Improving Interprocess Communication in Globl/solve &

Improving Slew Models for IVS antennas

Lina Olandersson, Erik Thorsell, Simon Strandberg

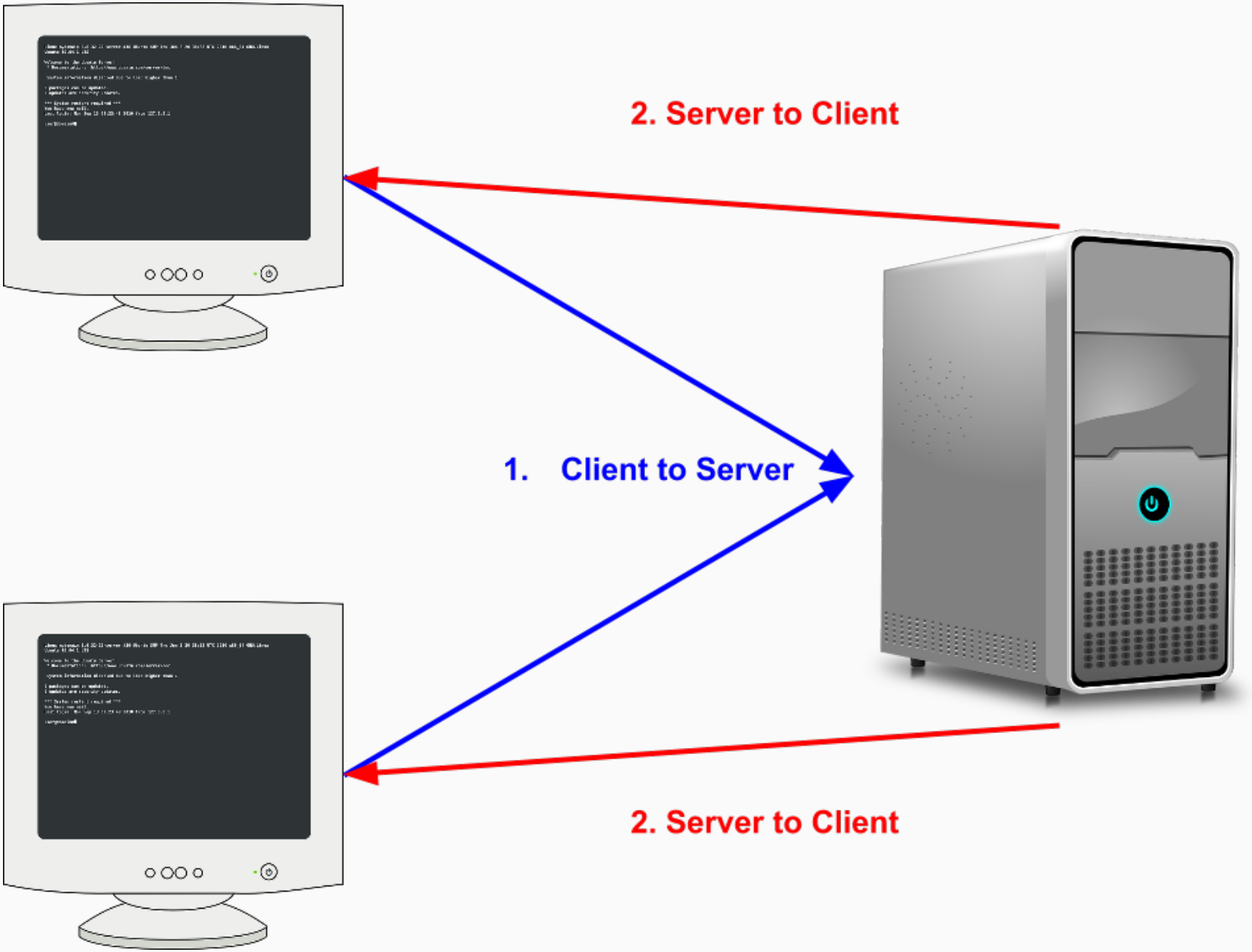
2016-08-18

Created for NVI Inc. at the Goddard Space Flight Centre

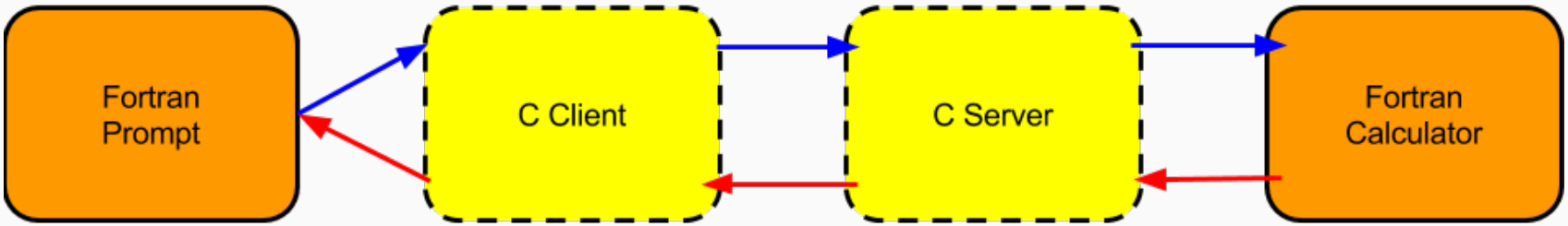
Table of Contents

Learning Fortran

Server/Client



Server/Client



Calculator

```
erik@Antergos-Laptop ~/Programming/git/GSFC_Internship/server_client >./calculator_client.out
 Please enter the same port number as for the server (e.g. 55555).
55555
 Usage: Write two numbers followed by an operator (add, sub, mul, div).
 Only integer answers are supported. (2 4 div will return 0)
1238 4883 add
 The answer is:         6121
erik@Antergos-Laptop ~/Programming/git/GSFC_Internship/server_client >./calculator_client.out
 Please enter the same port number as for the server (e.g. 55555).
55555
 Usage: Write two numbers followed by an operator (add, sub, mul, div).
 Only integer answers are supported. (2 4 div will return 0)
128 412 mul
 The answer is:        52736
erik@Antergos-Laptop ~/Programming/git/GSFC_Internship/server_client >./calculator_client.out
 Please enter the same port number as for the server (e.g. 55555).
55555
 Usage: Write two numbers followed by an operator (add, sub, mul, div).
 Only integer answers are supported. (2 4 div will return 0)
121 11 div
 The answer is:           11
erik@Antergos-Laptop ~/Programming/git/GSFC_Internship/server_client >./calculator_client.out
 Please enter the same port number as for the server (e.g. 55555).
55555
 Usage: Write two numbers followed by an operator (add, sub, mul, div).
 Only integer answers are supported. (2 4 div will return 0)
2138 1312 sub
 The answer is:          826
erik@Antergos-Laptop ~/Programming/git/GSFC_Internship/server_client >_
```

Figure 3: Example usage of the calculator.

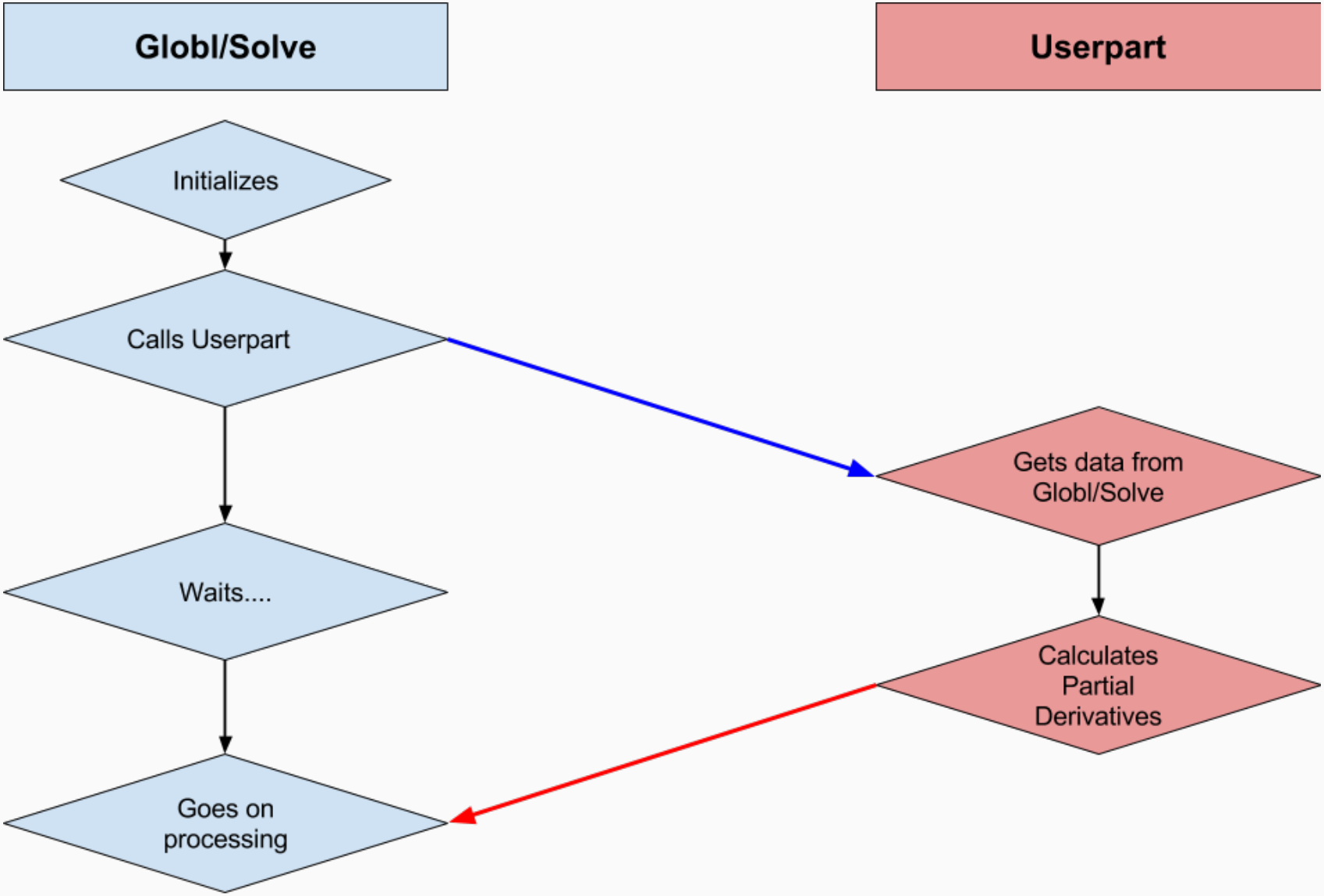
Our First Project

Problem Description

Rewrite how Globl/solve handles its passing of data to and from userpartials and userprogs.

By minimizing disc I/O we want to increase the speed at which data is sent.

