Implementation of neighboring communication in QWS

Issaku Kanamori (RIKEN)

August 4th, 2020

Asia-Pacific Symposium for Lattice Field Theory (APLAT 2020)

RIKEN
Center for
Computational Science

Outline

1. Introduction specification of Fugaku
2. Algorithm and Implementation double buffering
3. As a communication library benchmark with a 2-dim Poisson equation
4. Summary and Outlooks

Outline

1. Introduction   specification of Fugaku
2. Algorithm and Implementation   double buffering
3. As a communication library   benchmark with a 2-dim Poisson equation
4. Summary and Outlooks

Acknowledgments

This talk is based on discussion with the LQCD codesign team in flagship 2020 project:

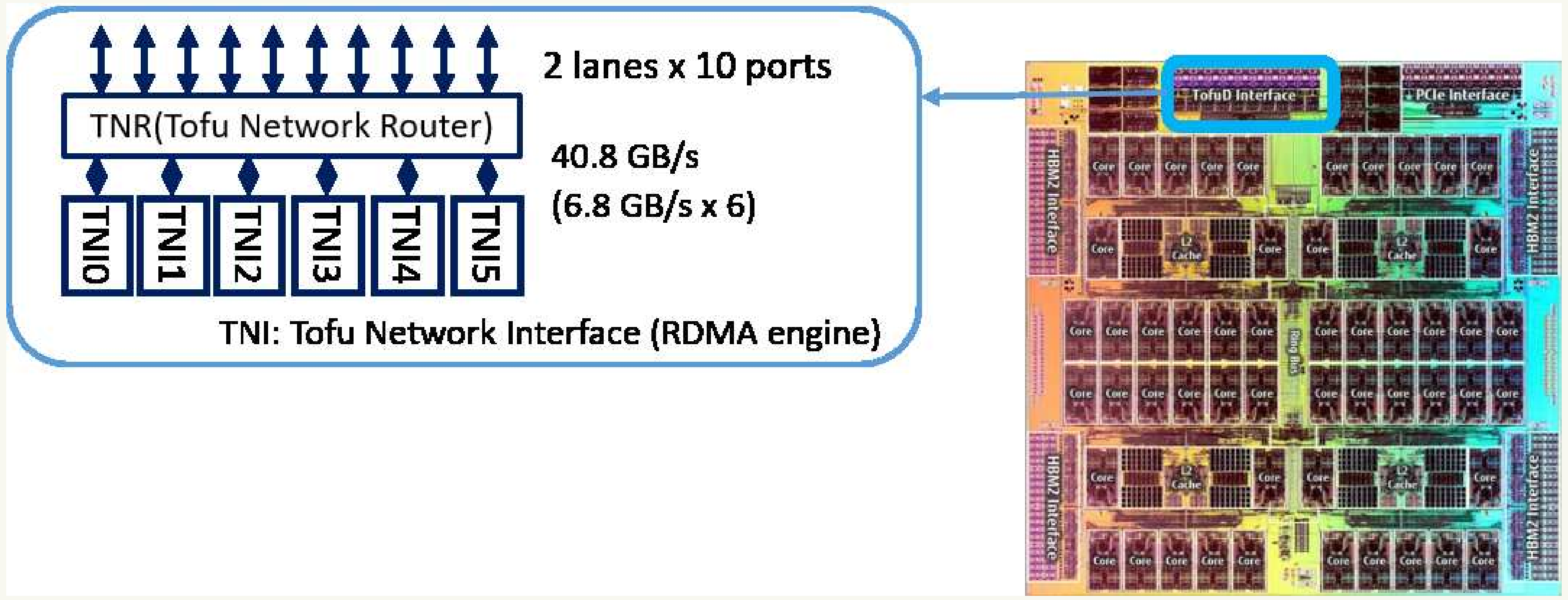
RIKEN) Y.Nakamura, I.K, K.Nitadori, M.Tsuji
Fujitsu) I.Miyoshi, Y.Mukai, T.Nishiki
Hiroshima) K.-I.Ishikawa
KEK) H.Matsufuru

I.K. also thanks the MEXT as "Program for Promoting Researches on the Supercomputer Fugaku" (Simulation for basic science: from fundamental laws of particles to creation of nuclei) and JICFuS.

Disclaimer

The software used for the evaluation, such as the compiler, is still under development and its performance, which is obtained by “performance estimation tool” and even actual execution on a prototype machine, may be different when the supercomputer Fugaku starts is operation.

Feature of Fugaku: TofuD interconnect



https://postk-web.r-ccs.riken.jp/spec.html

Flops/node: 3TFlops[double]
($\times$23 of K-computer)
injection BW/node 40.8GB/s
(only $\times$2 of K-computer)
communication is important

- 6D torus/mesh network with 10 nearest neighbors (small 3d “torus” $[2 \times 3 \times 2]$) $\times$ (large 2d torus $\times$ 1d mesh),
- Each node ($\neq$ process) can send data to 6 different directions simultaneously [QCD has 8 directions]
- Latency: 0.49μs, $>$ 90% efficiency for the nearest neighbor put

cf. https://www.fujitsu.com/global/Images/the-tofu-interconnect-d-for-supercomputer-fugaku.pdf

- interface for TofuD: uTofu $\Leftarrow$ QWS uses uTofu for neighboring comm.

keywords: TNI: Tofu Network Interface (RDMA engine) RDMA: Remote Direct Memory Access
uTofu: Low Level Communcation API for TofuD

QWS

QCD Wide Simd Library: see Y.Nakamura's talk

- Clover solver designed for Fugaku

  also runs on other architectures s.t. intel
- `https://github.com/RIKEN-LQCD/qws`

Algorithm and Implementation

Double Buffering

single buffering

- sender: 1 send buffer
- receiver: 1 recv. buffer

doulbe buffering

- sender: 1 send buffer
- receiver: 2 recv. buffers, used alternatingly

Double Buffering

single buffering

- sender: 1 send buffer
- receiver: 1 recv. buffer

doulbe buffering

- sender: 1 send buffer
- receiver: 2 recv. buffers, used alternatingly

- smaller overhead: no need to check if the recv. buffer is ready

  the other buffer is always available
- robuster against load imbalance

  no need to wait till the recv. buffer becomes available

Implementation: using uTofu interface

- smaller latency than calling MPI
- we use RDMA put   write directly to the memory in the target process
- can (or must) tune the TNI assignment

6 TNIs: 6 simultaneous RDMA put to different directions

the load to each TNI should be balanced

  - the boundary size

    depends on the direction: the local volume can be a hyper-rectangular
  - the rank map

    6-dim Tofu coordinate → 4-dim QCD proc. coordinate → 1-dim MPI rank id

a proper rank map is important

the logical “neighbor” may not be a physical neighbor

Implementation in QWS

`rdma_utofu_comlib.c`: wrapper functions for calling uTofu

```
rdma_comlib_data buf;

// buf is to send data (of size) to dst_rank and receive from rcv_rank using TNI of tni_id
// cf. MPI_Recv_init and MPI_Send_init
rdma_comlib_new(&buf, &tni_id, &dst_rank, &rcv_rank, &size);
// start sending with RDMA put
rdma_comlib_isendrecv(&buf);
rdma_comlib_irecv_check(&buf); // cf. MPI_Wait for receiving
rdma_comlib_isend_check(&buf); // cf. MPI_Wait for sending
```

`rdma_comlib_2buf`: a class for double buffering

built with functions in `rdma_utofu_comlib.c`

`qws_xbound_rdma.cc`: communication routines in QWS

(uTofu RDMA version)

```
rdma_comlib_2buf buff_rdma[8];
...
// initialize communication in x−direction
buff_rdma[0].init(tni_list[0], pxb, pxf,size);
buff_rdma[1].init(tni_list[1], pxf, pxb,size);

buff_rdma[req].isendrecv(); // in void xbound(int req, int prec)
buff_rdma[req].irecv_check(); // in void xbound_wait(int req, int prec)
```

Some details of the implementation

with a proper data alignment and suitable flags to uTofu interface

sender

- uTofu put is thread parallelized

receiver

- we monitor the last byte of the buffer to check the data has arrived
- received data goes directly to the cache   cache injection

As a Communication Library

Benchmark with a 2-dim Poisson equation on Fugaku

Target System

base: `http://theo.phys.sci.hiroshima-u.ac.jp/~ishikawa/APL9WG/stencil_double_buffering_mpi-1.0.tar.gz`

$Mx = b$ with

$$(Mx)(i,j) = \underbrace{(4+m^2)x(i,j)}_{\equiv Dx} \underbrace{-x(i+1,j) - x(i-1,j) - x(i,j+1) - x(i,j-1)}_{\equiv Hx}$$

$$\xrightarrow{\text{cont. limit}} (m^2 - \partial^2)x$$

Jacobi method

$$x^{(k)} \longrightarrow x^{(k+1)} = D^{-1}(b - Hx^{(k)})$$

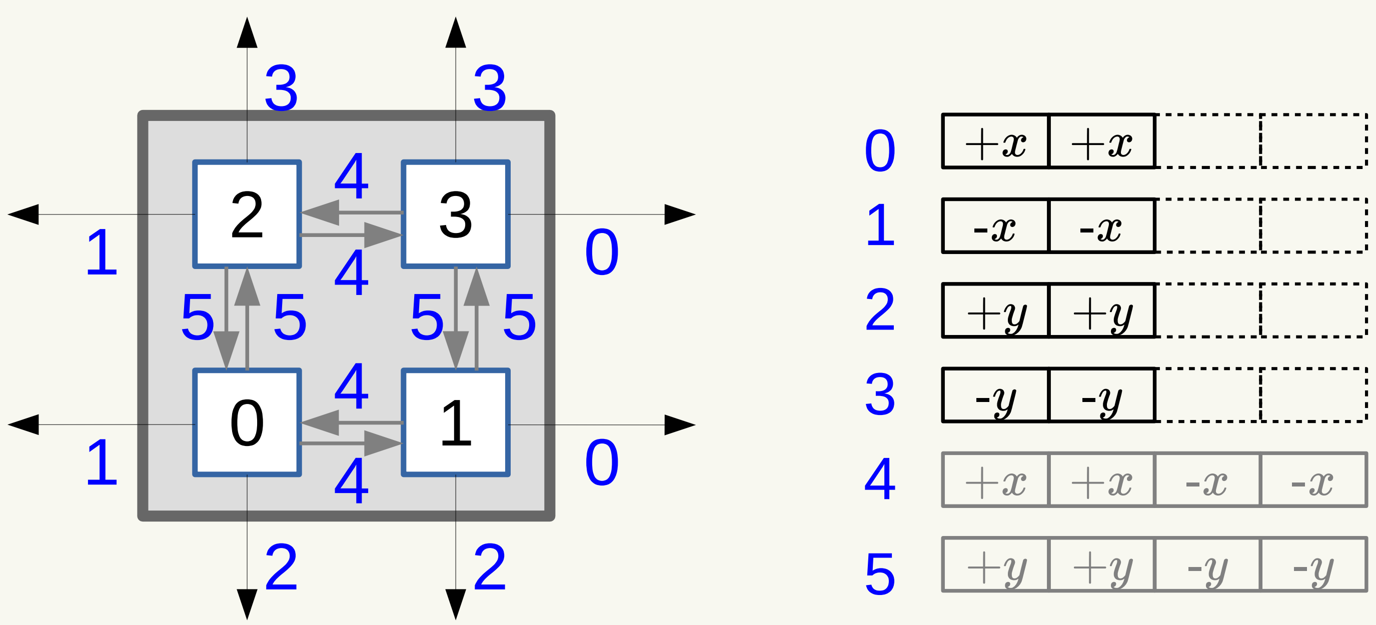
Only the hopping H contains the communication

- fixed number of iterations: 1000
- calculation of the residual norm (`MPI_Allreduce`) in every 10 iter.
- local lattice size: 60×60
- communication buffer (60 elements for each direction) in enlarged by 1–8192 (+ a flag as the end of buffer + alignment)

Theoretical Bandwidth for uTofu Communication

4 MPI ranks/node (2×2 ranks in each node)

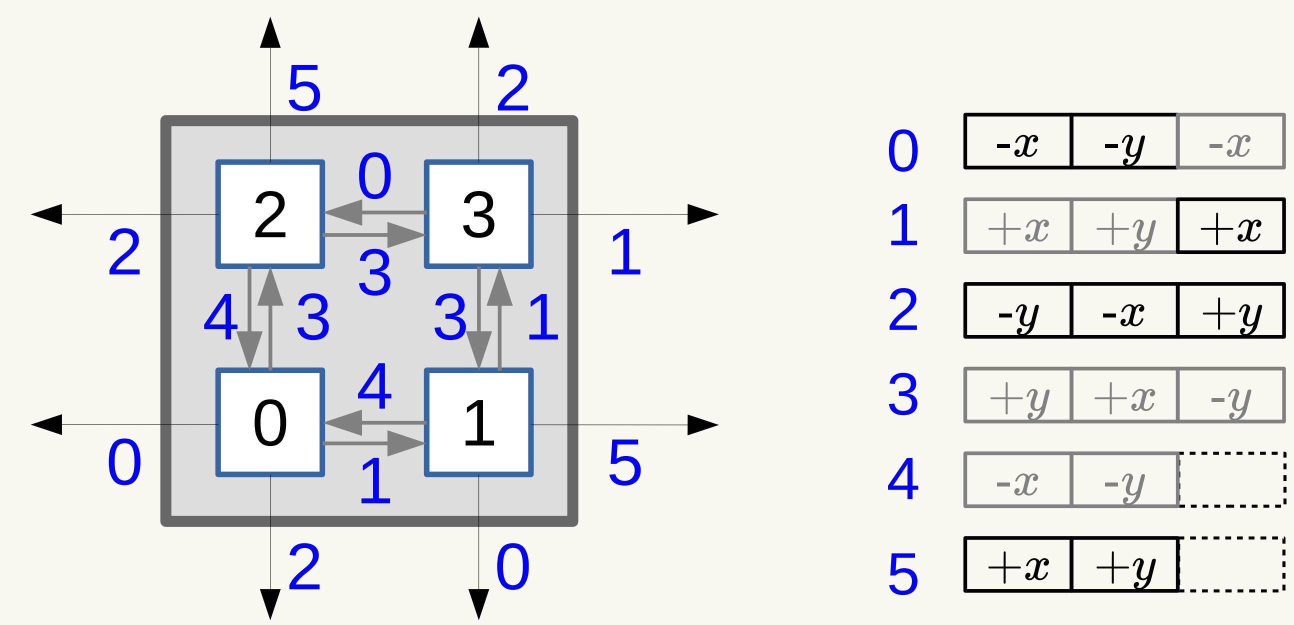
pattern 1



effective band width:

$$40.8 \times \frac{16}{24} = 27.2\text{GB/s}$$

pattern 2 (round robin)

