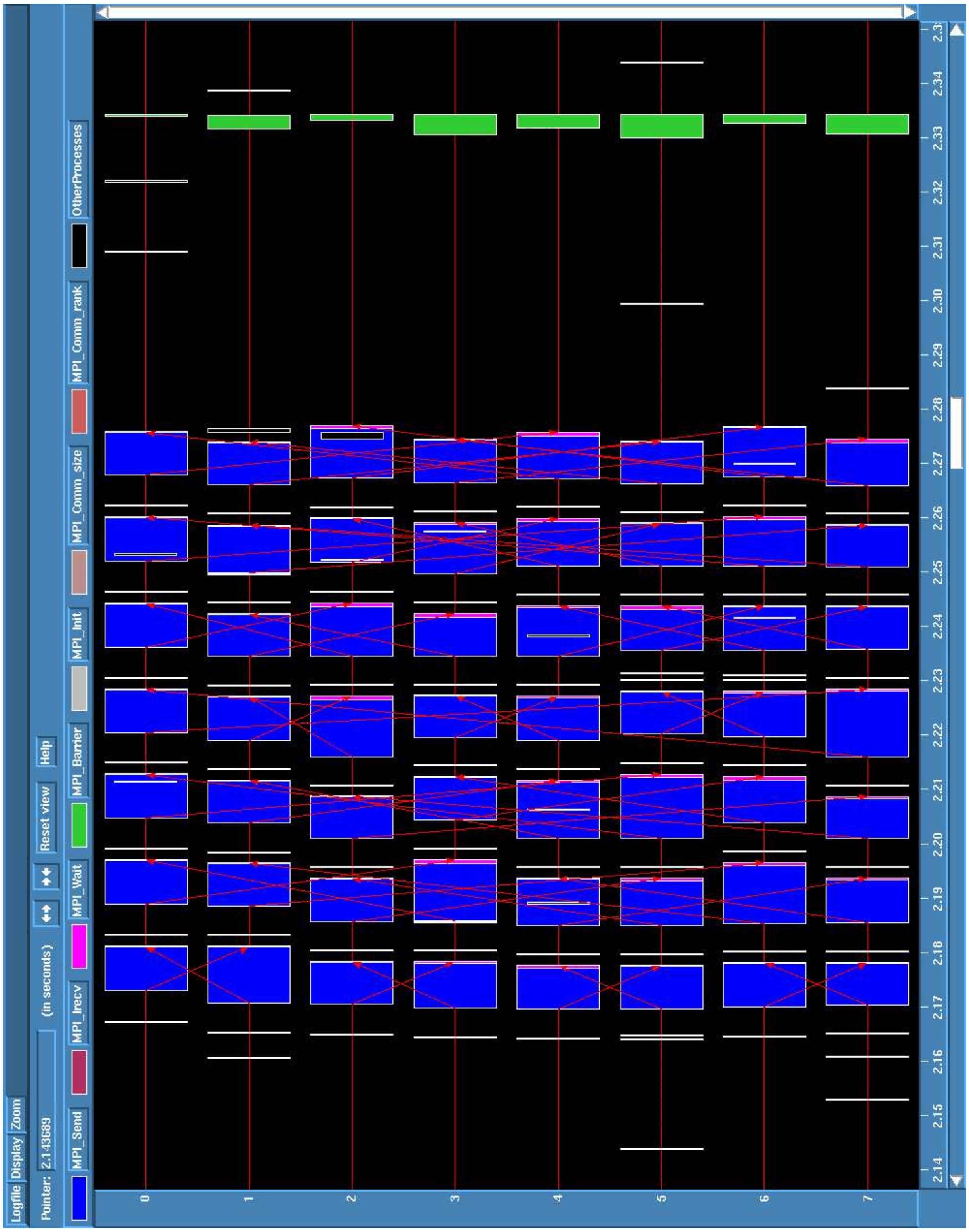

Figure 3: Zoomed-in view of UTE/MPI trace of the 3rd run of 2D-FFT on 8 processors