Problem Description



Testing I/O performance

I/O Performance

Protocols we have tried and used:

I/O Performance

Protocols we have tried and used:

- Using the hard drive:

I/O Performance

Protocols we have tried and used:

- Using the hard drive:
  - Read/Write to **file**

I/O Performance

Protocols we have tried and used:

- Using the hard drive:
  - Read/Write to **file**
  - Sending/Receiving with **pipes**

I/O Performance

Protocols we have tried and used:

- Using the hard drive:
  - Read/Write to **file**
  - Sending/Receiving with **pipes**
- Using the memory:

I/O Performance

Protocols we have tried and used:

- Using the hard drive:
  - Read/Write to **file**
  - Sending/Receiving with **pipes**
- Using the memory:
  - Sending/Receiving with **TCP/IP** Sockets

I/O Performance

Protocols we have tried and used:

- Using the hard drive:
  - Read/Write to **file**
  - Sending/Receiving with **pipes**
- Using the memory:
  - Sending/Receiving with **TCP/IP** Sockets
  - Sending/Receiving with **OpenMPI**

I/O Performance

Protocols we have tried and used:

- Using the hard drive:
  - Read/Write to **file**
  - Sending/Receiving with **pipes**
- Using the memory:
  - Sending/Receiving with **TCP/IP** Sockets
  - Sending/Receiving with **OpenMPI**
  - Sending/Receiving with **ZeroMQ**

Performance Test

1. The producer generates a list of length n and fills it with integers.

Performance Test

1. The producer generates a list of length n and fills it with integers.
2. The producer writes the list to file (or sends it over the designated transfer protocol).

Performance Test

1. The producer generates a list of length n and fills it with integers.
2. The producer writes the list to file (or sends it over the designated transfer protocol).
3. The consumer reads (or receives) the list.

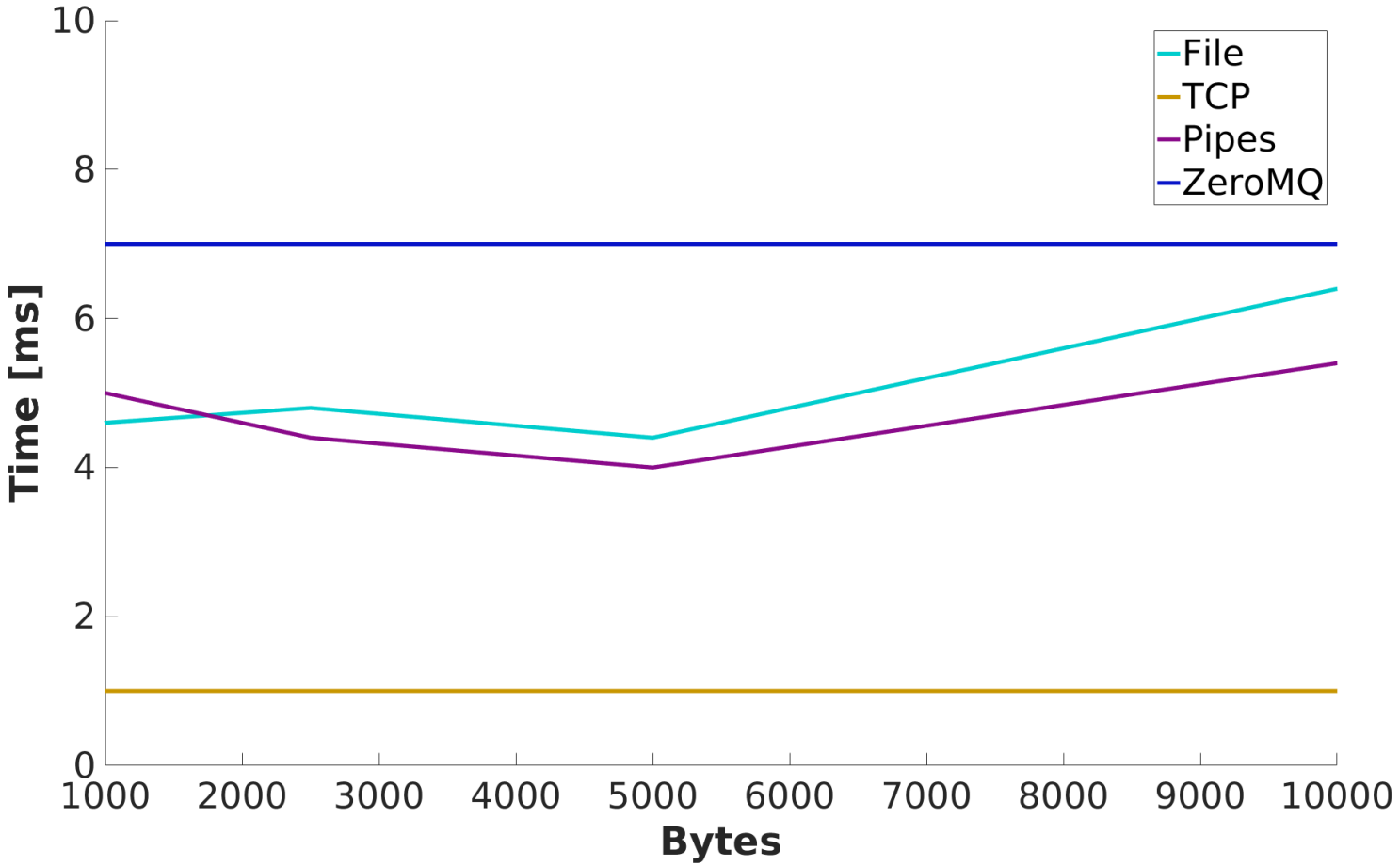
Performance Test

1. The producer generates a list of length n and fills it with integers.
2. The producer writes the list to file (or sends it over the designated transfer protocol).
3. The consumer reads (or receives) the list.
4. The consumer squares each int in the list and sends it back to the producer.

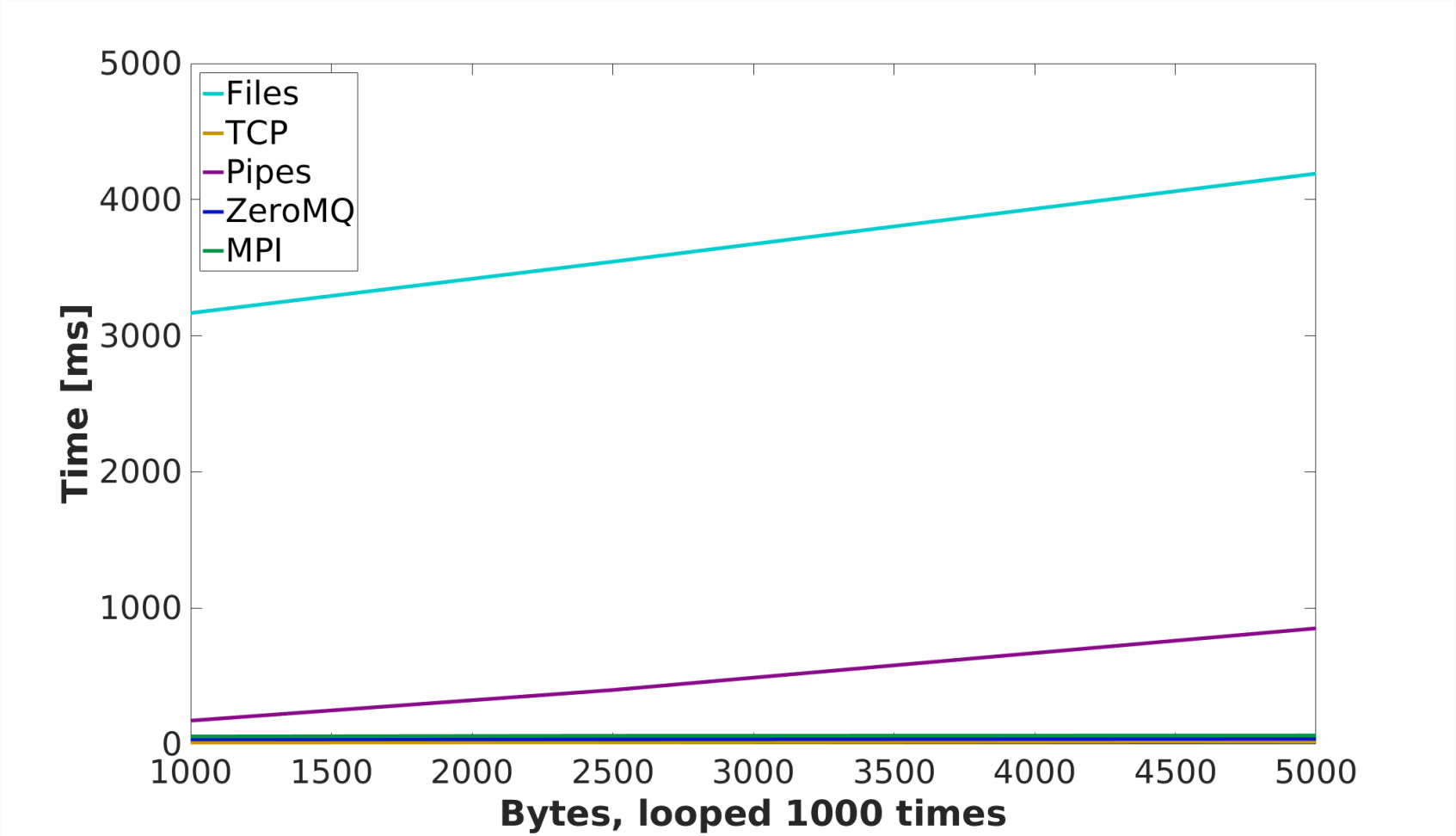
Performance Test

1. The producer generates a list of length n and fills it with integers.
2. The producer writes the list to file (or sends it over the designated transfer protocol).
3. The consumer reads (or receives) the list.
4. The consumer squares each int in the list and sends it back to the producer.
5. The producer reads (or receives) the modified list.