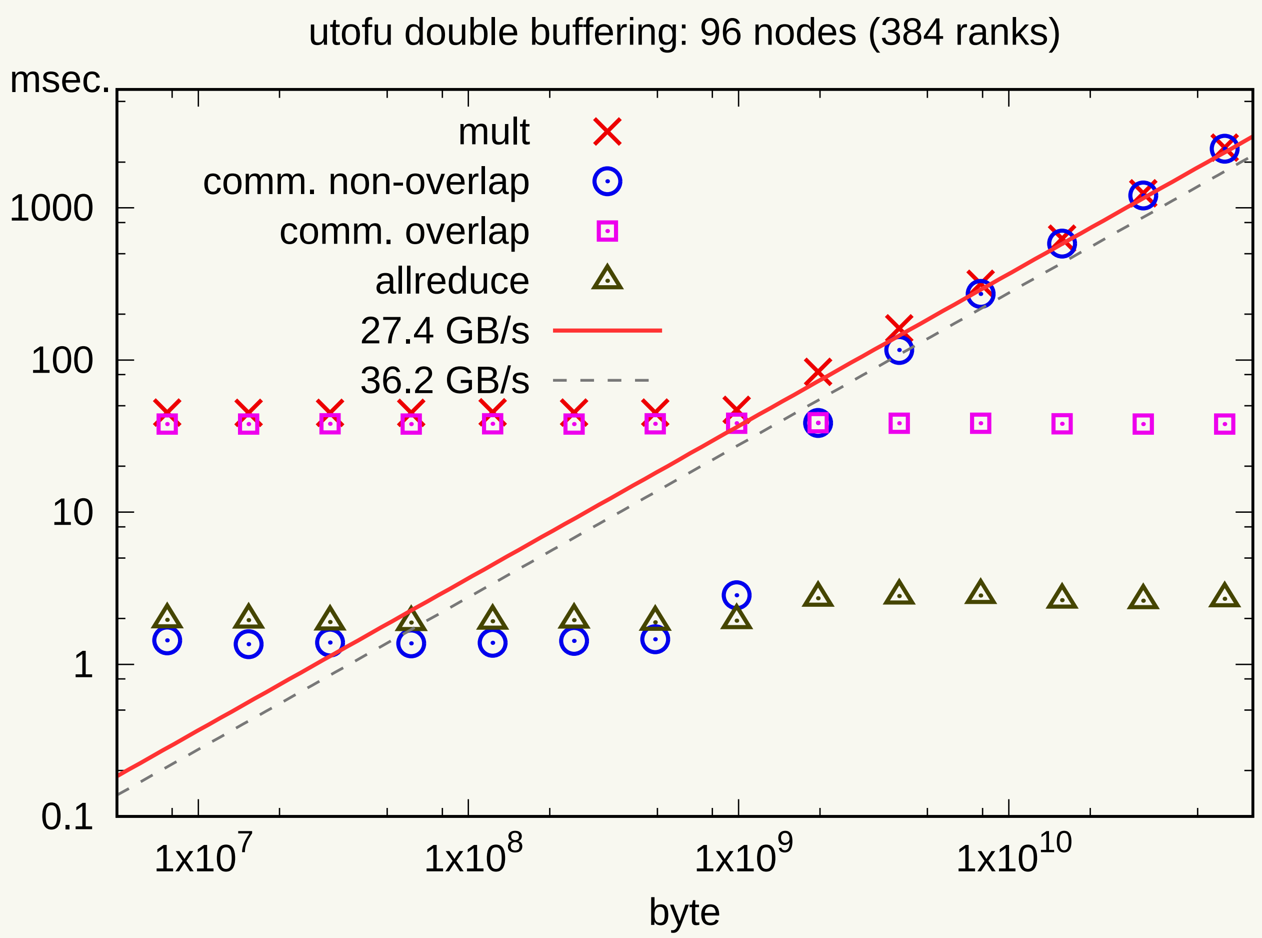


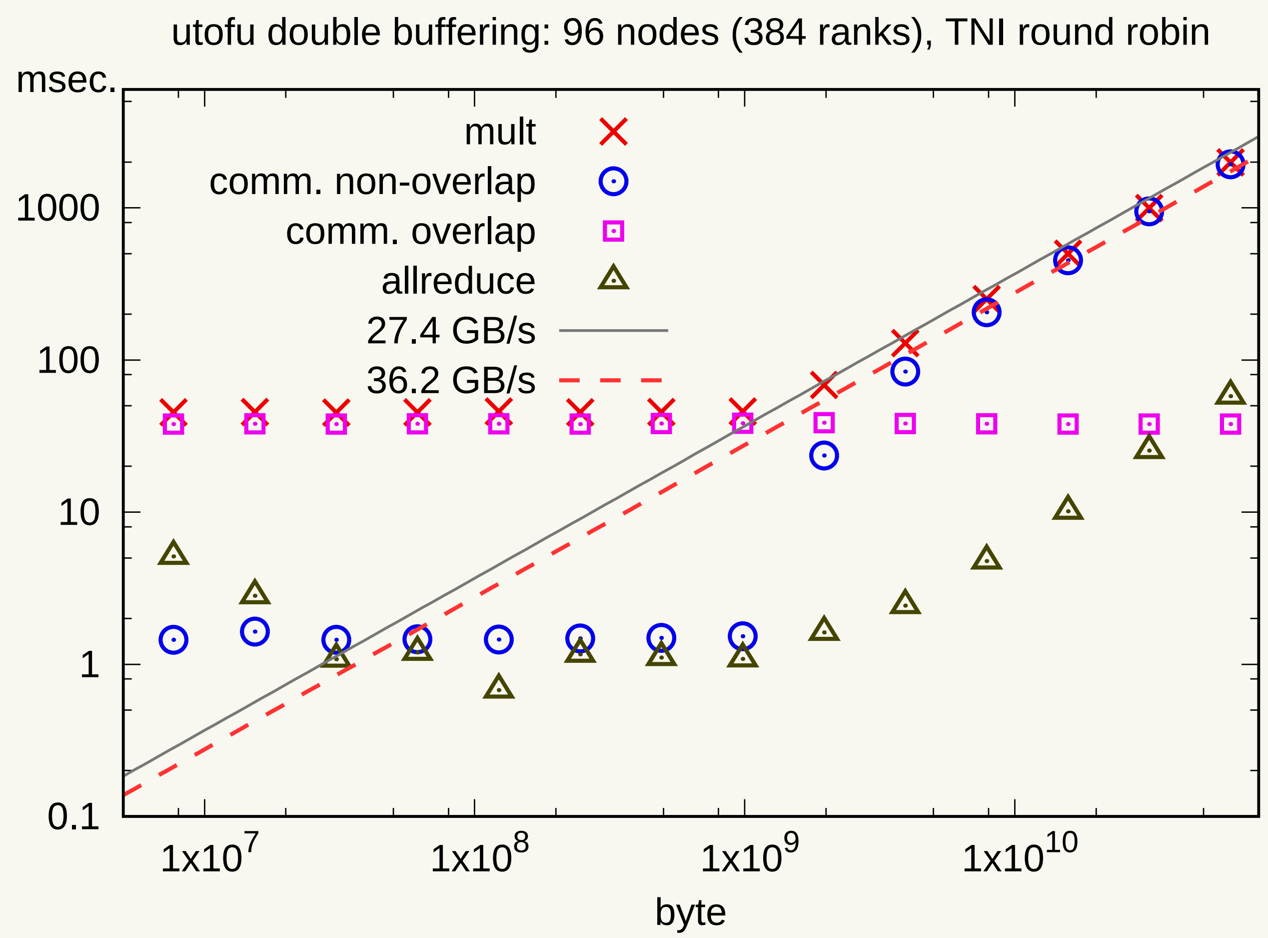
effective band width:

$$40.8 \times \frac{16}{18} = 36.3\text{GB/s}$$

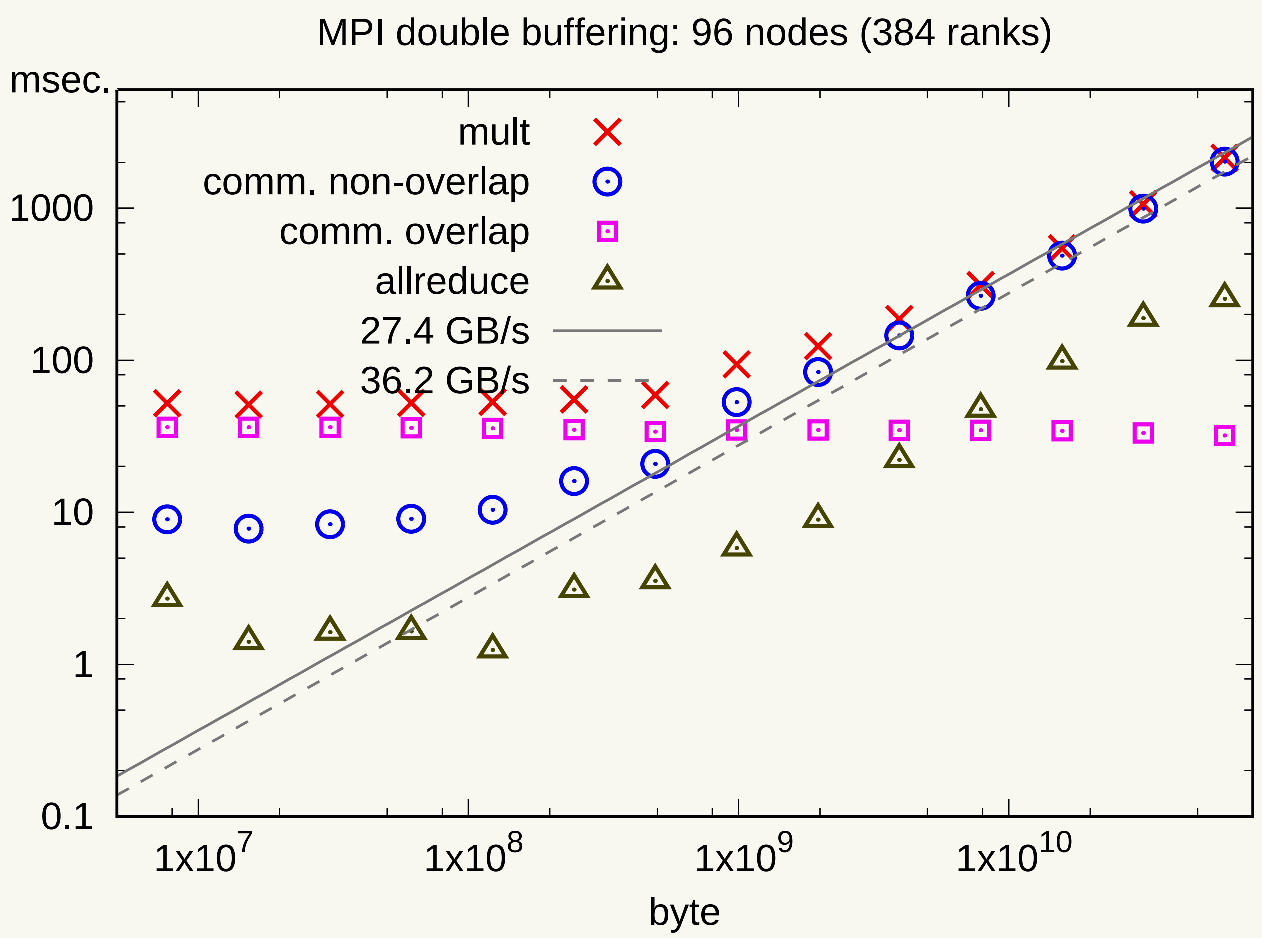
- a proper TNI assignment is important to maximize the effective BW
- if (boundary size for x)$\neq$(boundary size for y), we can enjoy more games with TNI assignment
- (TNI assignments for MPI communication is unclear)

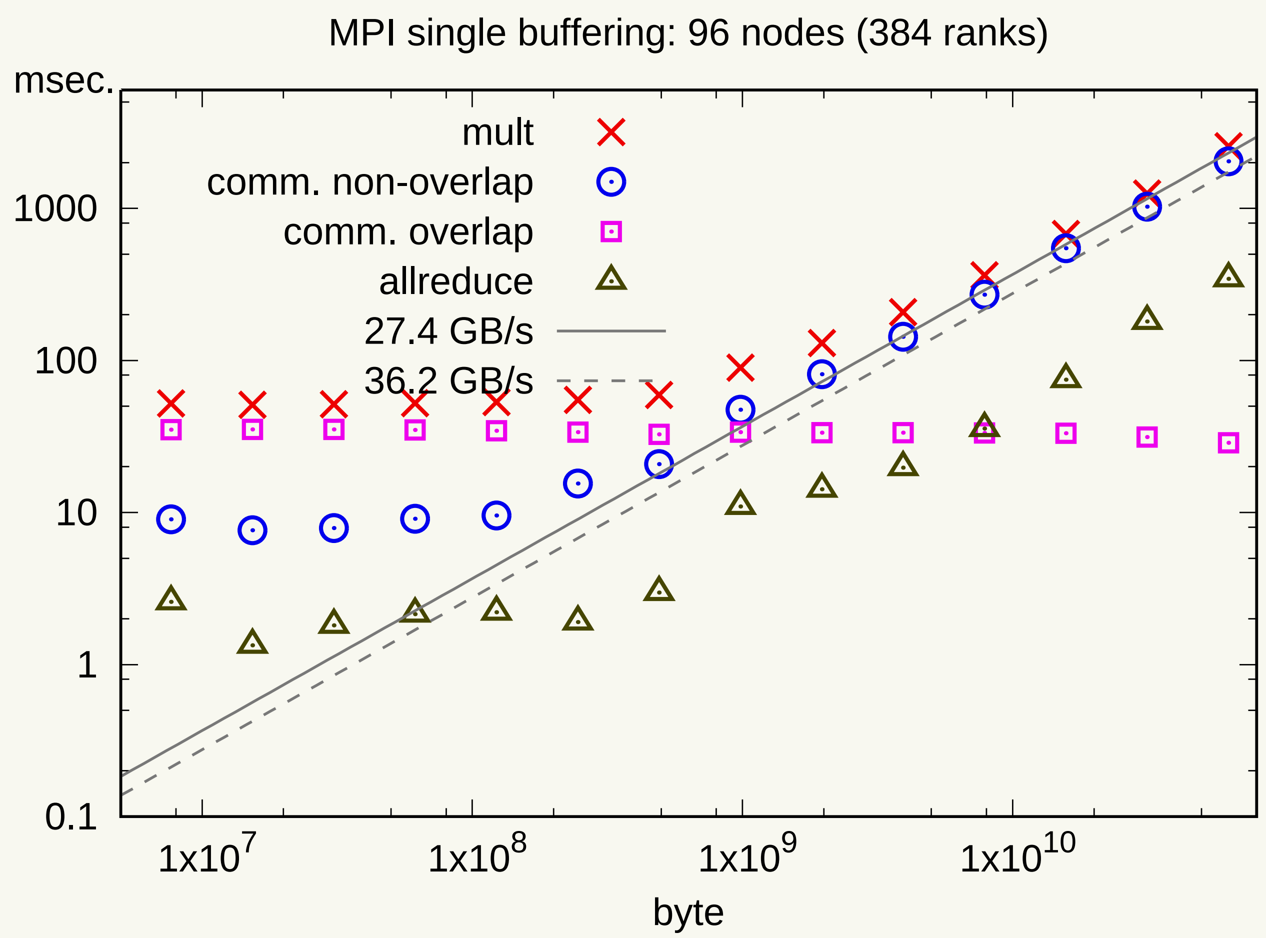
Elapsed Time vs. Amount of Communication: 96 nodes