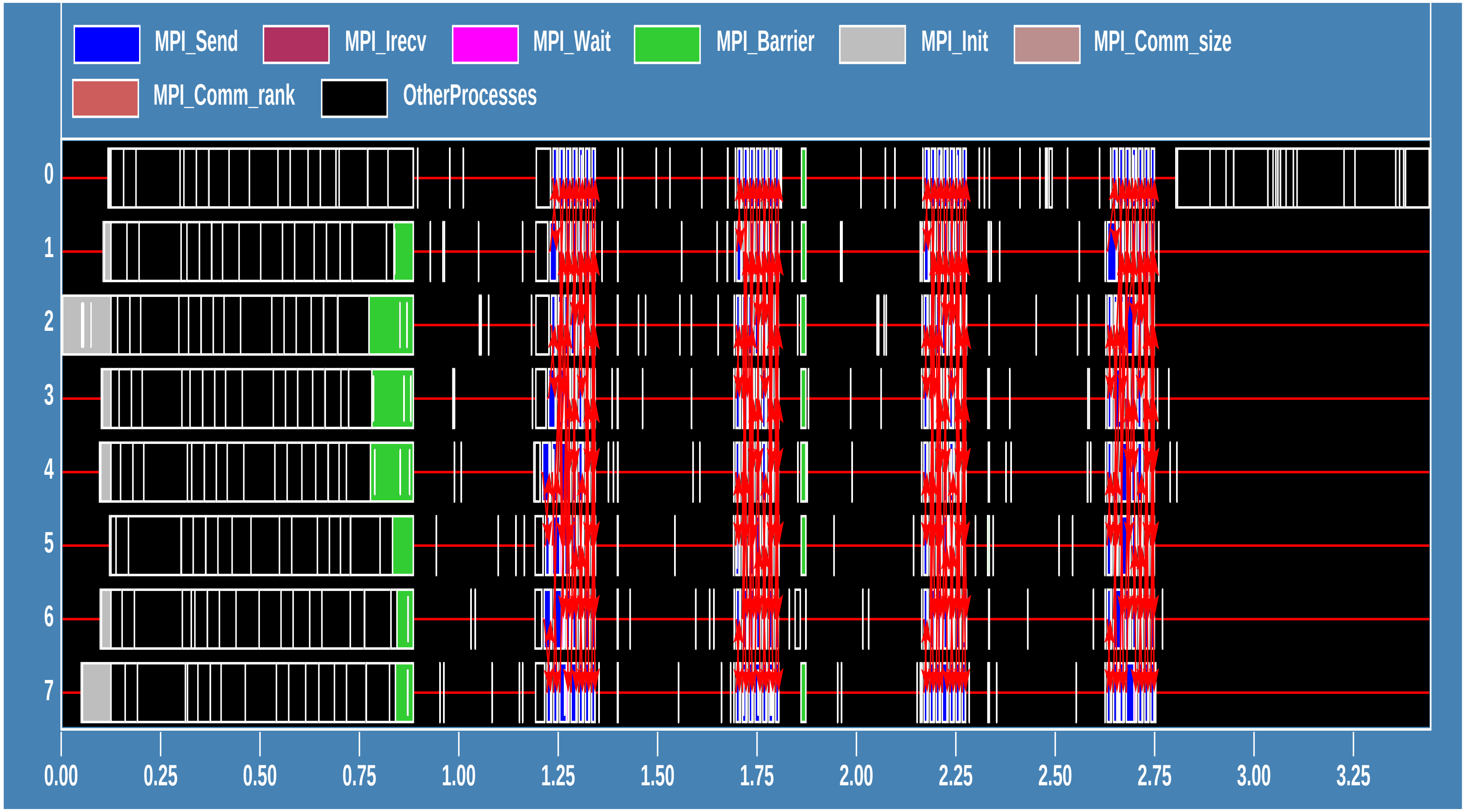

Figure 4: Overview of the UTE/MPI trace for 4 runs of 2D-FFT on 8 processors