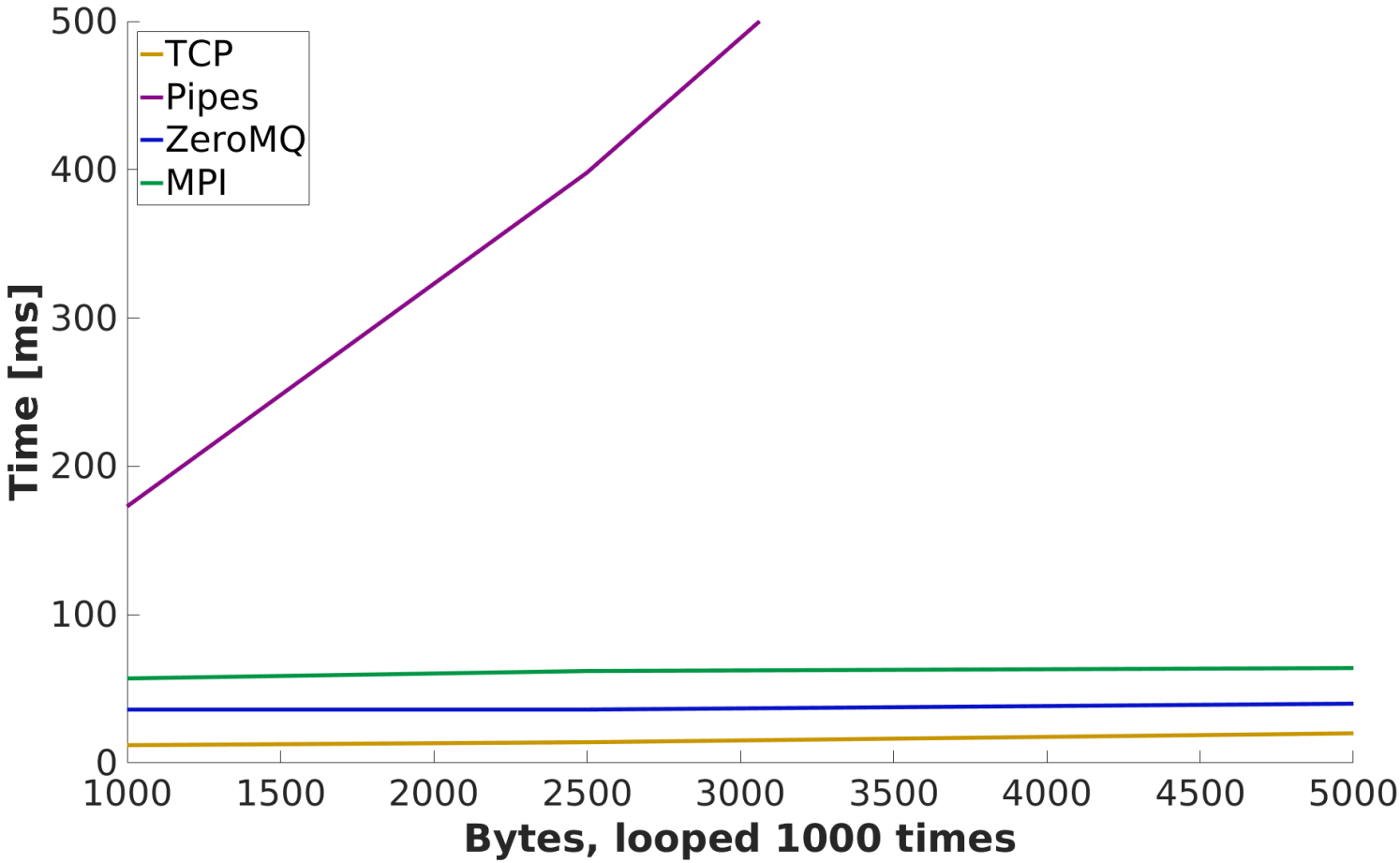
Result for I/O Performance



Result for I/O Performance



Result for I/O Performance



Result for I/O Performance

- TCP was the fastest, but the most difficult to implement.

Result for I/O Performance

- TCP was the fastest, but the most difficult to implement.
- Since we assumed a lot of data would be passed we opted for ZeroMQ due to its presumptive ease of use and performance.

Implementation

Implementation

- Installation of ZeroMQ on **bootes**

Implementation

- Installation of ZeroMQ on **bootes**
- Porting our code to a different compiler: gfortran $\rightarrow$ ifort

Implementation

- Installation of ZeroMQ on **bootes**
- Porting our code to a different compiler: gfortran $\rightarrow$ ifort
- Understanding where to put what and why

Implementation

- Installation of ZeroMQ on **bootes**
- Porting our code to a different compiler: gfortran $\rightarrow$ ifort
- Understanding where to put what and why
- A lot of coding

Implementation

- Two programs:

```
include 'solve.i'             ! Includes parameters used by oborg.i
include 'oborg.i'             ! Includes the common block used for testing
include 'f77_zmq.h'           ! Includes the ZMQ binding
include 'variables_client.i'  ! Includes a small common block holding the ZMQ related variables

integer, dimension(22) :: i_buffer ! Buffer for data tranmsmition
integer, dimension(20) :: i_array, i_array1, i_array2, i_array3 ! Generic arrays
integer                   i_rc ! if >= 0: the number of bytes sent or received, else: error
integer                   i, i_lbuf, c, i_sbuf, buffNumber, i_obsNumber

! Set up the communication.
address = 'tcp://localhost:55555'               ! Address to listen to.
context   = f77_zmq_ctx_new()                   ! Create new context
requester = f77_zmq_socket(context, ZMQ_REQ)    ! Use the REQ protocol
i_rc        = f77_zmq_connect(requester,address)! Declare the client as a requester

i_lbuf = size(i_buffer)
i_sbuf = i_lbuf * 4

! Clear the buffer
do i=1, i_lbuf
    i_buffer(i) = 0
enddo

! Fill the arrays with arbitrary numbers. The arrays hold two less elements
! since the buffer carries two control bits (buffer(1), buffer(2)).
do i=1, (i_lbuf-2)
    i_array1(i) = 21
    i_array2(i) = 22
    i_array3(i) = 23
enddo

! Invoke the program, l 121
call init(i_buffer, i_rc)

! Due to the nature of the program, a loop is used to provide the opportunity
! to test all functions. When used in production, a program would be written
! that makes use of the subroutines, with no need for human interaction.
do while (.true.)
    if ((i_buffer(1) .eq. 9) .and. (i_rc .ge. 0)) then
        print *, "All is good, please chose between the options below:"
        print *, "(1) Send"
        print *, "(2) Get"
        print *, "(3) Send Obs"
        print *, "(4) Get Obs"
        print *, "n/a (5) Get Size"
        print *, "n/a (6) Done"
        print *, "(7) Exit"
        read(*,*) c

        select case (c)
            case (1)
                print *, "Which array do you want parent to send (1-3)?"
                read(*,*) buffNumber
                call send(buffNumber, i_buffer)
```

Implementation

- Two programs:
  - child_com.f

```
include 'solve.i'             ! Includes parameters used by oborg.i
include 'oborg.i'             ! Includes the common block used for testing
include 'f77_zmq.h'           ! Includes the ZMQ binding
include 'variables_client.i'  ! Includes a small common block holding the ZMQ related variables

integer, dimension(22) :: i_buffer ! Buffer for data tranmsmition
integer, dimension(20) :: i_array, i_array1, i_array2, i_array3 ! Generic arrays
integer                   i_rc ! if >= 0: the number of bytes sent or received, else: error
integer                   i, i_lbuf, c, i_sbuf, buffNumber, i_obsNumber

! Set up the communication.
address = 'tcp://localhost:55555'               ! Address to listen to.
context   = f77_zmq_ctx_new()                   ! Create new context
requester = f77_zmq_socket(context, ZMQ_REQ)    ! Use the REQ protocol
i_rc        = f77_zmq_connect(requester,address)! Declare the client as a requester

i_lbuf = size(i_buffer)
i_sbuf = i_lbuf * 4

! Clear the buffer
do i=1, i_lbuf
    i_buffer(i) = 0
enddo

! Fill the arrays with arbitrary numbers. The arrays hold two less elements
! since the buffer carries two control bits (buffer(1), buffer(2)).
do i=1, (i_lbuf-2)
    i_array1(i) = 21
    i_array2(i) = 22
    i_array3(i) = 23
enddo

! Invoke the program, l 121
call init(i_buffer, i_rc)

! Due to the nature of the program, a loop is used to provide the opportunity
! to test all functions. When used in production, a program would be written
! that makes use of the subroutines, with no need for human interaction.
do while (.true.)
    if ((i_buffer(1) .eq. 9) .and. (i_rc .ge. 0)) then
        print *, "All is good, please chose between the options below:"
        print *, "(1) Send"
        print *, "(2) Get"
        print *, "(3) Send Obs"
        print *, "(4) Get Obs"
        print *, "n/a (5) Get Size"
        print *, "n/a (6) Done"
        print *, "(7) Exit"
        read(*,*) c

        select case (c)
            case (1)
                print *, "Which array do you want parent to send (1-3)?"
                read(*,*) buffNumber
                call send(buffNumber, i_buffer)
```

Implementation

- Two programs:
  - child_com.f
  - parent_com.f

```
include 'solve.i'             ! Includes parameters used by oborg.i
include 'oborg.i'             ! Includes the common block used for testing
include 'f77_zmq.h'           ! Includes the ZMQ binding
include 'variables_client.i'  ! Includes a small common block holding the ZMQ related variables

integer, dimension(22) :: i_buffer ! Buffer for data tranmsmition
integer, dimension(20) :: i_array, i_array1, i_array2, i_array3 ! Generic arrays
integer                   i_rc ! if >= 0: the number of bytes sent or received, else: error
integer                   i, i_lbuf, c, i_sbuf, buffNumber, i_obsNumber

! Set up the communication.
address = 'tcp://localhost:55555'               ! Address to listen to.
context   = f77_zmq_ctx_new()                   ! Create new context
requester = f77_zmq_socket(context, ZMQ_REQ)    ! Use the REQ protocol
i_rc        = f77_zmq_connect(requester,address)! Declare the client as a requester

i_lbuf = size(i_buffer)
i_sbuf = i_lbuf * 4

! Clear the buffer
do i=1, i_lbuf
    i_buffer(i) = 0
enddo

! Fill the arrays with arbitrary numbers. The arrays hold two less elements
! since the buffer carries two control bits (buffer(1), buffer(2)).
do i=1, (i_lbuf-2)
    i_array1(i) = 21
    i_array2(i) = 22
    i_array3(i) = 23
enddo

! Invoke the program, l 121
call init(i_buffer, i_rc)

! Due to the nature of the program, a loop is used to provide the opportunity
! to test all functions. When used in production, a program would be written
! that makes use of the subroutines, with no need for human interaction.
do while (.true.)
    if ((i_buffer(1) .eq. 9) .and. (i_rc .ge. 0)) then
        print *, "All is good, please chose between the options below:"
        print *, "(1) Send"
        print *, "(2) Get"
        print *, "(3) Send Obs"
        print *, "(4) Get Obs"
        print *, "n/a (5) Get Size"
        print *, "n/a (6) Done"
        print *, "(7) Exit"
        read(*,*) c

        select case (c)
            case (1)
                print *, "Which array do you want parent to send (1-3)?"
                read(*,*) buffNumber
                call send(buffNumber, i_buffer)
```