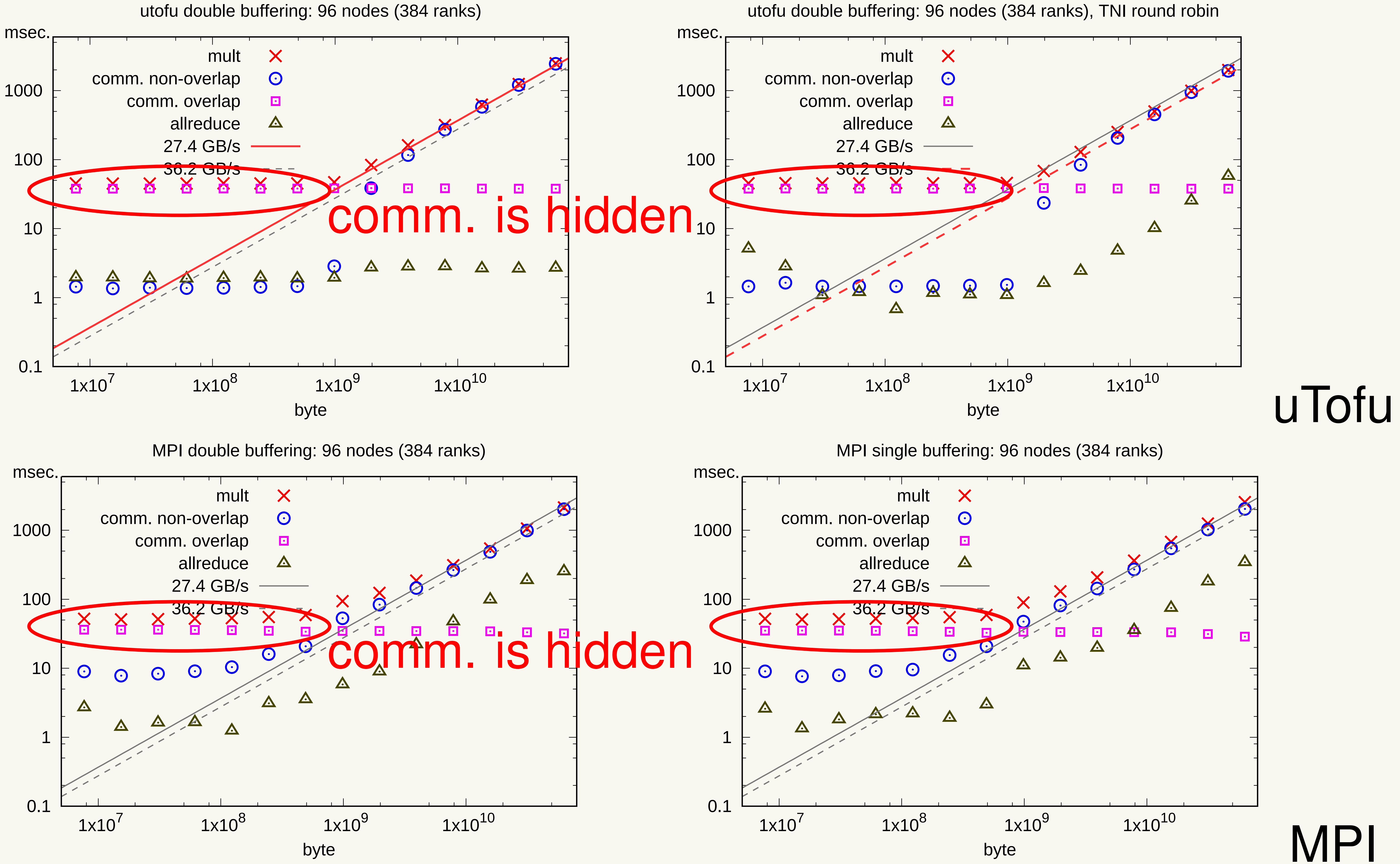
uTofu



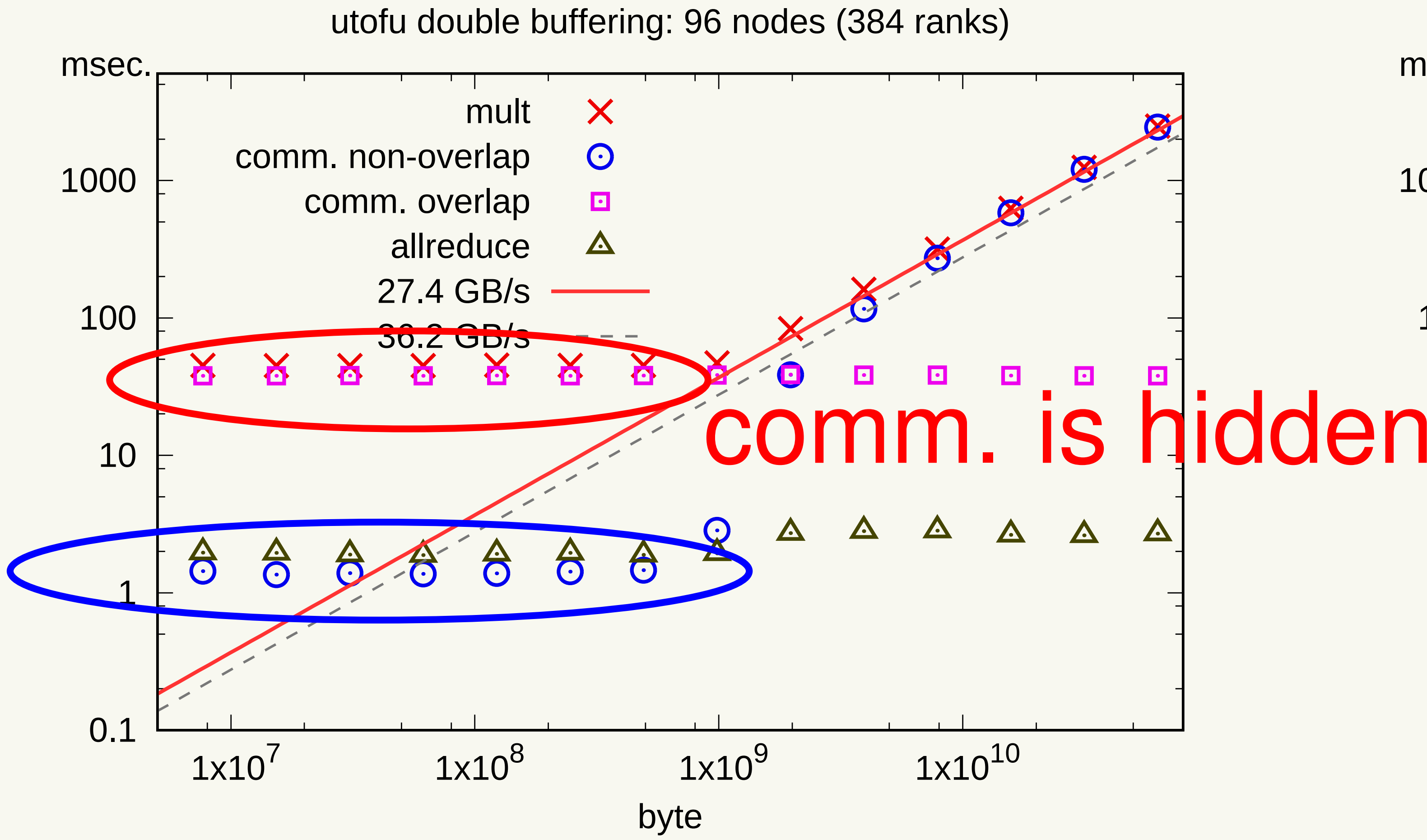


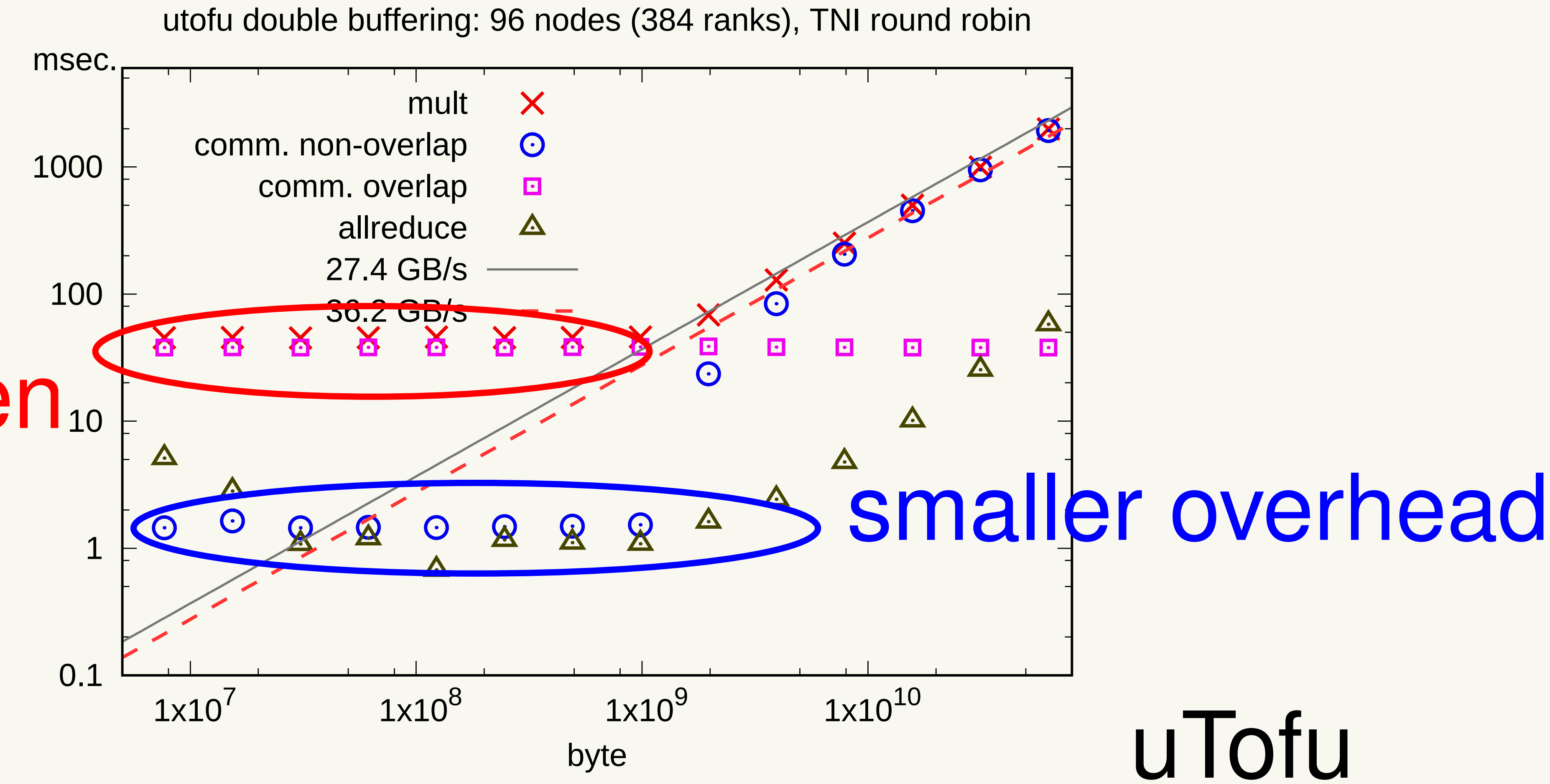
MPI

Elapsed Time vs. Amount of Communication: 96 nodes

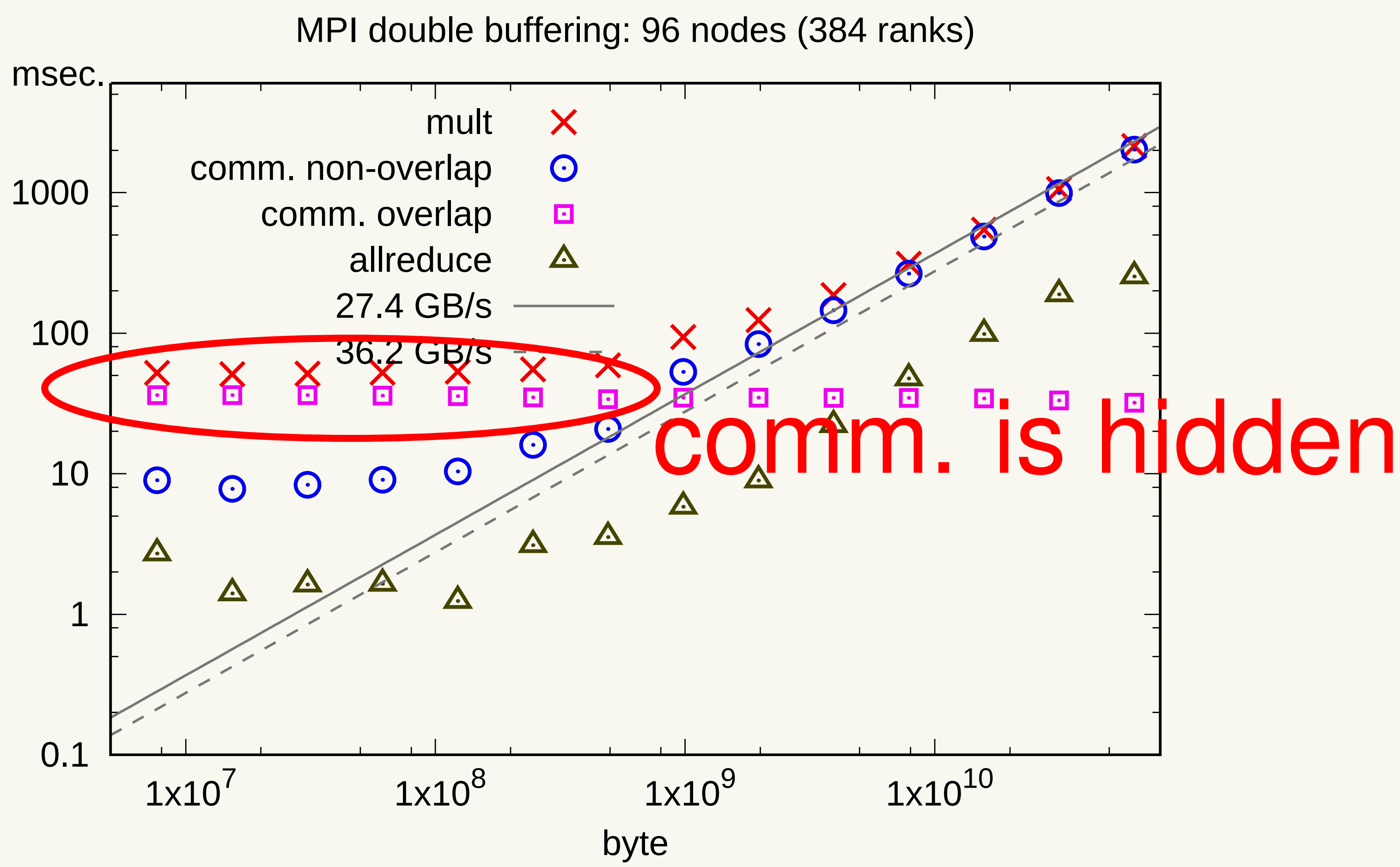


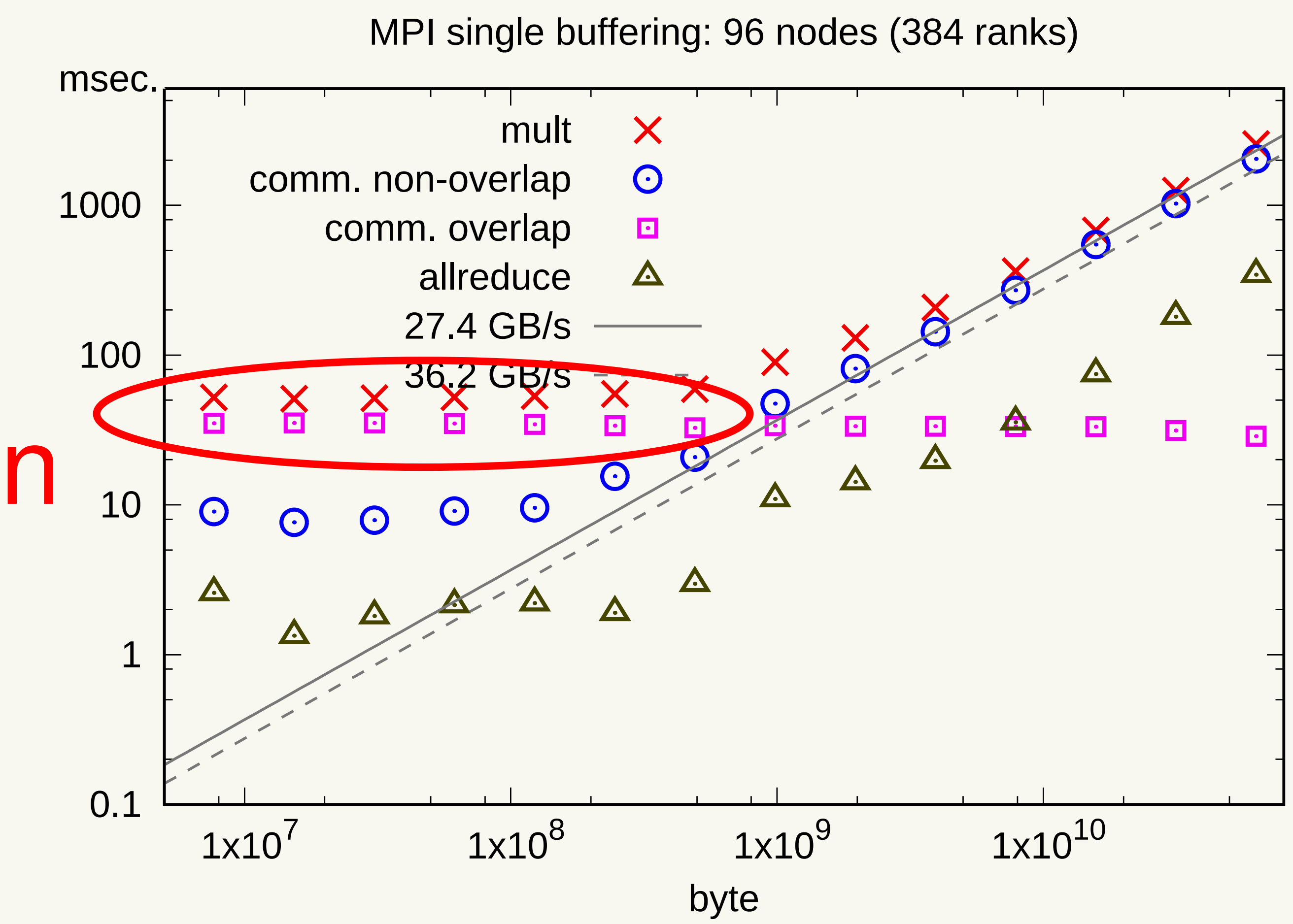
Elapsed Time vs. Amount of Communication: 96 nodes