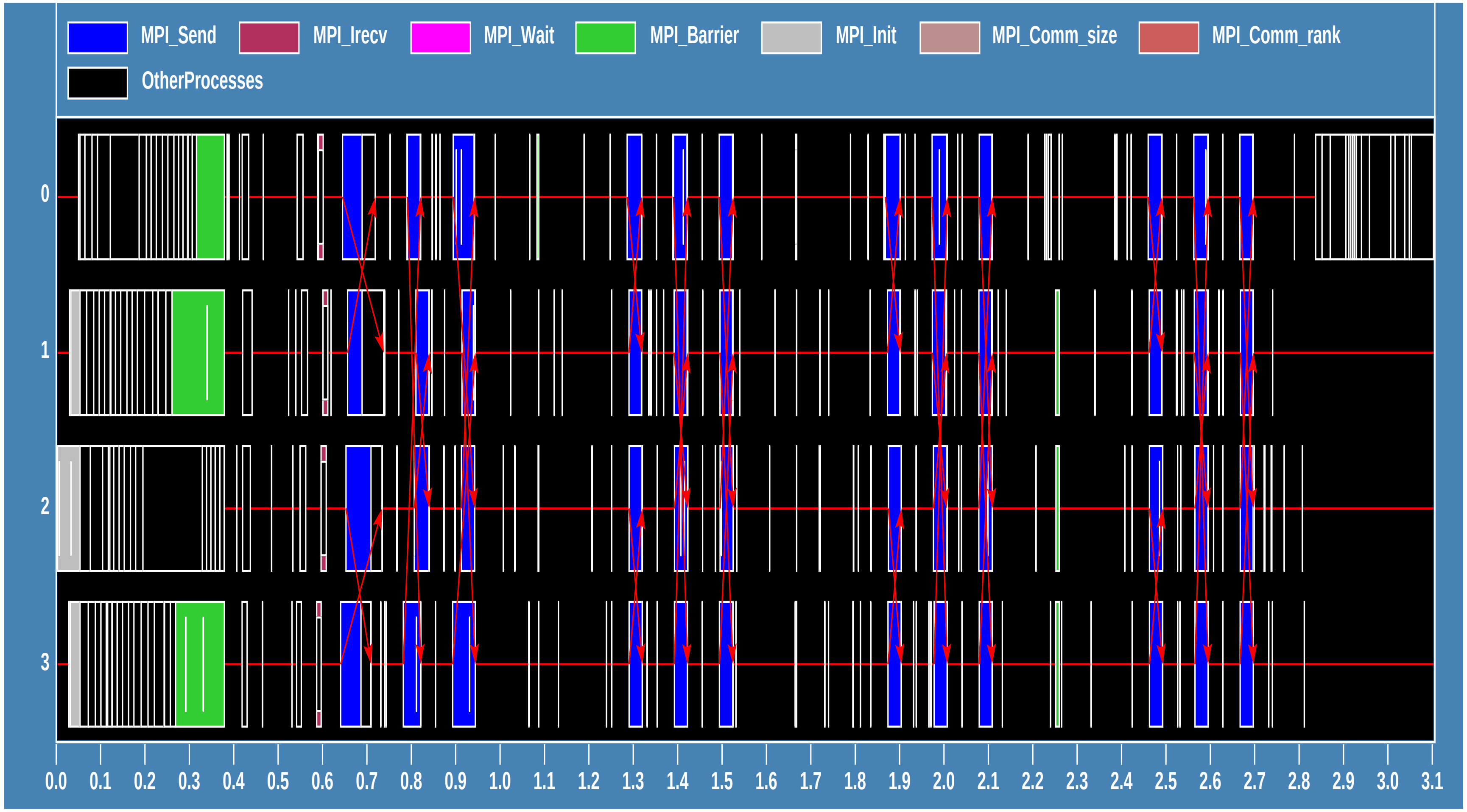

Figure 5: Overview of the UTE/MPI trace for 4 runs of 2D-FFT on 4 processors