Implementation

- Two programs:
  - child_com.f
  - parent_com.f
- Strategy:

```
include 'solve.i'             ! Includes parameters used by oborg.i
include 'oborg.i'             ! Includes the common block used for testing
include 'f77_zmq.h'           ! Includes the ZMQ binding
include 'variables_client.i'  ! Includes a small common block holding the ZMQ related variables

integer, dimension(22) :: i_buffer ! Buffer for data tranmsmition
integer, dimension(20) :: i_array, i_array1, i_array2, i_array3 ! Generic arrays
integer                   i_rc ! if >= 0: the number of bytes sent or received, else: error
integer                   i, i_lbuf, c, i_sbuf, buffNumber, i_obsNumber

! Set up the communication.
address = 'tcp://localhost:55555'               ! Address to listen to.
context   = f77_zmq_ctx_new()                   ! Create new context
requester = f77_zmq_socket(context, ZMQ_REQ)    ! Use the REQ protocol
i_rc        = f77_zmq_connect(requester,address)! Declare the client as a requester

i_lbuf = size(i_buffer)
i_sbuf = i_lbuf * 4

! Clear the buffer
do i=1, i_lbuf
    i_buffer(i) = 0
enddo

! Fill the arrays with arbitrary numbers. The arrays hold two less elements
! since the buffer carries two control bits (buffer(1), buffer(2)).
do i=1, (i_lbuf-2)
    i_array1(i) = 21
    i_array2(i) = 22
    i_array3(i) = 23
enddo

! Invoke the program, l 121
call init(i_buffer, i_rc)

! Due to the nature of the program, a loop is used to provide the opportunity
! to test all functions. When used in production, a program would be written
! that makes use of the subroutines, with no need for human interaction.
do while (.true.)
    if ((i_buffer(1) .eq. 9) .and. (i_rc .ge. 0)) then
        print *, "All is good, please chose between the options below:"
        print *, "(1) Send"
        print *, "(2) Get"
        print *, "(3) Send Obs"
        print *, "(4) Get Obs"
        print *, "n/a (5) Get Size"
        print *, "n/a (6) Done"
        print *, "(7) Exit"
        read(*,*) c

        select case (c)
            case (1)
                print *, "Which array do you want parent to send (1-3)?"
                read(*,*) buffNumber
                call send(buffNumber, i_buffer)
```

Implementation

- Two programs:
  - child_com.f
  - parent_com.f
- Strategy:
  - Open and close the connection as few times as possible

```
include 'solve.i'             ! Includes parameters used by oborg.i
include 'oborg.i'             ! Includes the common block used for testing
include 'f77_zmq.h'           ! Includes the ZMQ binding
include 'variables_client.i'  ! Includes a small common block holding the ZMQ related variables

integer, dimension(22) :: i_buffer ! Buffer for data tranmsmition
integer, dimension(20) :: i_array, i_array1, i_array2, i_array3 ! Generic arrays
integer                   i_rc ! if >= 0: the number of bytes sent or received, else: error
integer                   i, i_lbuf, c, i_sbuf, buffNumber, i_obsNumber

! Set up the communication.
address = 'tcp://localhost:55555'               ! Address to listen to.
context   = f77_zmq_ctx_new()                   ! Create new context
requester = f77_zmq_socket(context, ZMQ_REQ)    ! Use the REQ protocol
i_rc        = f77_zmq_connect(requester,address)! Declare the client as a requester

i_lbuf = size(i_buffer)
i_sbuf = i_lbuf * 4

! Clear the buffer
do i=1, i_lbuf
    i_buffer(i) = 0
enddo

! Fill the arrays with arbitrary numbers. The arrays hold two less elements
! since the buffer carries two control bits (buffer(1), buffer(2)).
do i=1, (i_lbuf-2)
    i_array1(i) = 21
    i_array2(i) = 22
    i_array3(i) = 23
enddo

! Invoke the program, l 121
call init(i_buffer, i_rc)

! Due to the nature of the program, a loop is used to provide the opportunity
! to test all functions. When used in production, a program would be written
! that makes use of the subroutines, with no need for human interaction.
do while (.true.)
    if ((i_buffer(1) .eq. 9) .and. (i_rc .ge. 0)) then
        print *, "All is good, please chose between the options below:"
        print *, "(1) Send"
        print *, "(2) Get"
        print *, "(3) Send Obs"
        print *, "(4) Get Obs"
        print *, "n/a (5) Get Size"
        print *, "n/a (6) Done"
        print *, "(7) Exit"
        read(*,*) c

        select case (c)
            case (1)
                print *, "Which array do you want parent to send (1-3)?"
                read(*,*) buffNumber
                call send(buffNumber, i_buffer)
```

Implementation

- Two programs:
  - child_com.f
  - parent_com.f
- Strategy:
  - Open and close the connection as few times as possible
  - Send as much data as possible each time

```
include 'solve.i'             ! Includes parameters used by oborg.i
include 'oborg.i'             ! Includes the common block used for testing
include 'f77_zmq.h'           ! Includes the ZMQ binding
include 'variables_client.i'  ! Includes a small common block holding the ZMQ related variables

integer, dimension(22) :: i_buffer ! Buffer for data tranmsmition
integer, dimension(20) :: i_array, i_array1, i_array2, i_array3 ! Generic arrays
integer                   i_rc ! if >= 0: the number of bytes sent or received, else: error
integer                   i, i_lbuf, c, i_sbuf, buffNumber, i_obsNumber

! Set up the communication.
address = 'tcp://localhost:55555'               ! Address to listen to.
context   = f77_zmq_ctx_new()                   ! Create new context
requester = f77_zmq_socket(context, ZMQ_REQ)    ! Use the REQ protocol
i_rc        = f77_zmq_connect(requester,address)! Declare the client as a requester

i_lbuf = size(i_buffer)
i_sbuf = i_lbuf * 4

! Clear the buffer
do i=1, i_lbuf
    i_buffer(i) = 0
enddo

! Fill the arrays with arbitrary numbers. The arrays hold two less elements
! since the buffer carries two control bits (buffer(1), buffer(2)).
do i=1, (i_lbuf-2)
    i_array1(i) = 21
    i_array2(i) = 22
    i_array3(i) = 23
enddo

! Invoke the program, l 121
call init(i_buffer, i_rc)

! Due to the nature of the program, a loop is used to provide the opportunity
! to test all functions. When used in production, a program would be written
! that makes use of the subroutines, with no need for human interaction.
do while (.true.)
    if ((i_buffer(1) .eq. 9) .and. (i_rc .ge. 0)) then
        print *, "All is good, please chose between the options below:"
        print *, "(1) Send"
        print *, "(2) Get"
        print *, "(3) Send Obs"
        print *, "(4) Get Obs"
        print *, "n/a (5) Get Size"
        print *, "n/a (6) Done"
        print *, "(7) Exit"
        read(*,*) c

        select case (c)
            case (1)
                print *, "Which array do you want parent to send (1-3)?"
                read(*,*) buffNumber
                call send(buffNumber, i_buffer)
```

Results for Project One

Results for Project One

Running userpartials for $\sim$200 sessions.

Results for Project One

Running userpartials for $\sim$200 sessions.

- Old version of globl/solve: 8 minutes 47 seconds

Results for Project One

Running userpartials for ∼200 sessions.

- Old version of globl/solve: 8 minutes 47 seconds
- New version of globl/solve: 9 minutes 5 seconds

Results for Project One

Running userpartials for $\sim$1400 sessions.

Results for Project One

Running userpartials for $\sim$1400 sessions.

- Old version of globl/solve: 42 minutes 20 seconds

Results for Project One

Running userpartials for $\sim$1400 sessions.

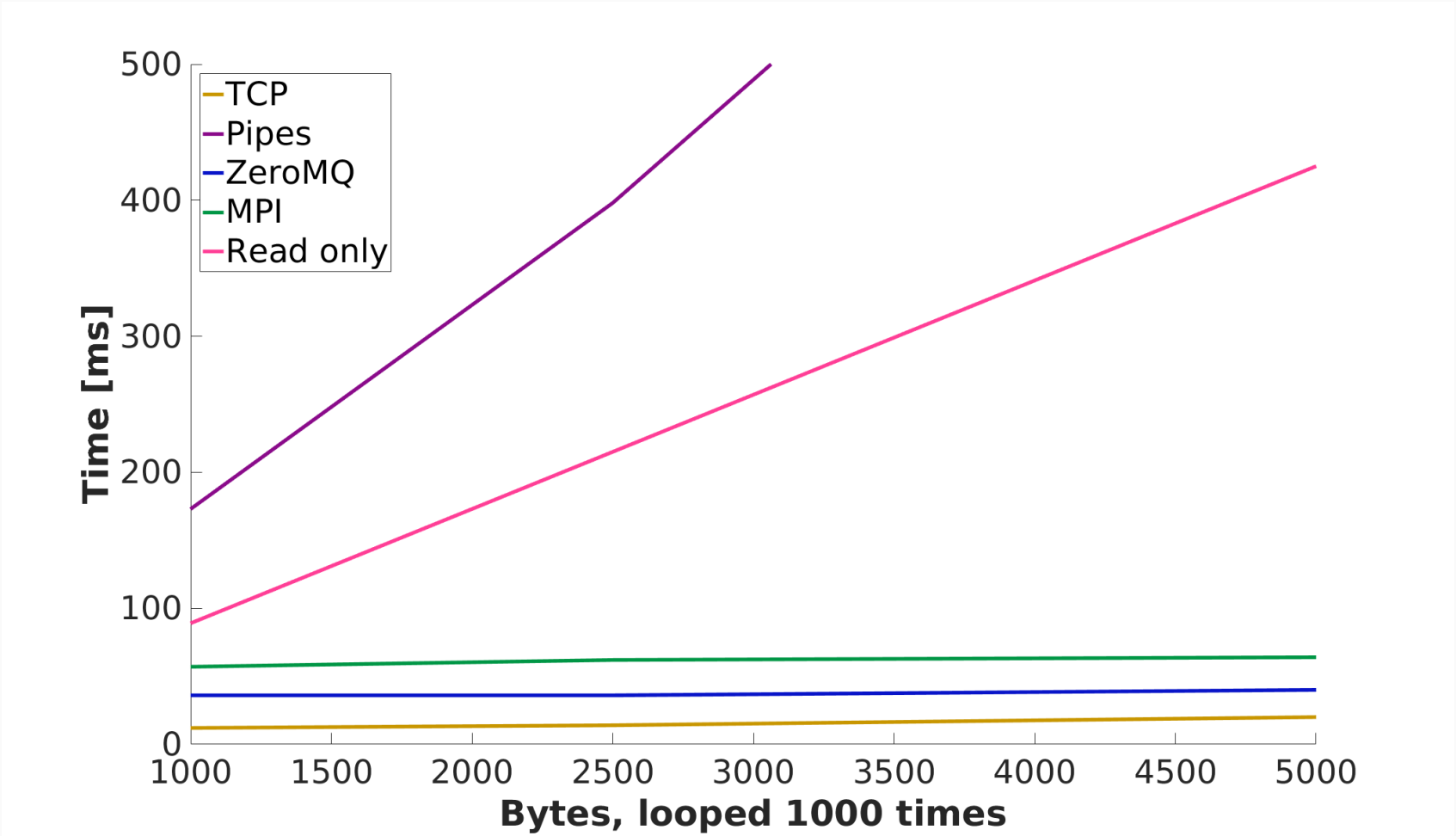
- Old version of globl/solve: 42 minutes 20 seconds
- New version of globl/solve: 46 minutes 34 seconds

Results for Project One

Why?