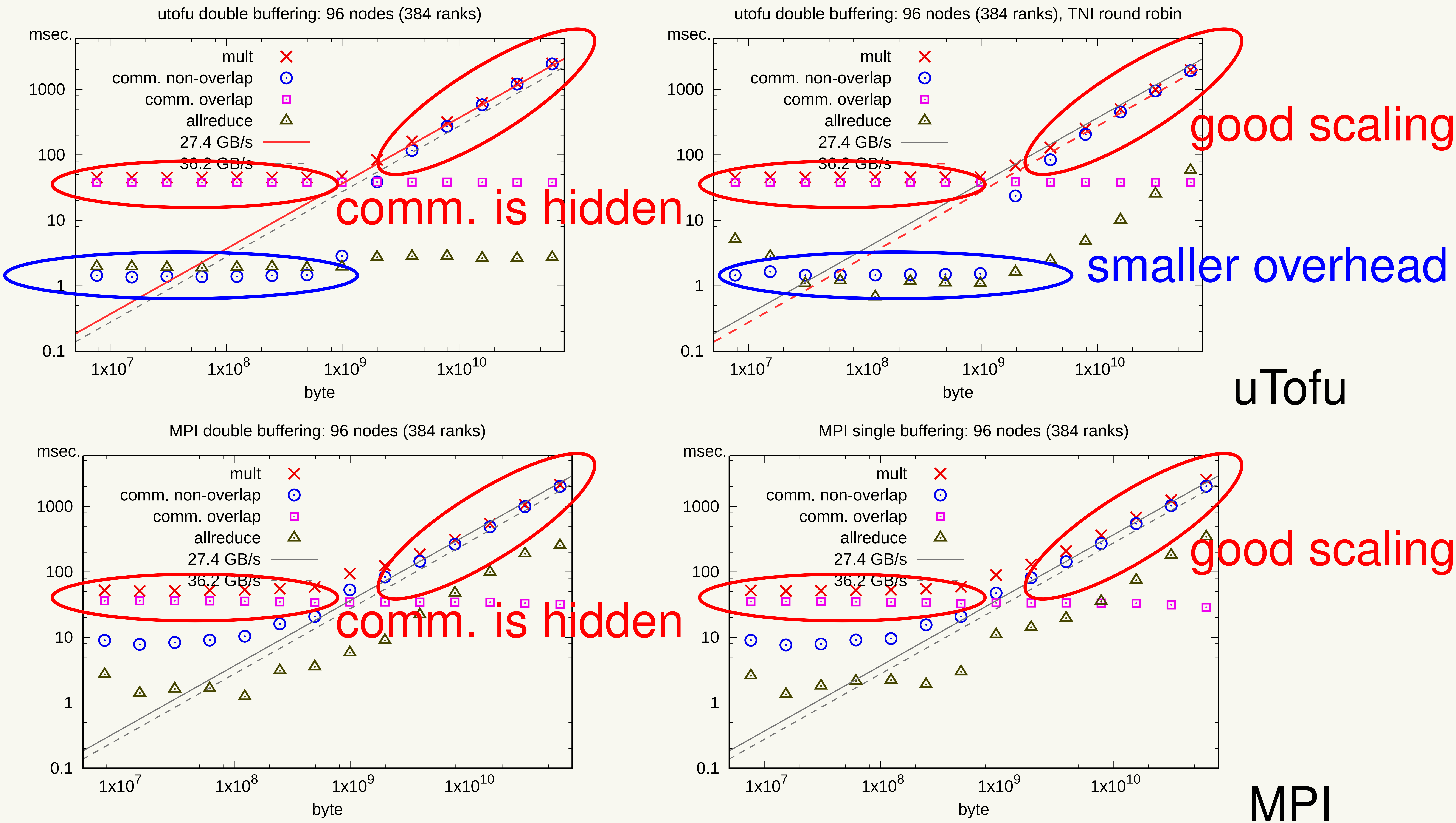
uTofu



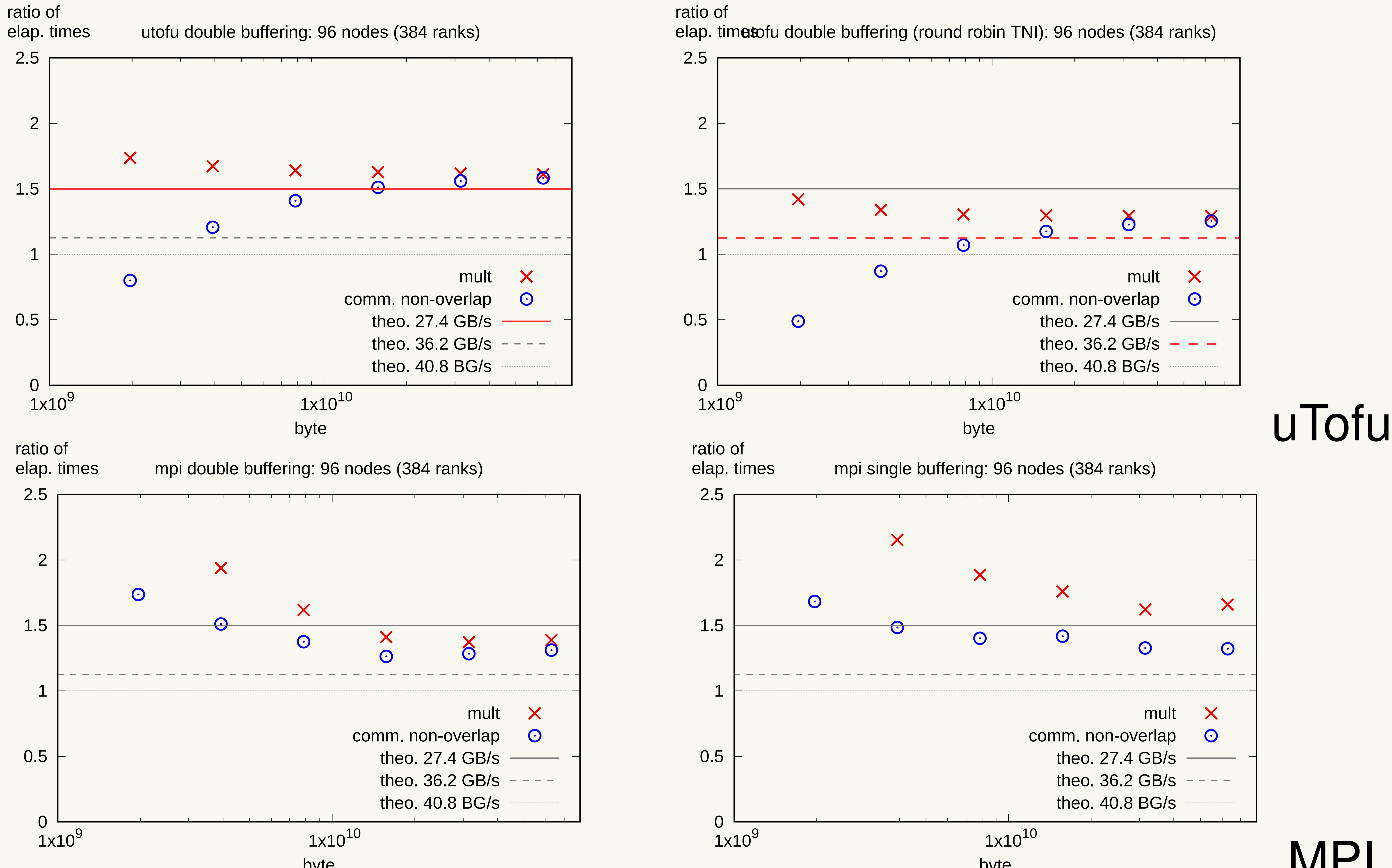


MPI

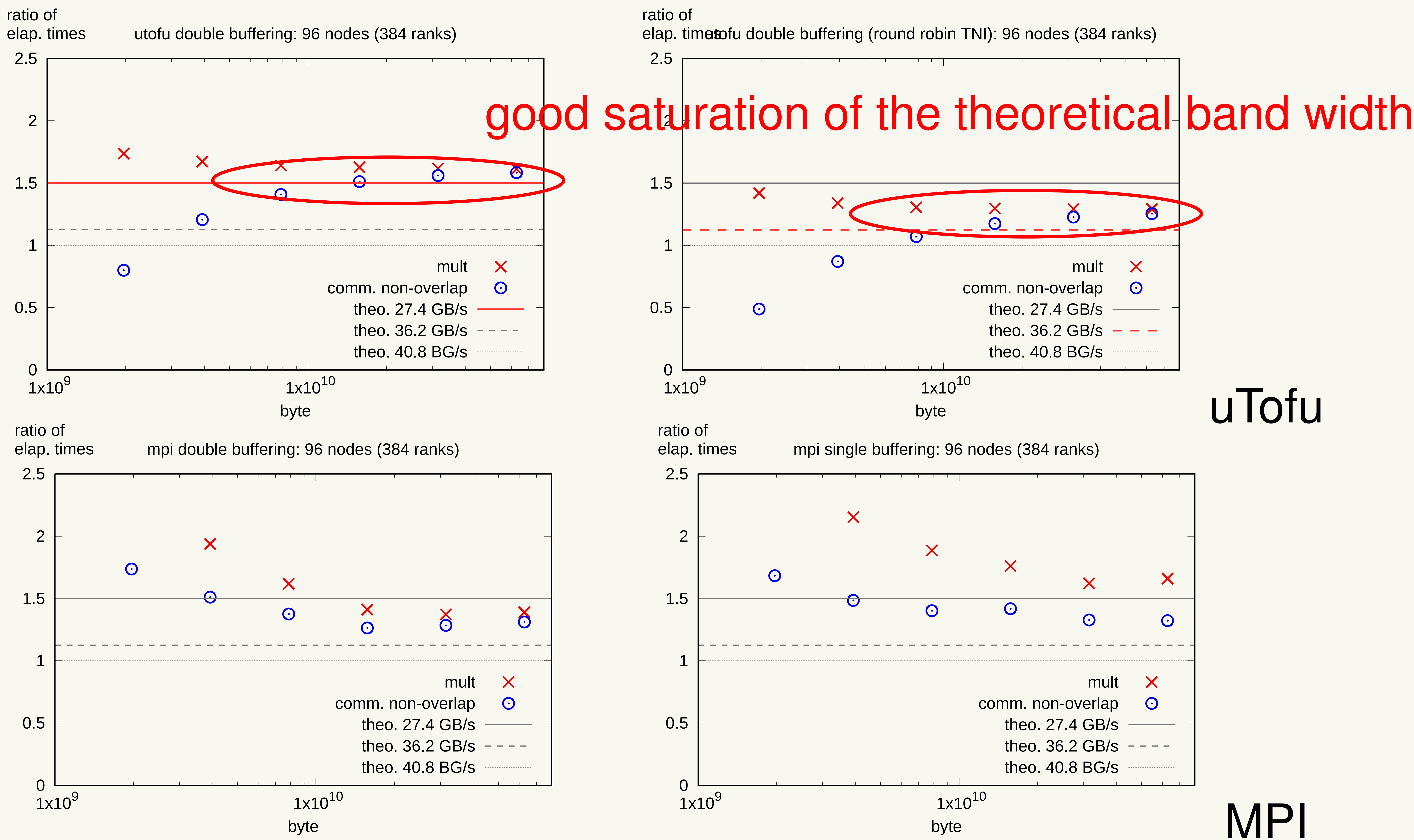
Elapsed Time vs. Amount of Communication: 96 nodes



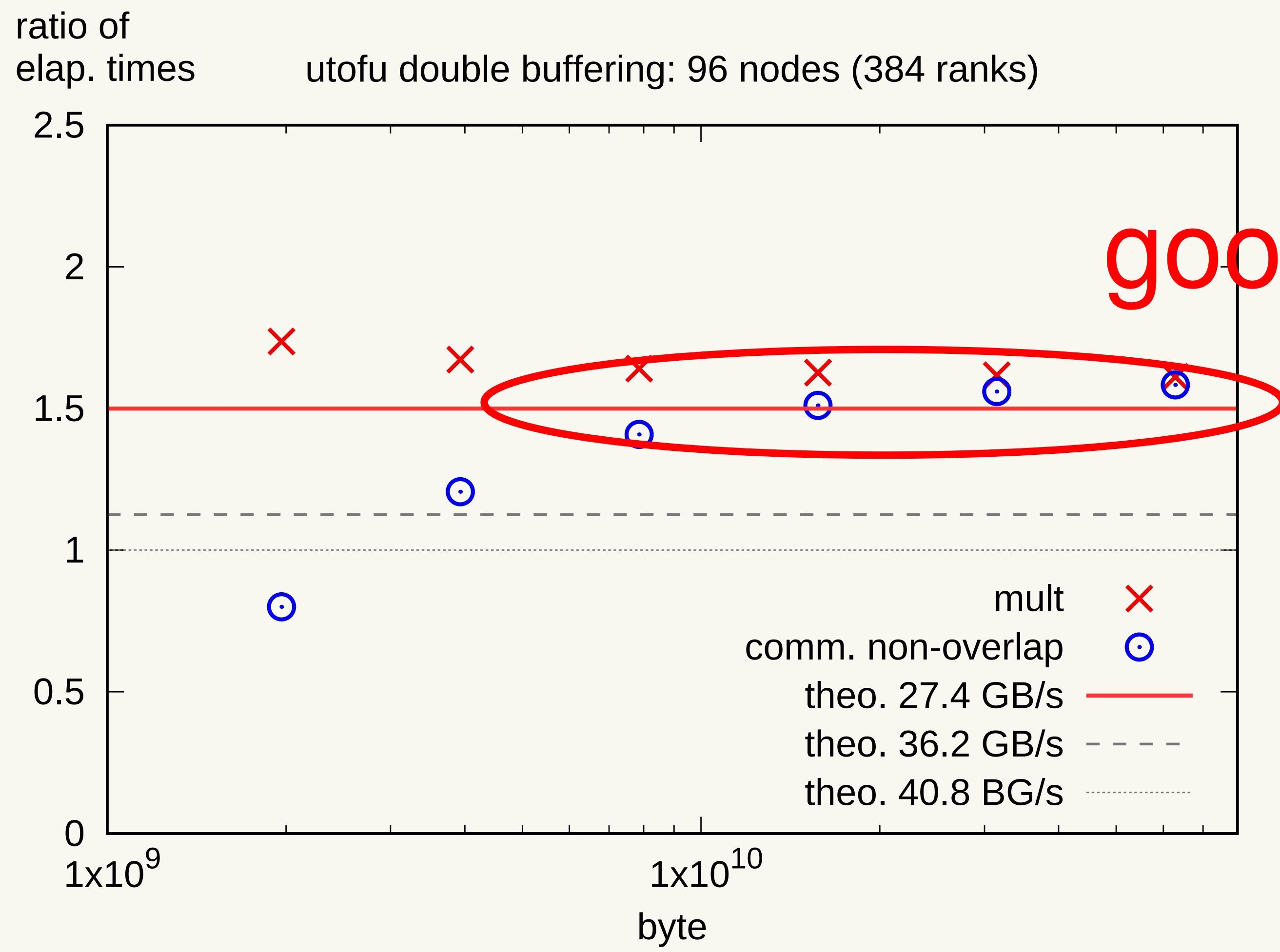
Saturation of the bandwidth for large data size

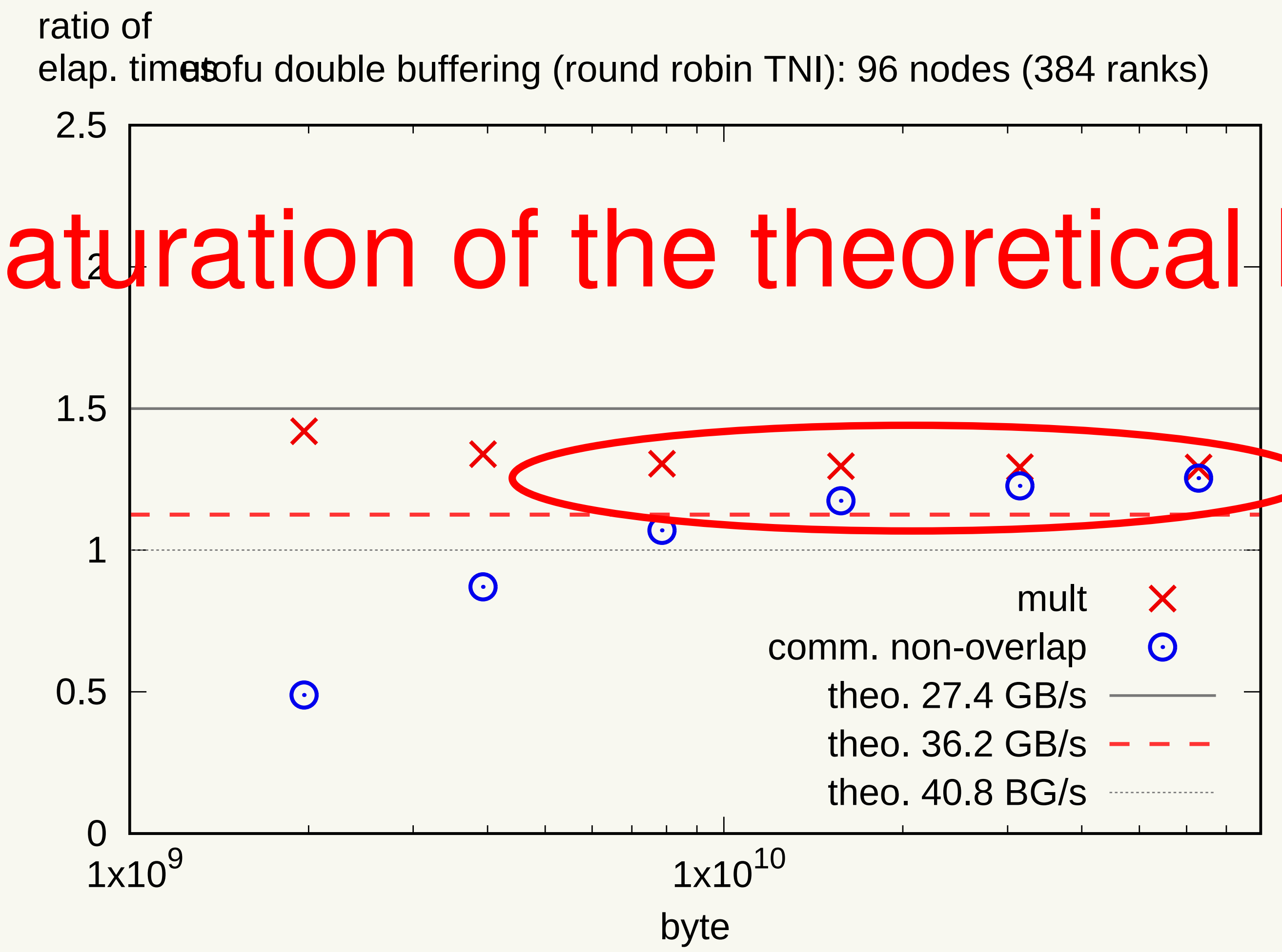


Saturation of the bandwidth for large data size



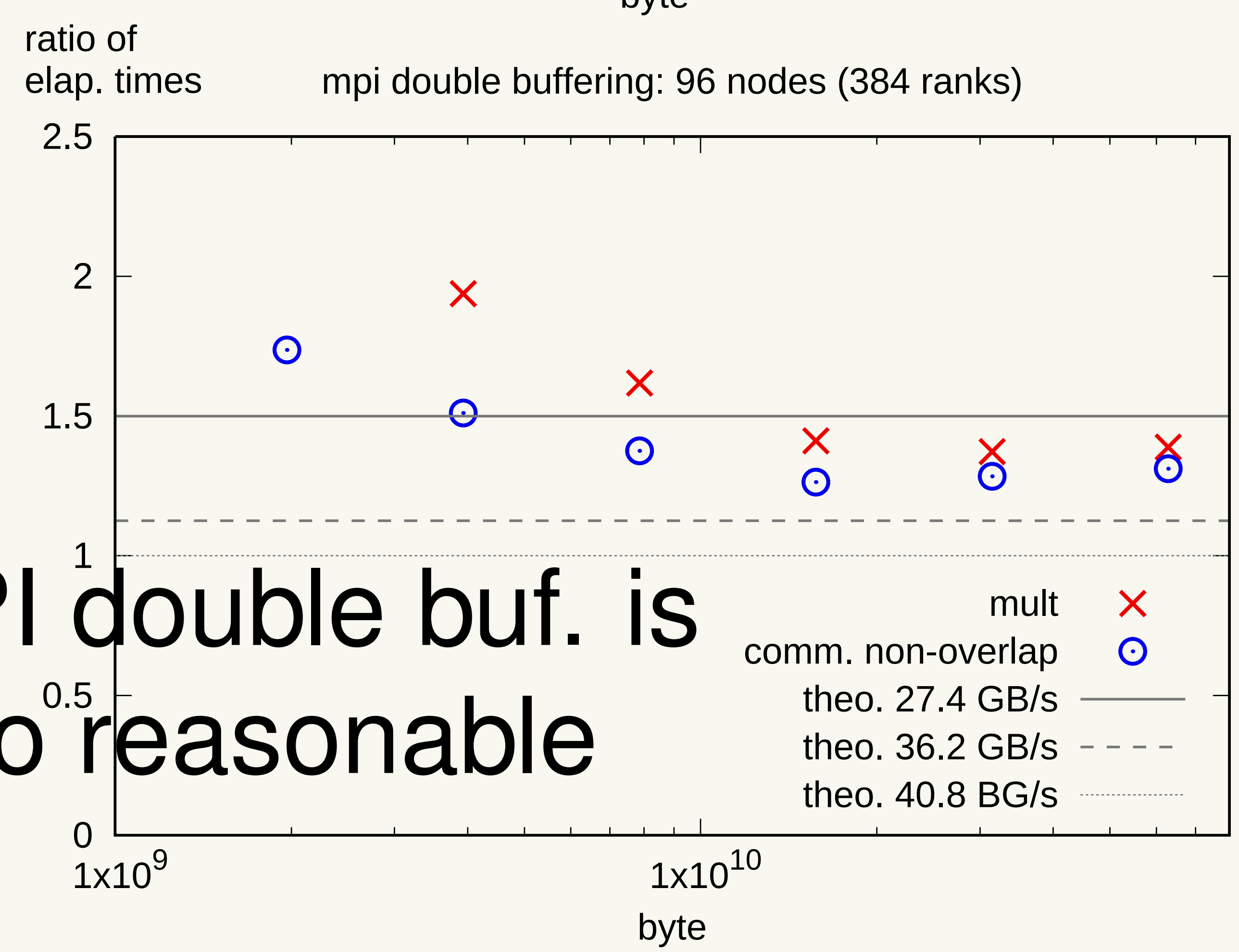
Saturation of the bandwidth for large data size



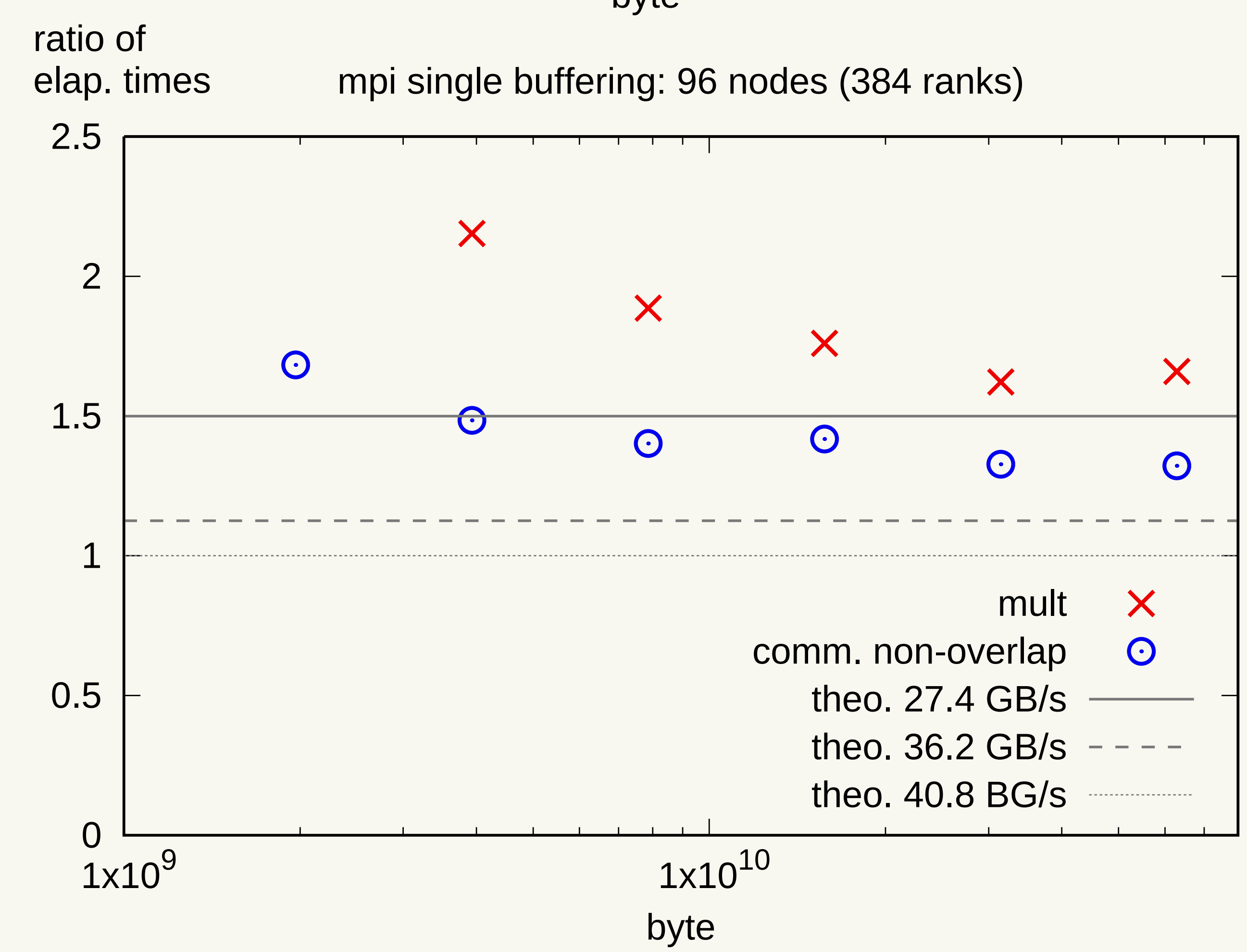


good saturation of the theoretical band width

uTofu



MPI double buf. is also reasonable



MPI

Summary and Outlooks

Neighboring Communication of QWS

- algorithm: double buffering
- implemented with uTofu   MPI version of QWS is available as well
- a proper TNI assignment is important   rank map is also important
- can be used as a library: ex. with 2-dim Poisson eq.
  - good saturation of the theoretical band width
  - good weak scaling
  - room (and/or freedom) for further optimization of TNI

Outlooks

- performance of QWS with practical system sizes
- [comm. part of] QWS + existing QCD code sets

TNI: Tofu Network Interface (RDMA engine)   RDMA: Remote Direct Memory Access
uTofu: Low Level Communcation API for TofuD

Neighboring Communication of QWS

- algorithm: double buffering
- implemented with uTofu MPI version of QWS is available as well
- a proper TNI assignment is important rank map is also important
- can be used as a library: ex. with 2-dim Poisson eq.
  - good saturation of the theoretical band width
  - good weak scaling
  - room (and/or freedom) for further optimization of TNI

Outlooks

- performance of QWS with practical system sizes
- [comm. part of] QWS + existing QCD code sets

TNI: Tofu Network Interface (RDMA engine) RDMA: Remote Direct Memory Access
uTofu: Low Level Communcation API for TofuD

Thank you.

Backup Slides

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer

// 1st iter.
pack the boundary data
start sending
computation: bulk
wait for the boundary data comes
computation: boundary
clear the received flag
wait for sending is done
switch the buffer to send
// 2nd iter.
pack the boundary data
start sending
computation: bulk
wait for the boundary data comes
computation: boundary
clear the received flag
wait for sending is done
switch the buffer to send
...