Results for Project One

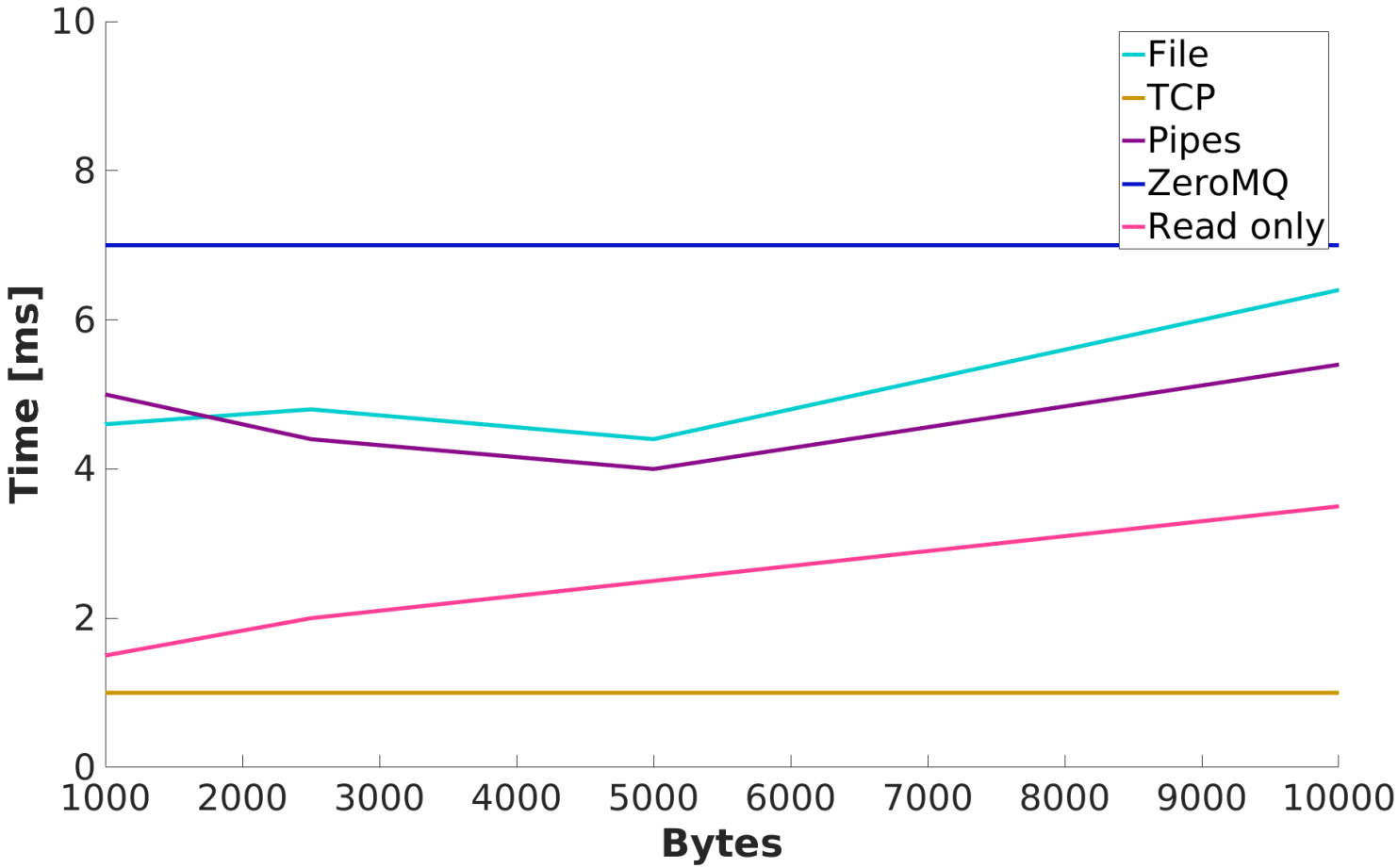
Globl/solve does not *read and write* a lot, it **reads a lot,**



but read only is still slower...

Results for Project One

Globl/solve does not read as much as we thought.



Results for Project One

Maybe not all in vain:

Results for Project One

Maybe not all in vain:

- ZeroMQ can be used for concurrency (parallel programming)

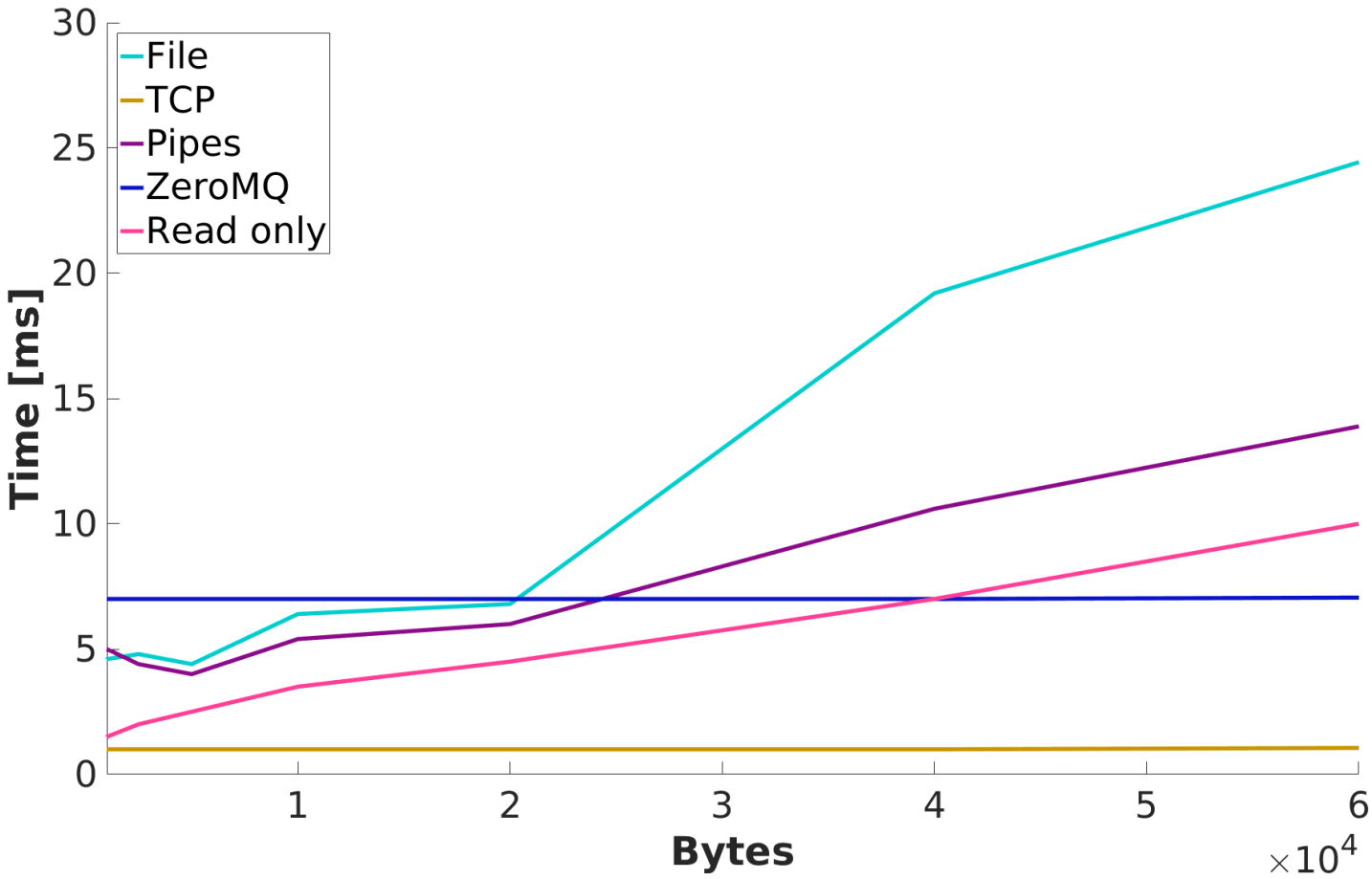
Results for Project One

Maybe not all in vain:

- ZeroMQ can be used for concurrency (parallel programming)
- We only calculate user partials, what about user programs?

Results for Project One

What about even larger files?



Future studies

- Remove the writes that are no longer needed.

Future studies

- Remove the writes that are no longer needed.
- TCP should have been the way to go?

Future studies

- Remove the writes that are no longer needed.
- TCP should have been the way to go?
- Posix Shared Memory?

Future studies

- Remove the writes that are no longer needed.
- TCP should have been the way to go?
- Posix Shared Memory?
  - Conversion between C and Fortran was non trivial.

Our Second Project

Problem

Mario noticed that we were getting error messages about antennas arriving late on source.

Problem Description

The model used for determining when an antenna acquires a source is a little bit off. Sked will say that an antenna should be on target at time T but the logfiles tells us that the antenna actually was on source at $T \pm c$.

Extract the Data

```
Observation listing from file ./r1747.skd  for experiment R1747---
Source      Start      AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL  AZ  EL
name     yyddd-hhmmss     Ft      Hh      Hb      Is      Ke      Ny      On      Kv      Sh      Ts      Wz      Yg
0834-201 16186-170000| 211 71| 257 20|       |       |       |       |       |       |       |       |       |       |
3C418    16186-170000|       |       |       | 341 74| 360 24| 410 45| 398 29|       | 378 68| 341 74| 397 23| 370  8|
2052-474 16186-170332|       |       | 239 80| 543  6| 536 57|       |       | 535  6| 530 11| 183  7|       | 505 66|
0808+019 16186-170341| 287 72|       |       |       |       |       | 612 13|       |       |       | 255 15|       |
1717+178 16186-170709|       |  55 23|       | 628 34|       | 468 22|       |       | 620 51| 268 34| 455 28|       |
1923+210 16186-170918|       |       | 325 18| 607 61| 329 49|       |       |       | 584 77| 247 61|       |       |
0955+476 16186-171036| 366 38|       |       |       |       | 593 55| 627 63|       |       |       | 283 61|       |
2008-159 16186-171235|       |       |       |       |       |       |       |       | 539 43| 202 35|       | 385 75|
0059+581 16186-171315|       |       |       |       |       |       |       | 400 42|       |       | 359 18|       |
1908-201 16186-171450|       |       | 290 49| 576 25| 253 65|       |       |       |       |       |       |       |
2059+034 16186-171504|       |       |       |       |       |       |       | 528 57| 514 60| 192 57|       |       |
1144+402 16186-171552| 392 37| 334 17|       |       |       | 561 50| 580 69|       |       |       | 601 74|       |
1759-396 16186-171642|       |       |       |       | 228 44|       |       |       |       |       |       | 237 66|
```

```
2016.186.17:01:31.00:source=2052-474,205616.36,-471447.6,2000.0,neutral
2016.186.17:01:32.00#flagr#flagr/antenna,new-source
2016.186.17:01:32.00:sx8m11a=1
2016.186.17:01:32.00:!2016.186.17:03:22
2016.186.17:01:52.00#trakl#Source acquired
```

Parse the Data

- Correlate the log data with the Sked data.