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer

```
// 1st iter.
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes
 computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 ...
```

recv. buffers

```
// 1st iter.
pack the boundary data
start sending
computation: bulk
wait for the boundary data comes
computation: boundary
clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer

P

```
// 1st iter.
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes
 computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 ...
```

recv. buffers

```
// 1st iter.
pack the boundary data

start sending
computation: bulk


wait for the boundary data comes
computation: boundary



clear the received flag
wait for sending is done


switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer (P, S)

```
// 1st iter.
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes
 computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 ...
```

recv. buffers

```
// 1st iter.
pack the boundary data
start sending
computation: bulk
wait for the boundary data comes
computation: boundary
clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer

```
// 1st iter.
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes
 computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 ...
```

recv. buffers

```
// 1st iter.
pack the boundary data
start sending
computation: bulk
wait for the boundary data comes
computation: boundary
clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer

```
// 1st iter.
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
// 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes
 computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
...
```

recv. buffers

```
// 1st iter.
pack the boundary data

start sending
computation: bulk


wait for the boundary data comes
computation: boundary



clear the received flag
wait for sending is done


switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer

```
// 1st iter.
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes
 computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 ...
```

recv. buffers

```
// 1st iter.
pack the boundary data
start sending
computation: bulk
wait for the boundary data comes
computation: boundary
clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

send buffer

```
// 1st iter.
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes
wait for sending is done
switch the buffer to send
 ...
```

recv. buffers

```
// 1st iter.
pack the boundary data

start sending
computation: bulk


wait for the boundary data comes
computation: boundary



clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

even if the 1st buffer is still busy but the 2nd is available

Double Buffering Algorithm

To hide a possible load imbalance btw. nodes, and to minimize the latency, we use double buffering algorithm and implement it with uTofu.

send buf.

- P : Packed
- P : Sending

recv. buf.

- R : Receiving
- R : Recv. done
- U : being Used

```
// 1st iter.                          send buffer
pack the boundary data
start sending
 computation: bulk
wait for the boundary data comes
computation: boundary
 clear the received flag
wait for sending is done
switch the buffer to send
 // 2nd iter.
pack the boundary data
start sending
 computation: bulk
 wait for the boundary data comes