Parse the Data

- Correlate the log data with the Sked data.
- Least Square Fit

Result for Project Two

Sessions and stations

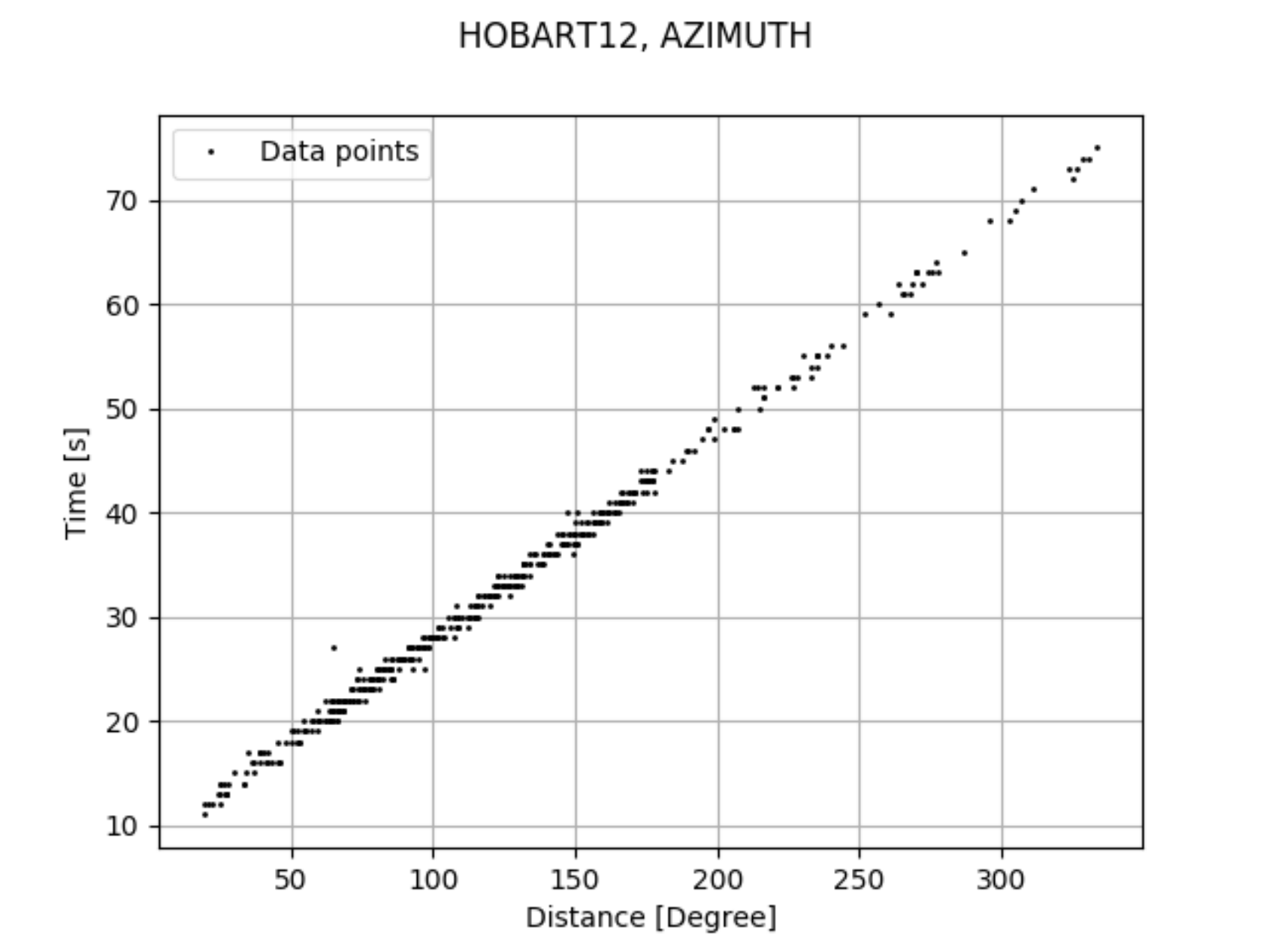
Sessions used:

- Regular Monday sessions
  - r1721 - r1752
- Regular Thursday sessions
  - r4721 - r4752

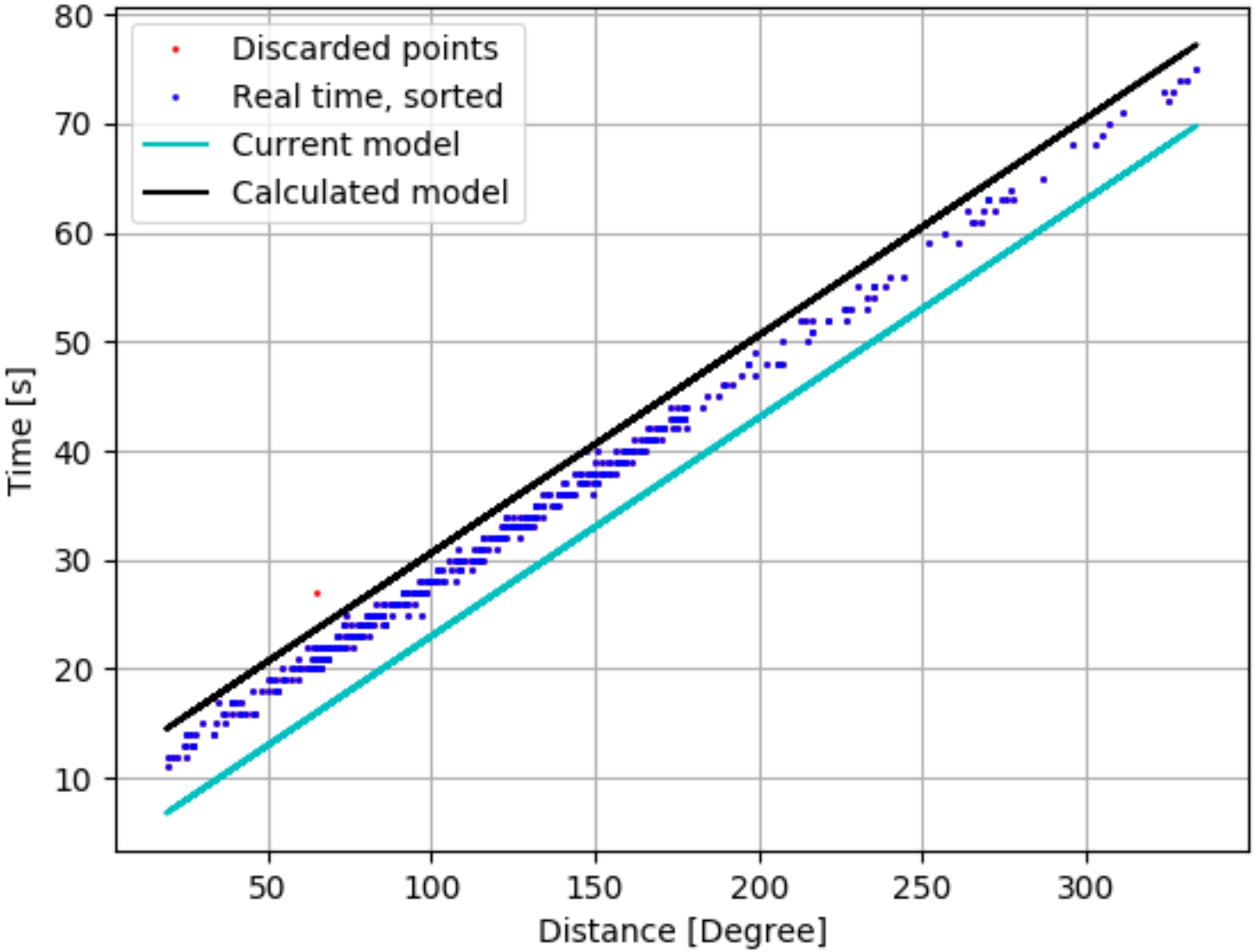
Stations in sessions:

- Badary
- Fortleza
- Hart15m
- Hobart12
- Ishioka
- Kashim34
- Kath12m
- Kokee
- Matera
- Medicina
- Nyales20
- Onsala60
- Raegyeb
- Sejong
- Seshan25
- Svetloe
- Tsukub32
- Urumqi
- Yarra12m
- Yebes40m
- Zelenchk