wait for sending is done
switch the buffer to send
 ...
```

```
// 1st iter.                          recv. buffers
pack the boundary data

start sending
computation: bulk


wait for the boundary data comes
computation: boundary



clear the received flag
wait for sending is done

switch the buffer to send
 // 2nd iter.
 pack the boundary data
 start sending
computation: bulk
wait for the boundary data comes
computation: boundary
 wait for sending is done
 switch the buffer to send
 ...
```

even if the 1st buffer is still busy but the 2nd is available

no need to check if the recv. buf. is available

Hopping (Mult of H)

1. packing the boundary data
2. start sending/receiving the boundary data
3. calculate: internal area
4. wait for receiving
5. calculate: boundary area
6. wait for sending finished

Hopping (Mult of H)

1. packing the boundary data
2. start sending/receiving the boundary data
3. calculate: internal area
4. wait for receiving
5. calculate: boundary area
6. wait for sending finished

overlap

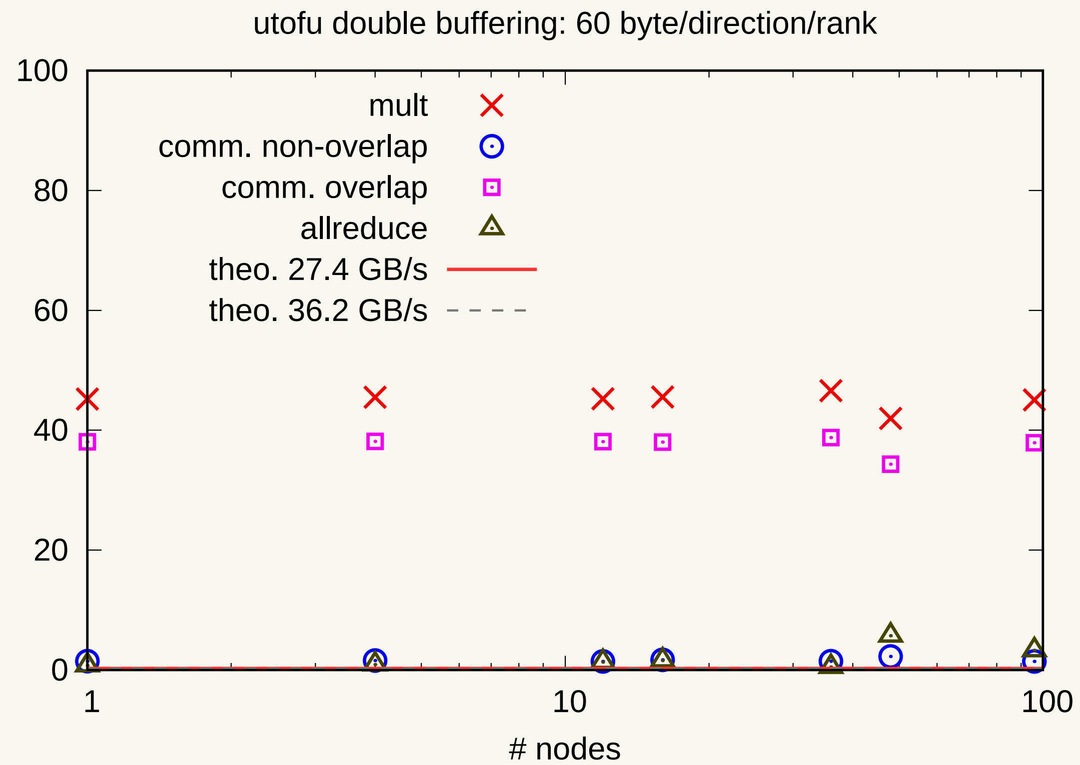
comm.

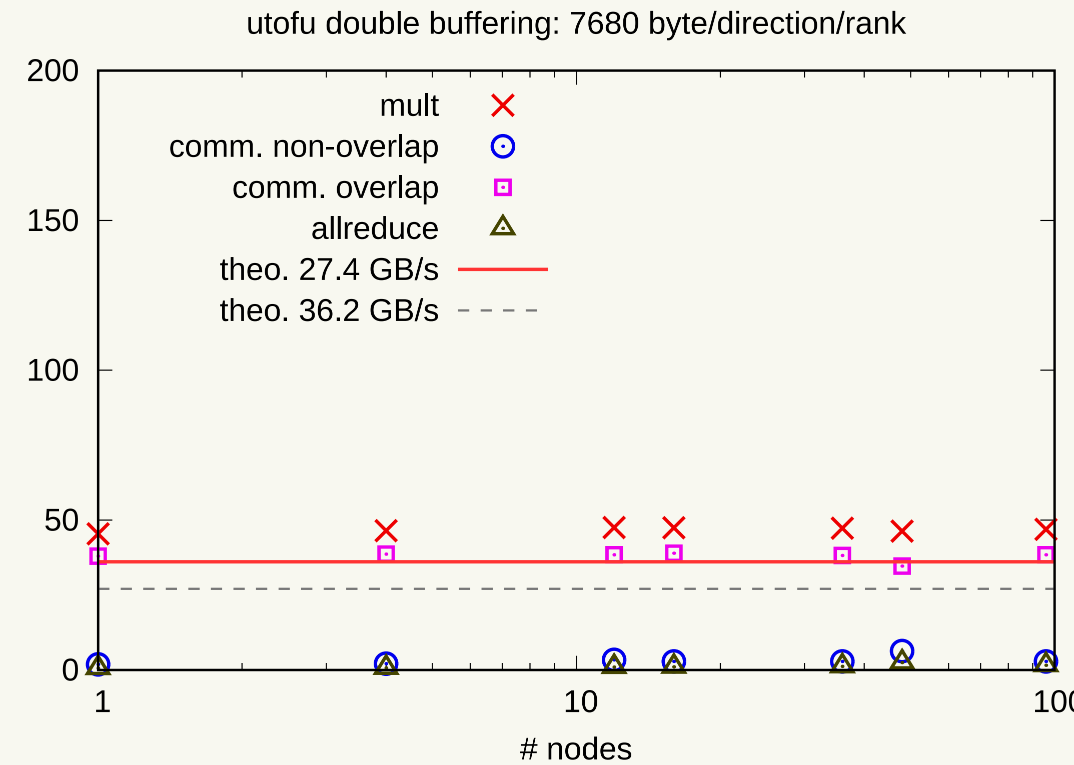