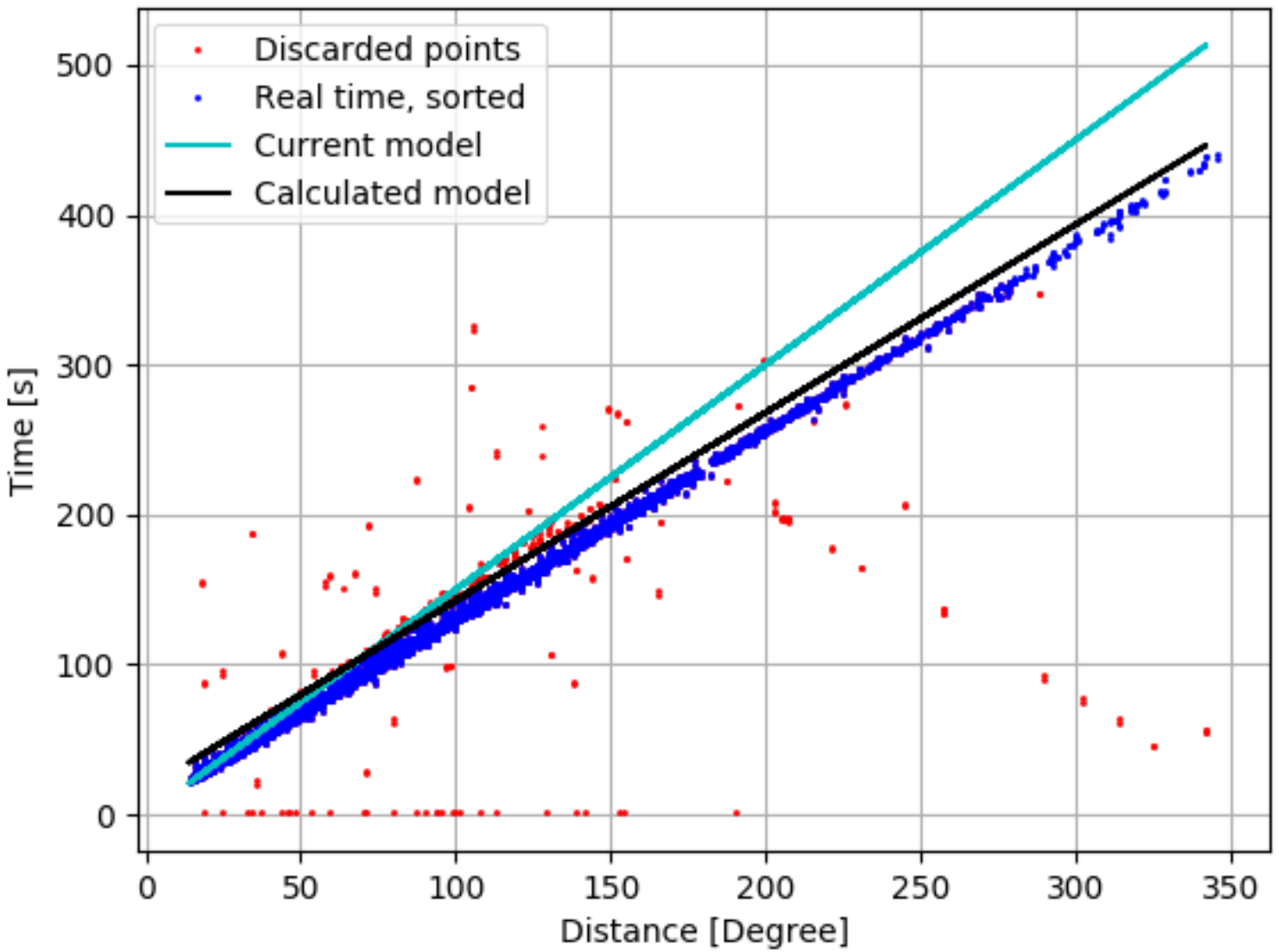
Extracted data points



Calculated model



Calculated model



Result for Project two

Antenna	Az_{Old} [deg/min]	Offset [s]	Az_{New} [deg/min]	Offset [s]	ΔAzimuth [deg/min]	ΔOffset [s]
Badary	60	40.0	65	40.2	+5	+.2
Fortleza	40	0.0	48	16.2	+8	+16.2
Hart15m	120	3.0	124	10.5	+4	+7.5
Hobart12	300	3.0	301	10.9	+1	+7.9
Ishioka	720	10.0	728	9.5	+8	-.5
Kashim34	45	0.0	49	13.8	+4	+13.8
Kath12m	300	3.0	300	11.0	0	+8.0
Kokee	117	15.0	127	13.4	+10	-1.6
Matera	90	20.0	105	37.3	+15	+17.3
Medicina	48	0.0	48	13.7	0	+13.7
Nyales20	120	9.0	137	11.7	+17	+2.7

Result for Project two

Antenna	Az_{Old} [deg/min]	Offset [s]	Az_{New} [deg/min]	Offset [s]	ΔAzimuth [deg/min]	ΔOffset [s]
Onsala60	144	20	184	19.0	+40	-1.0
Raegyeb	720	3	718	12.8	-2	+9.8
Sejong	300	0	312	22.7	+12	+22.7
Seshan25	60	0	56	10.1	-4	+10.1
Svetloe	60	40	67	34.9	+20	-5.1
Tsukub32	180	14	195	15.6	+15	+1.6
Urumqi	60	0	61	11.3	+1	+11.3
Yarra12m	300	3	300	10.0	0	+7.0
Yebes40m	60	10	121	19.6	+61	+9.6
Zelenchk	60	40	62	50.2	+2	+10.2

Result for Project two

Antenna	El_{Old} [deg/min]	Offset [s]	El_{New} [deg/min]	Offset [s]	ΔElevation [deg/min]	ΔOffset [s]
Badary	30	40.0	35	34.4	+5	-5.6
Fortleza	20	0.0	20	12.4	0	+12.4
Hart15m	60	3.0	58	8.7	-2	+5.7
Hobart12	75	3.0	76	9.1	+1	+6.1
Ishioka	360	10.0	366	9.2	+6	-.8
Kashim34	40	0.0	41	11.6	+1	+11.6
Kath12m	75	3.0	75	9.0	0	+6.0
Kokee	117	15.0	138	11.1	+21	-3.9
Matera	100	20.0	137	34.2	+37	+14.2
Medicina	30	0.0	30	11.0	0	+11.0
Nyales20	120	9.0	138	8.2	+18	-.8

Result for Project two

Antenna	El_{Old} [deg/min]	Offset [s]	El_{New} [deg/min]	Offset [s]	ΔElevation [deg/min]	ΔOffset [s]
Onsala60	60	20	66	15.8	+6	-4.2
Raegyeb	360	3	369	12.8	+9	+9.8
Sejong	300	0	221	23.8	-79	+23.8
Seshan25	30	0	32	9.1	+2	+9.1
Svetloe	30	40	26	24.5	-4	-15.5
Tsukub32	180	14	190	16.1	+10	+2.1
Urumqi	30	0	32	9.9	+2	+9.9
Yarra12m	75	3	75	9.0	0	+6.0
Yebes40m	60	10	60	18.0	0	+8.0
Zelenchk	30	40	32	43.3	+2	+3.3

Mirrored data points

- Some stations has a line of additional data points mirrored in the 180 degree line.

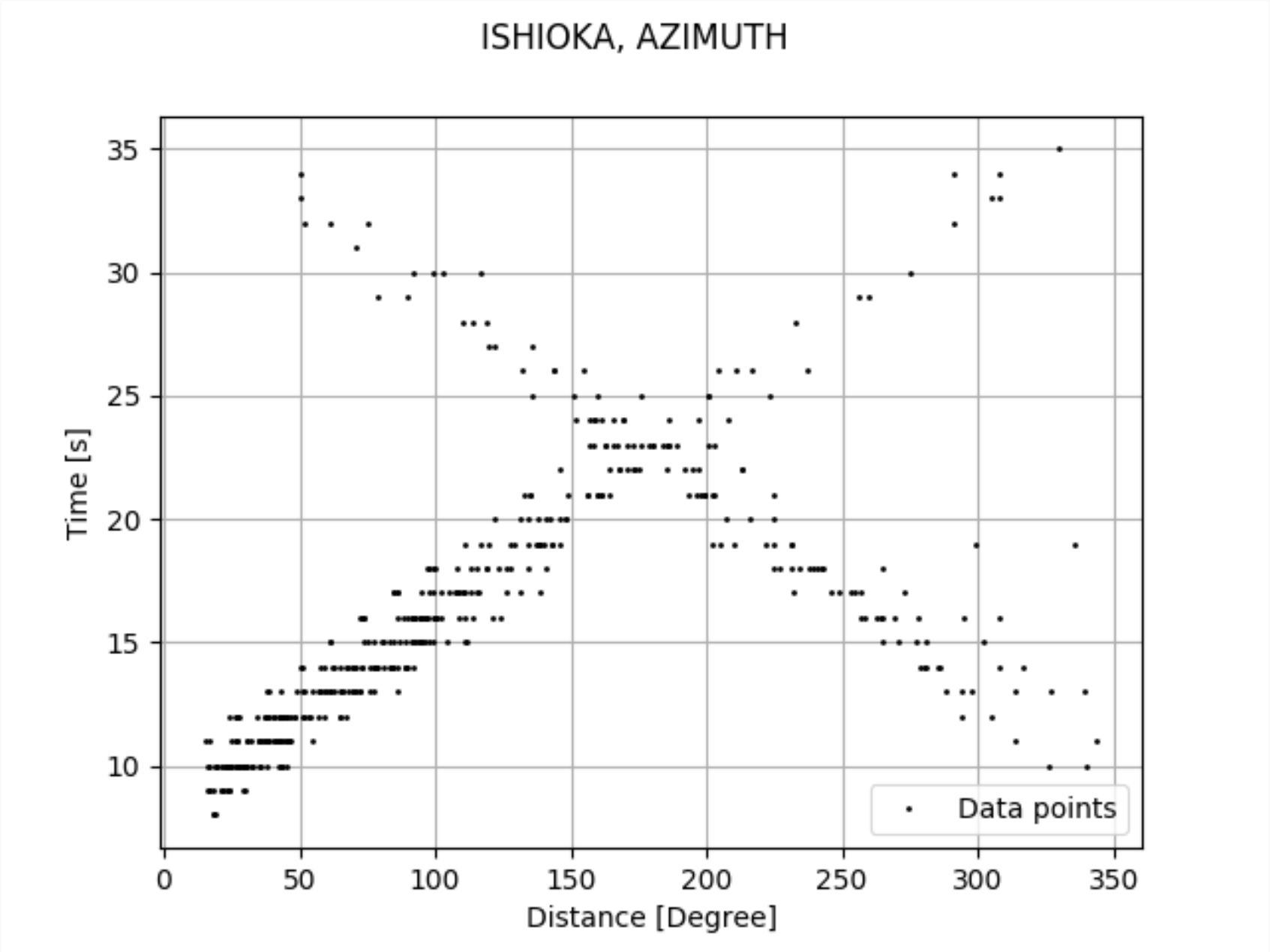
Mirrored data points

- Some stations has a line of additional data points mirrored in the 180 degree line.
- Probably due to cable wrap.

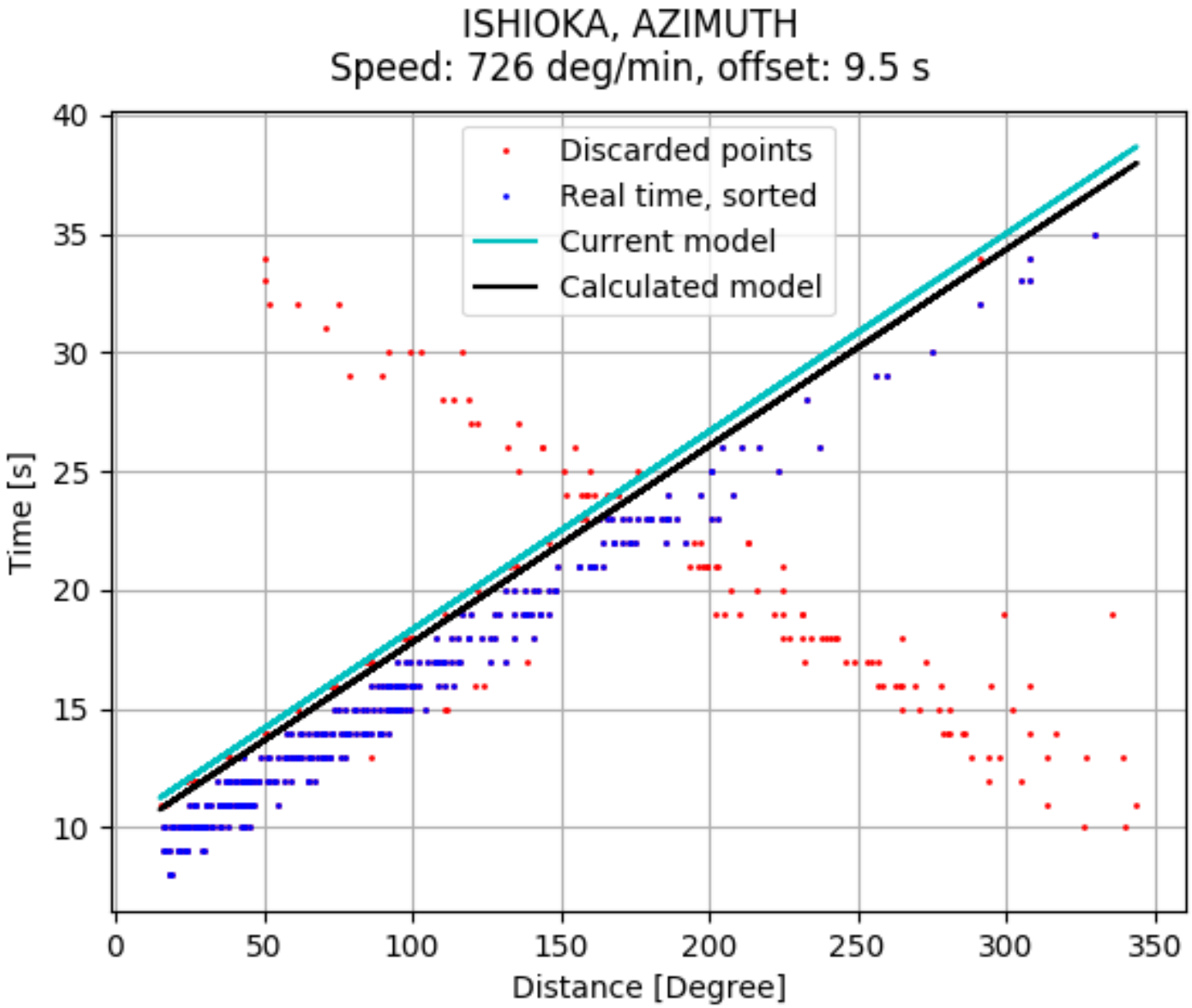
Mirrored data points

- Some stations has a line of additional data points mirrored in the 180 degree line.
- Probably due to cable wrap.
- Stations: Ishioka, Kath12m, Sejong and Yarra12m

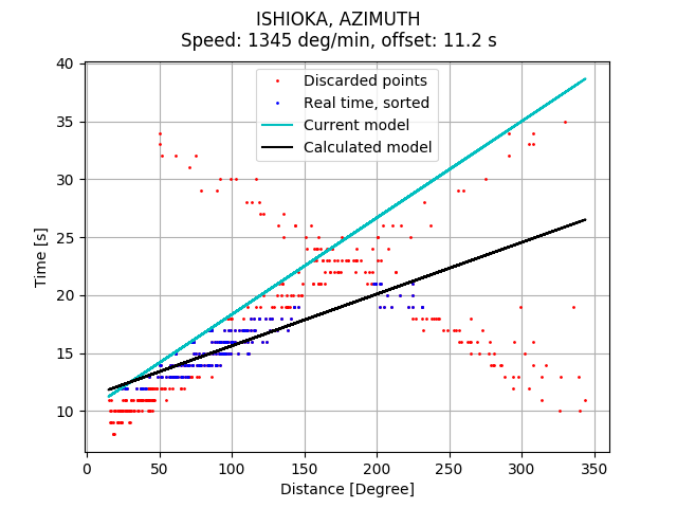
Extracted data points

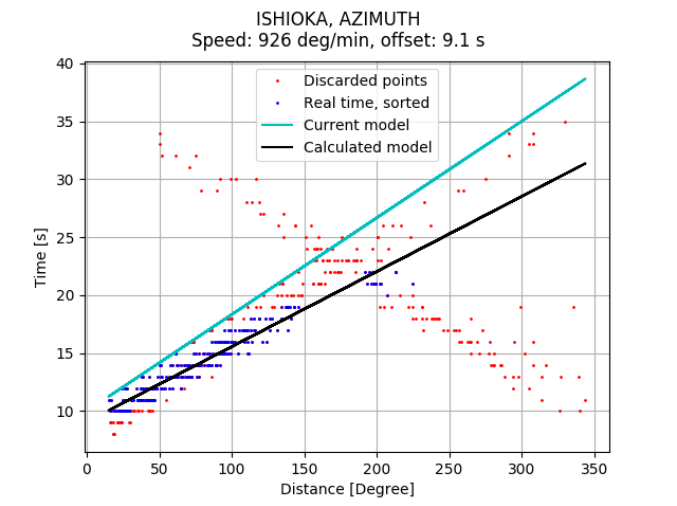


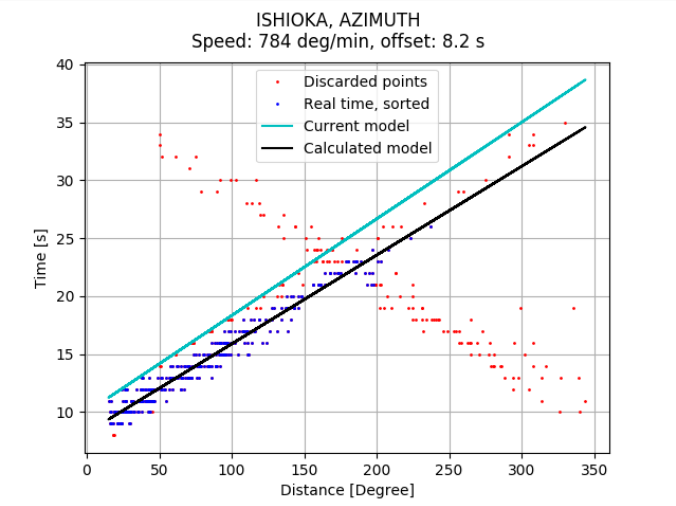
Calculated model

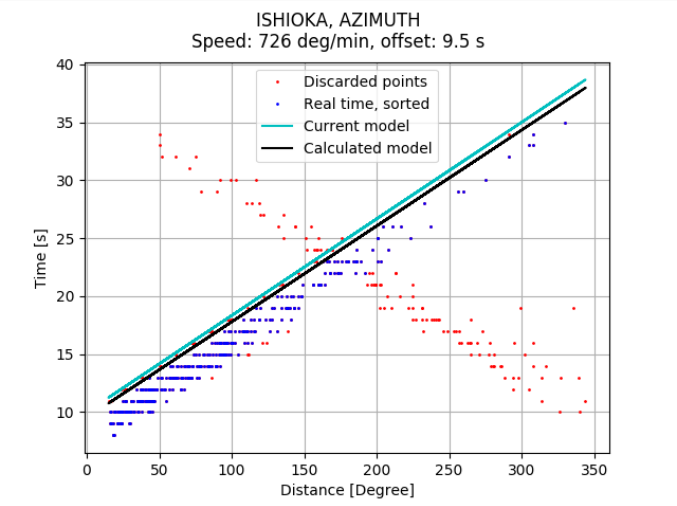


Iteration of Least Square Fit

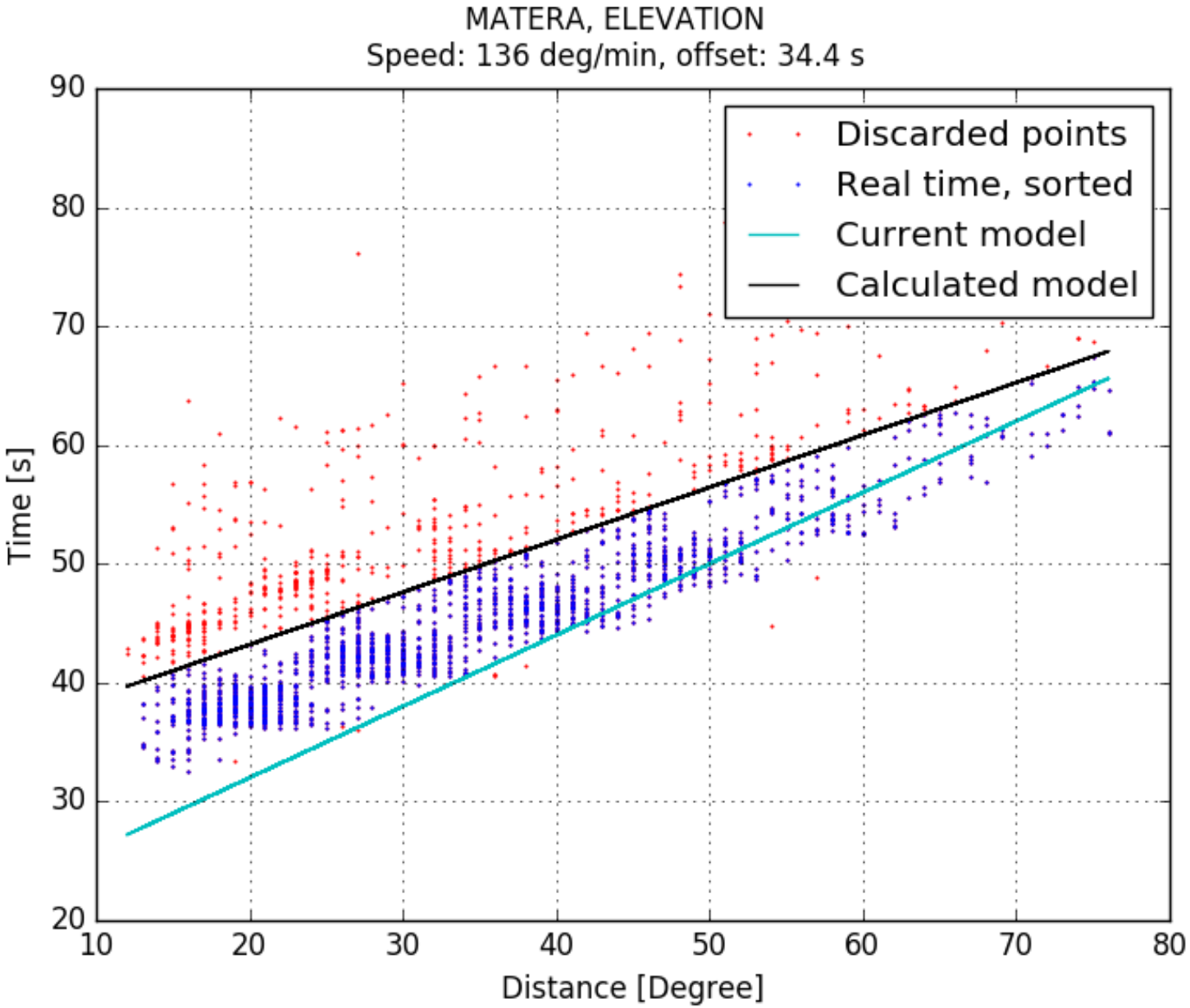








Additional Issues



Future studies

- Fix the bug in Sked.

Future studies

- Fix the bug in Sked.
- Set a more intelligent threshold.

Questions?

Questions?