non-overlap = comm. − overlap

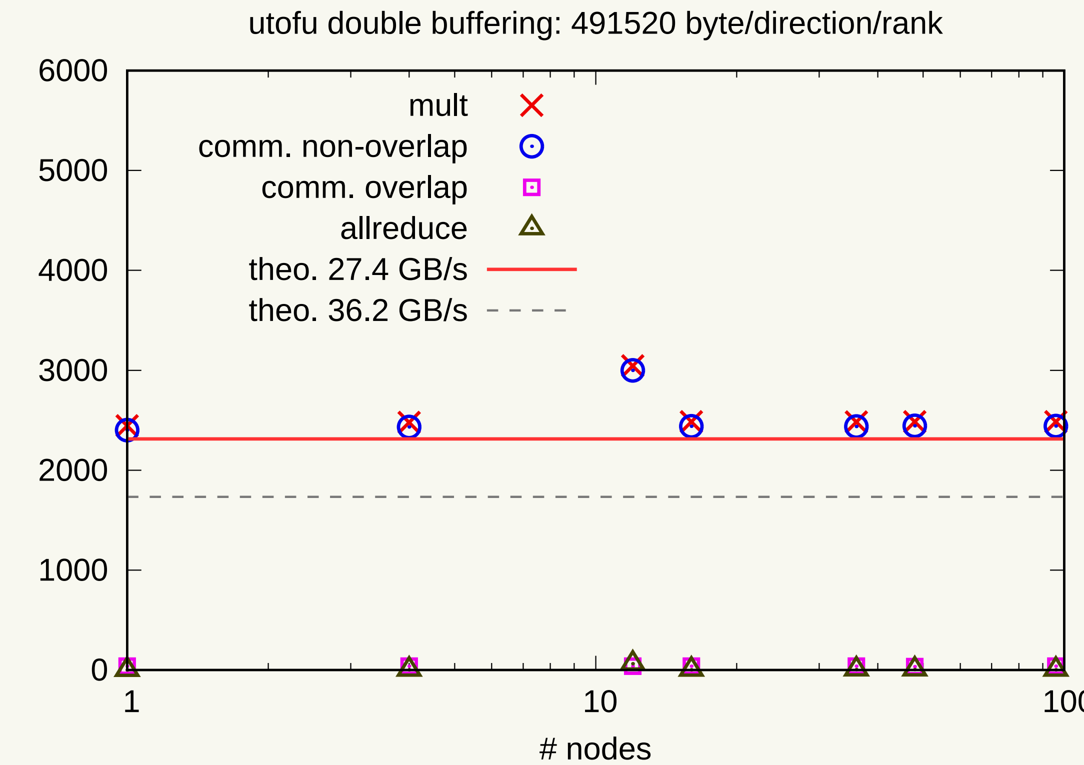
= start sending/receiving + wait for receiving

weak scaling with different data size

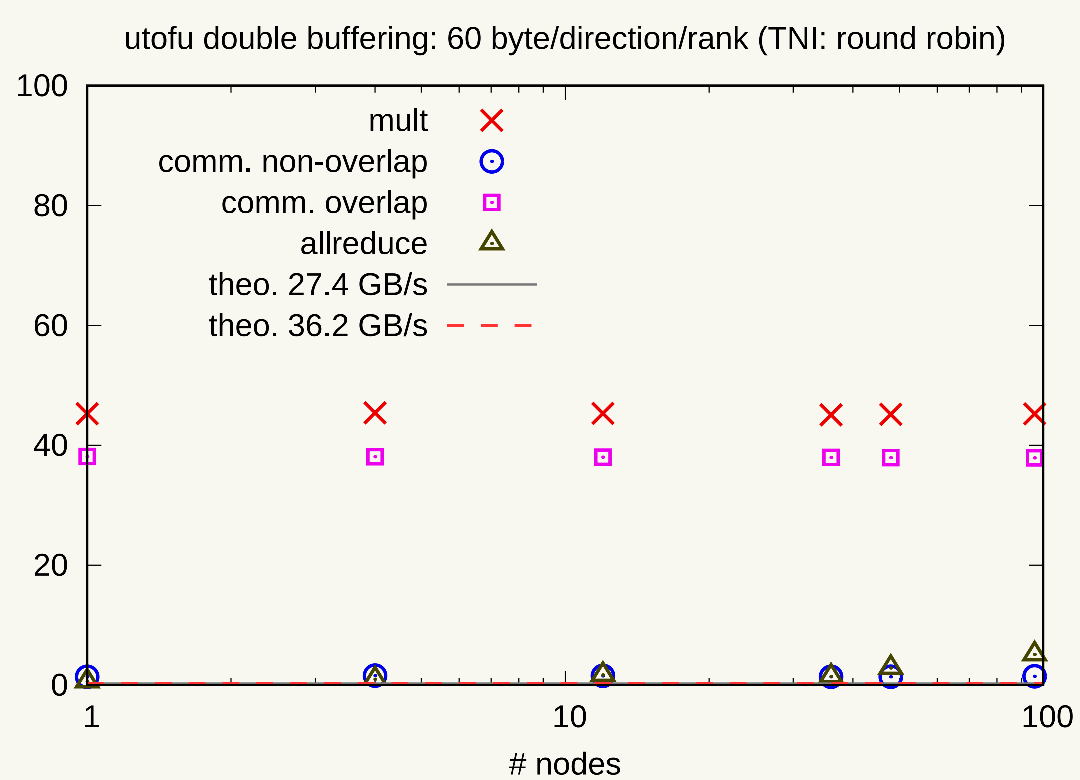
uTofu double buffering

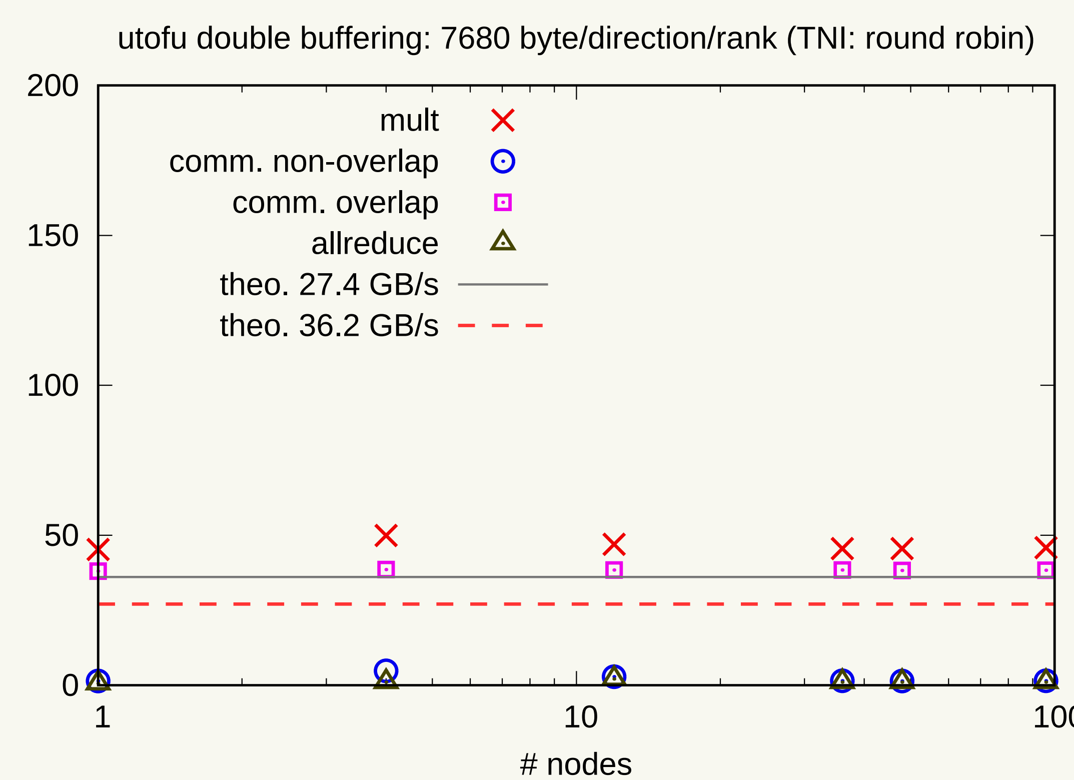


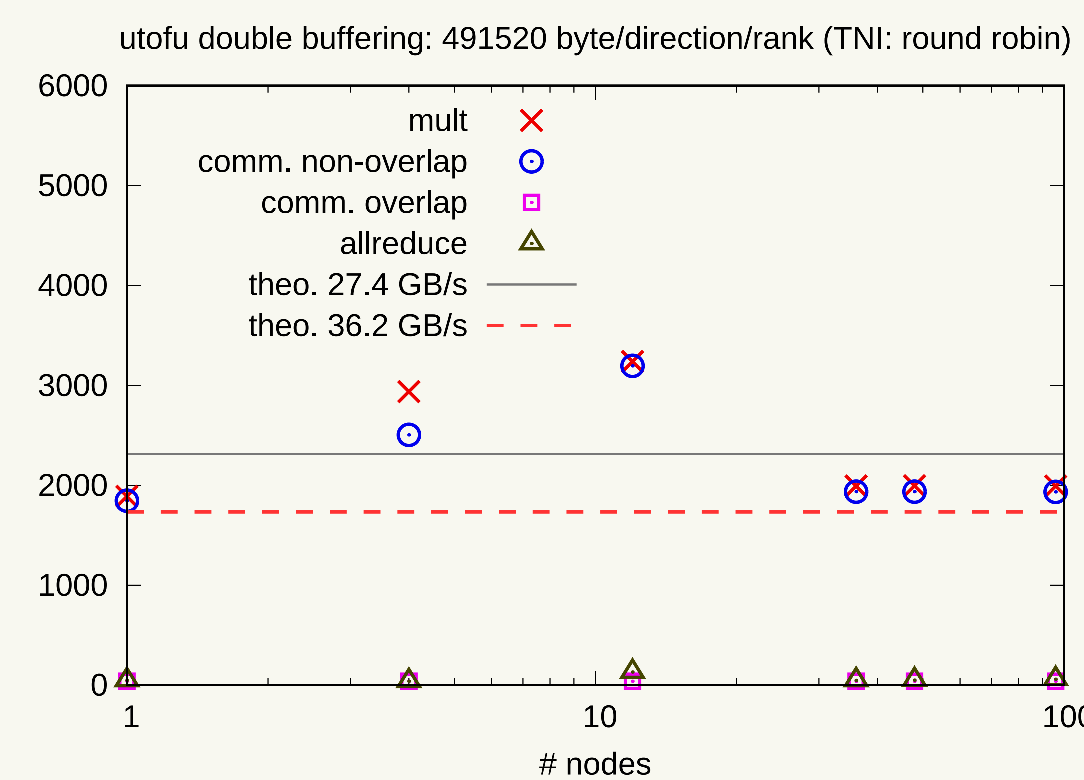




uTofu double buffering (round robin TNI)



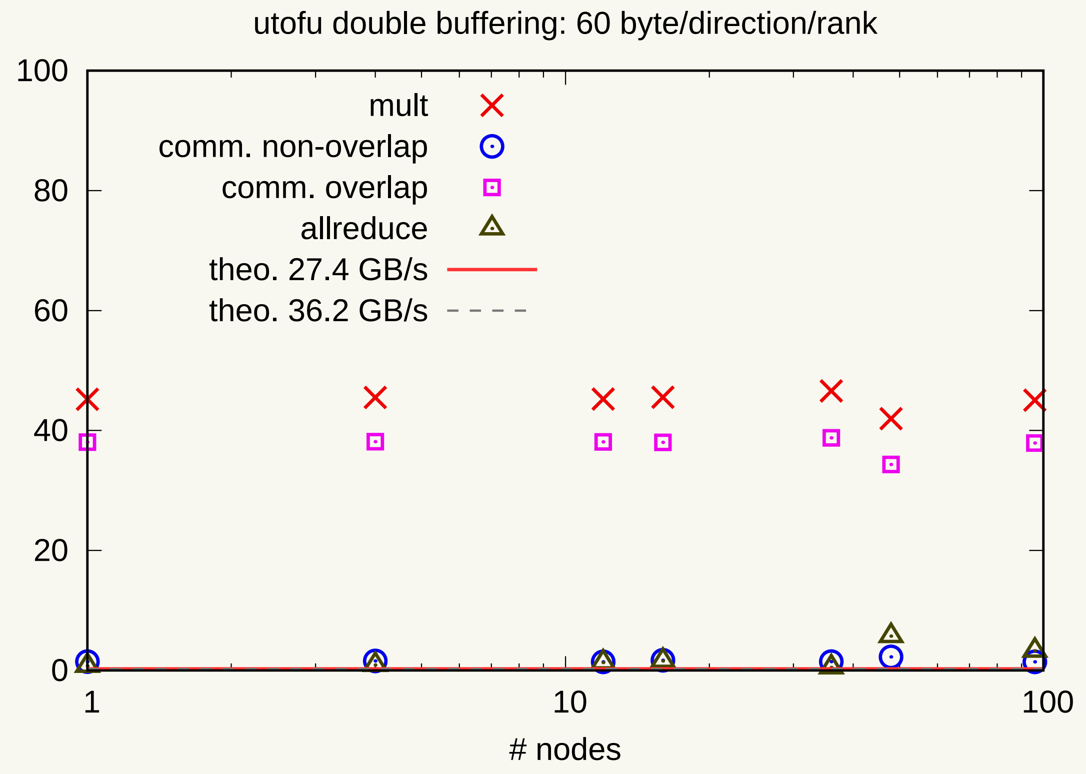


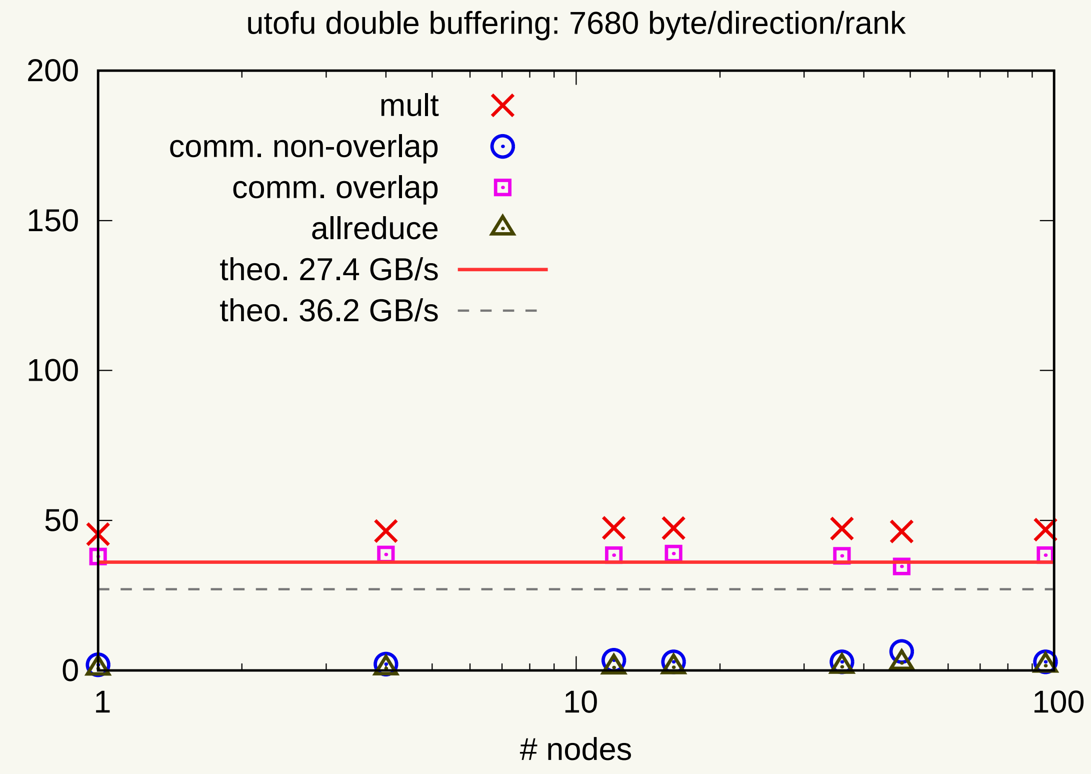


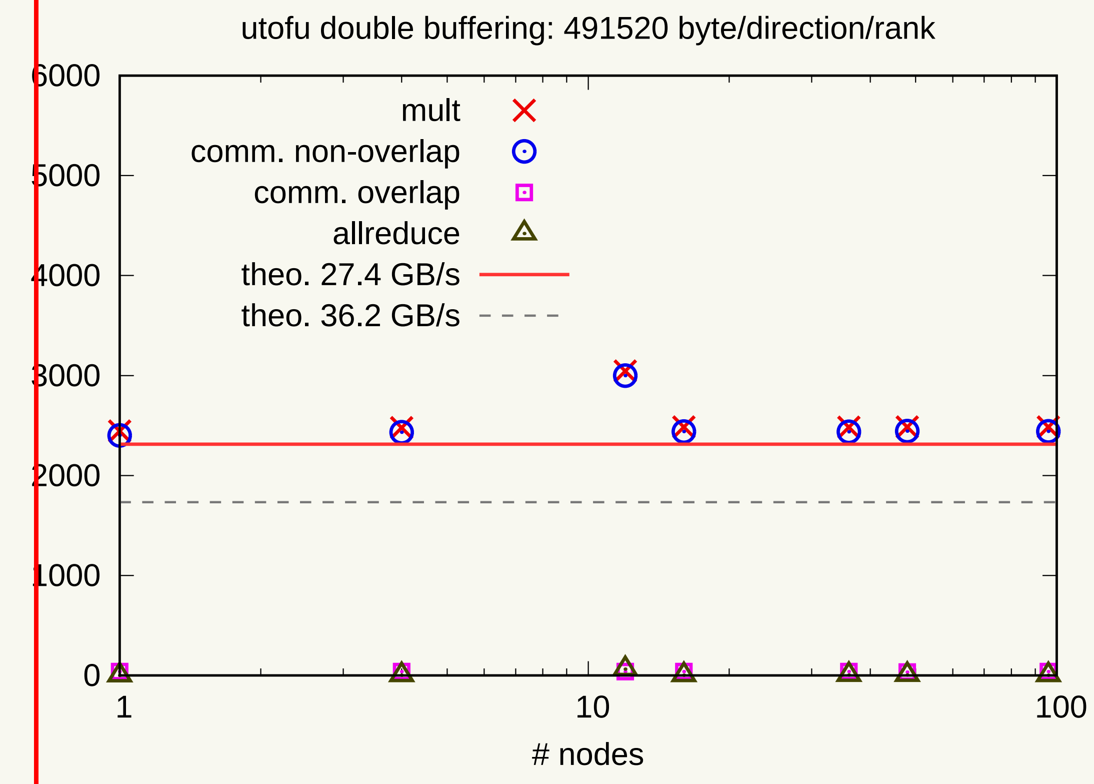
weak scaling with different data size

good weak scaling if the communication is fully overlapped

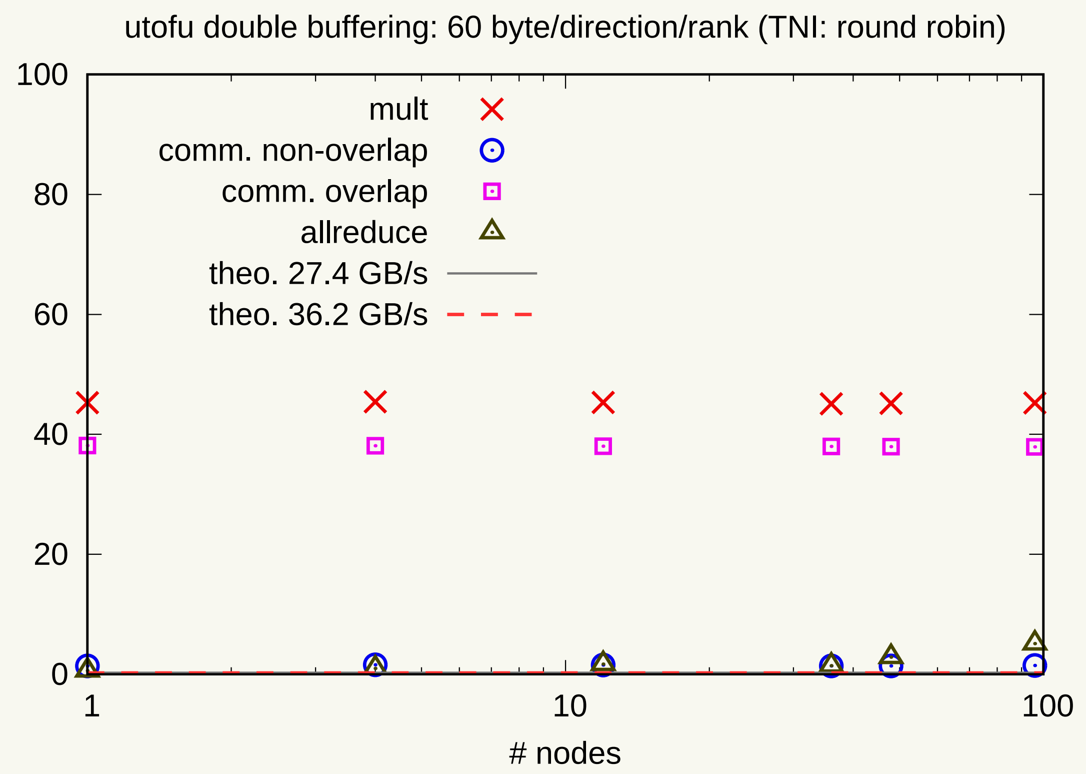
uTofu double buffering

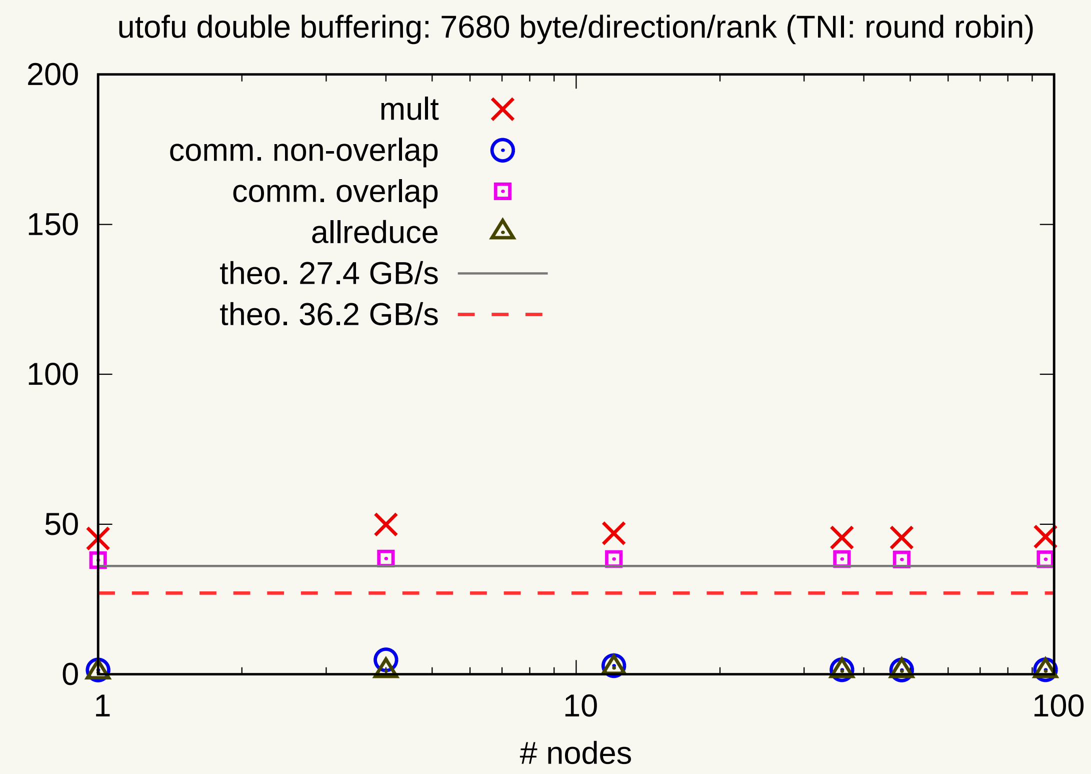


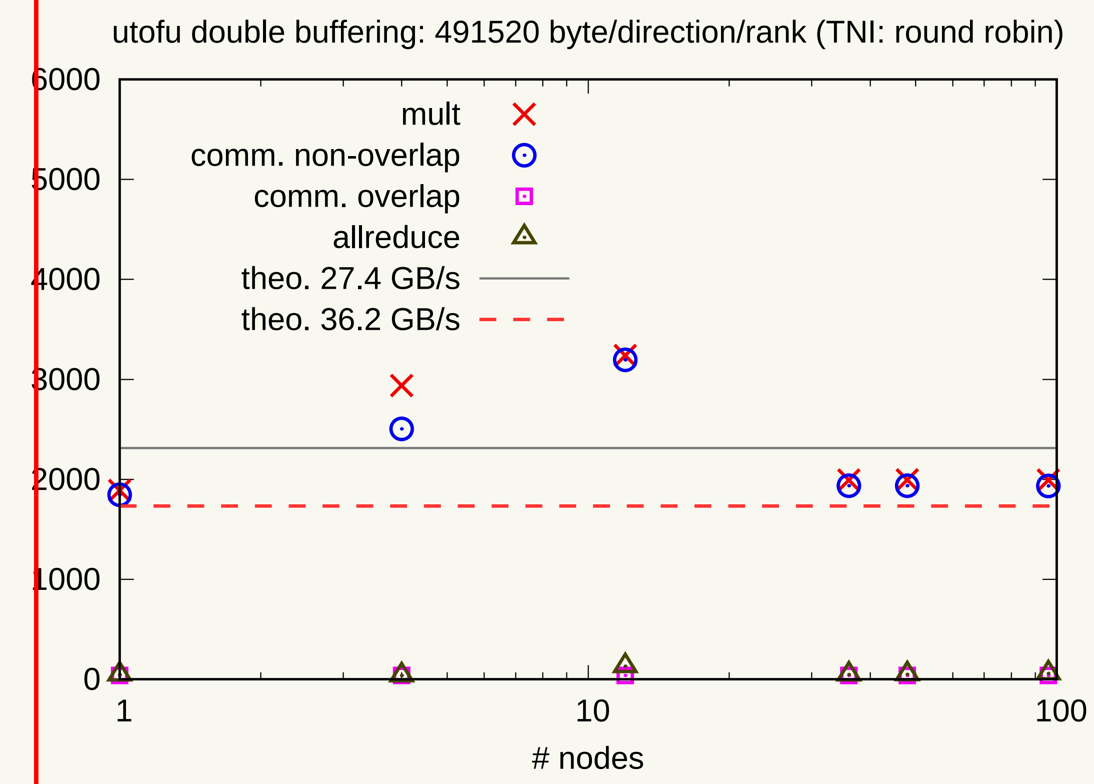




uTofu double buffering (round robin TNI)



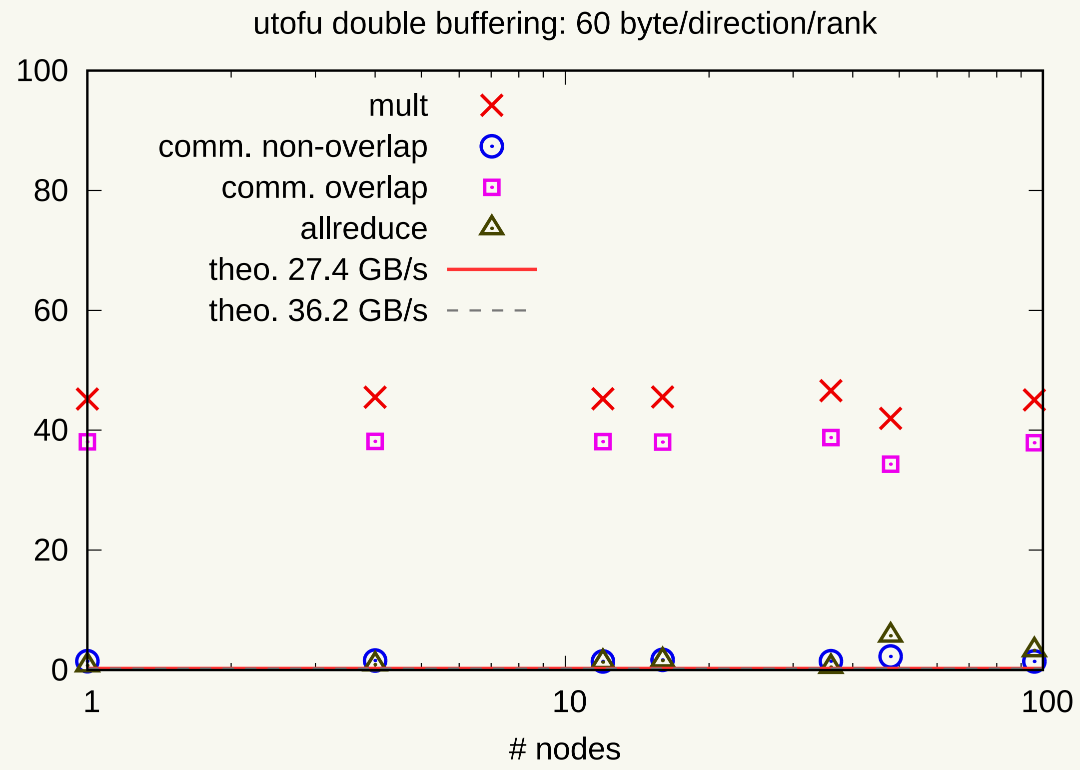


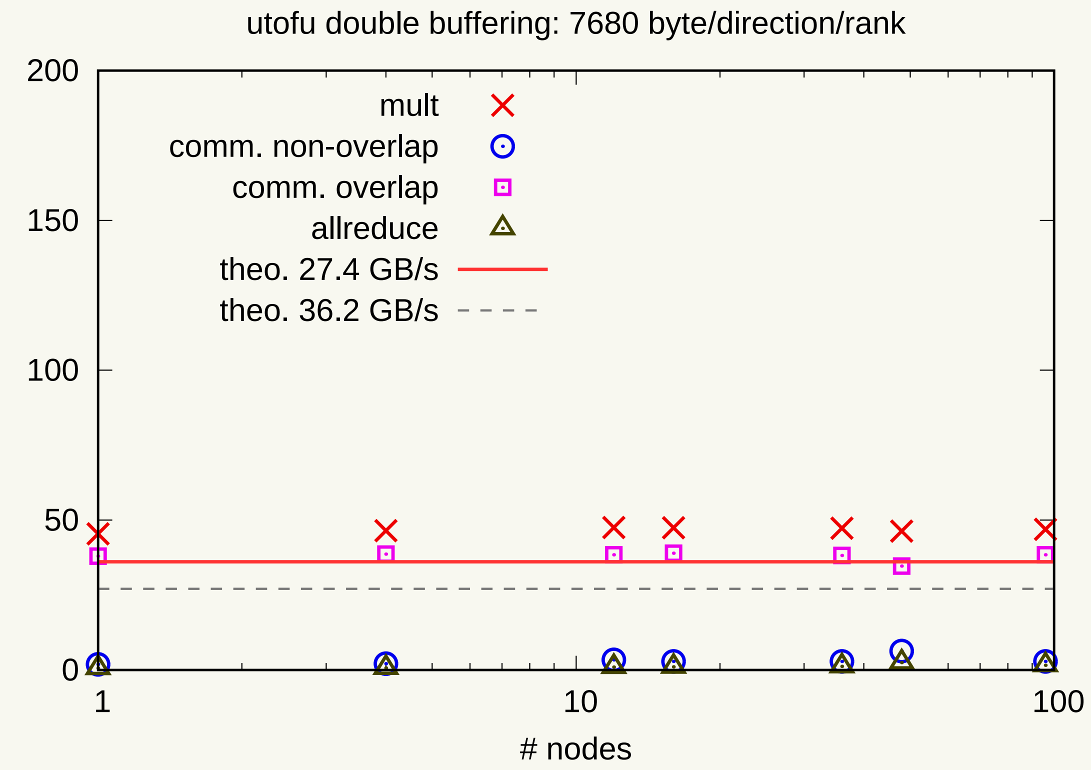


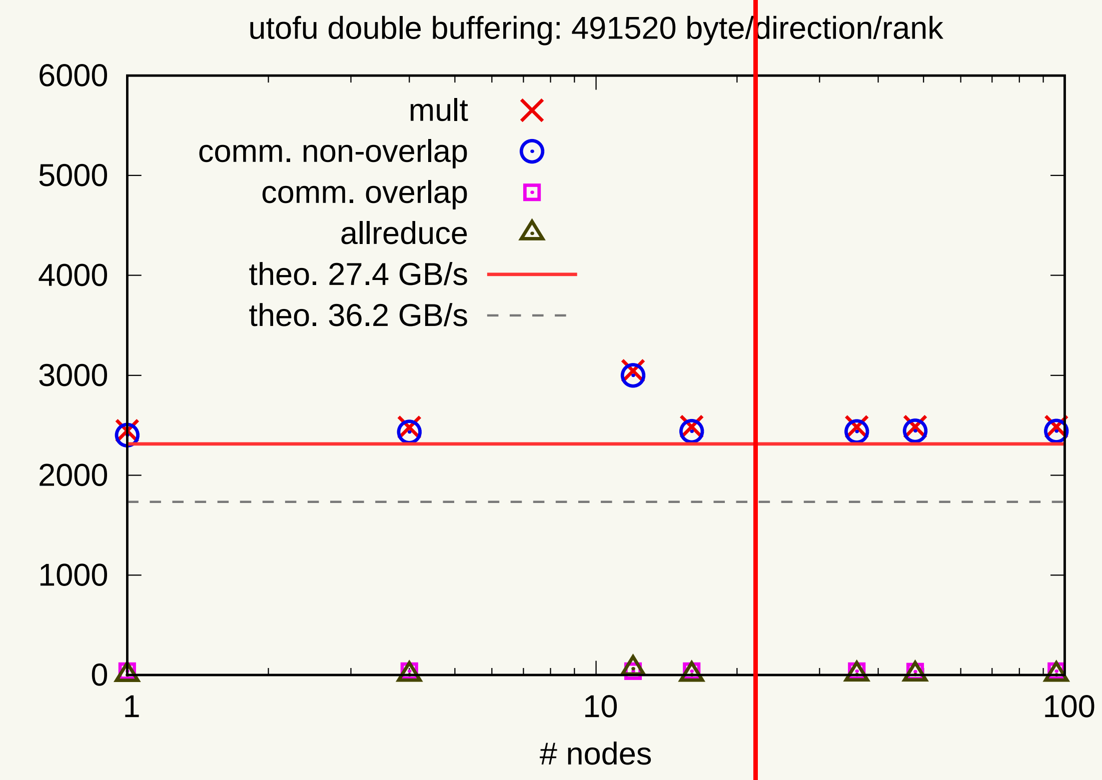
weak scaling with different data size

if comm. becomes visible, good weak scaling for large # of nodes

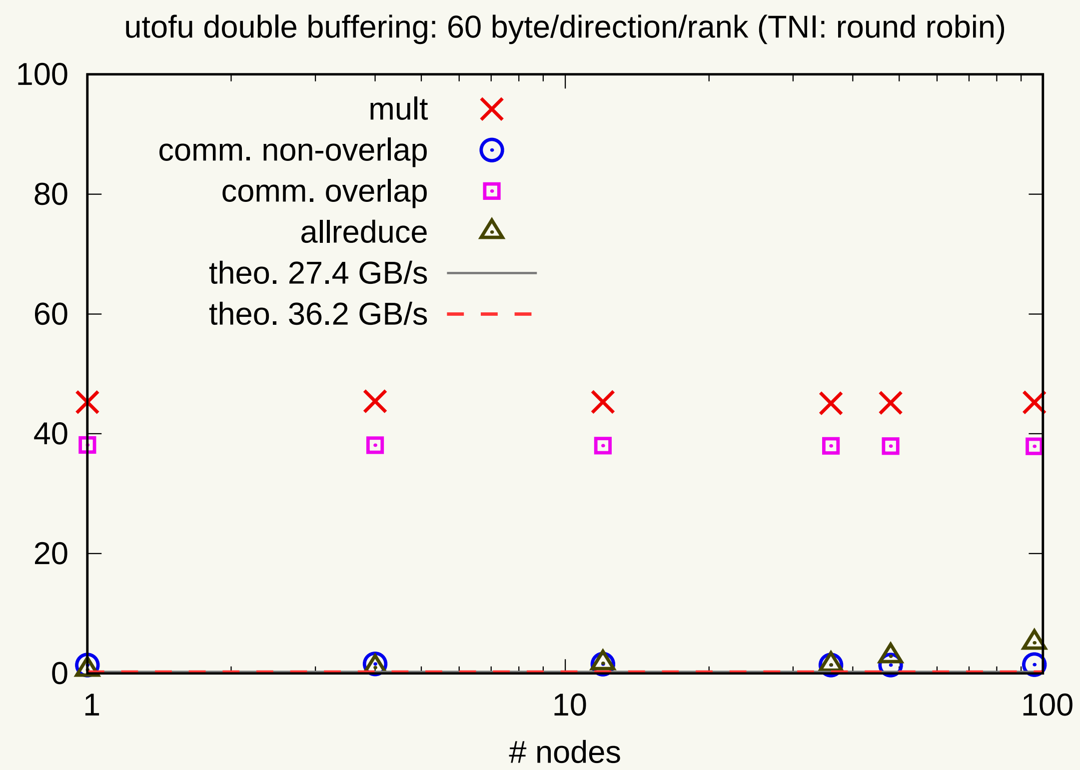
uTofu double buffering

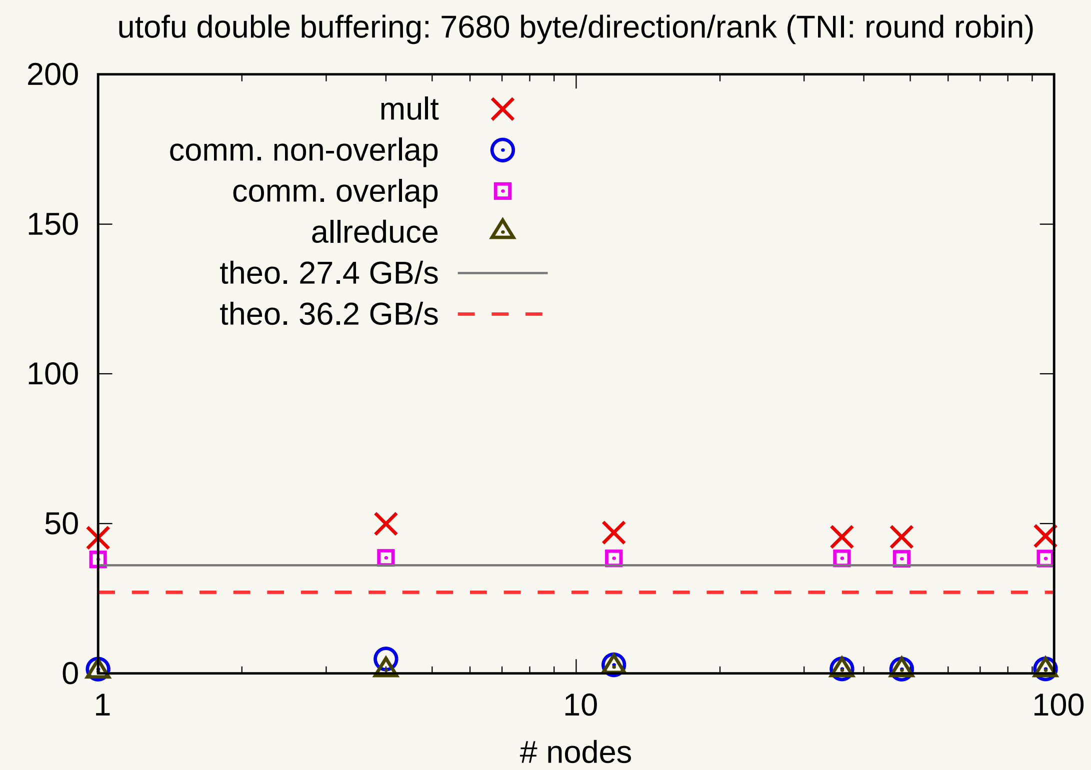


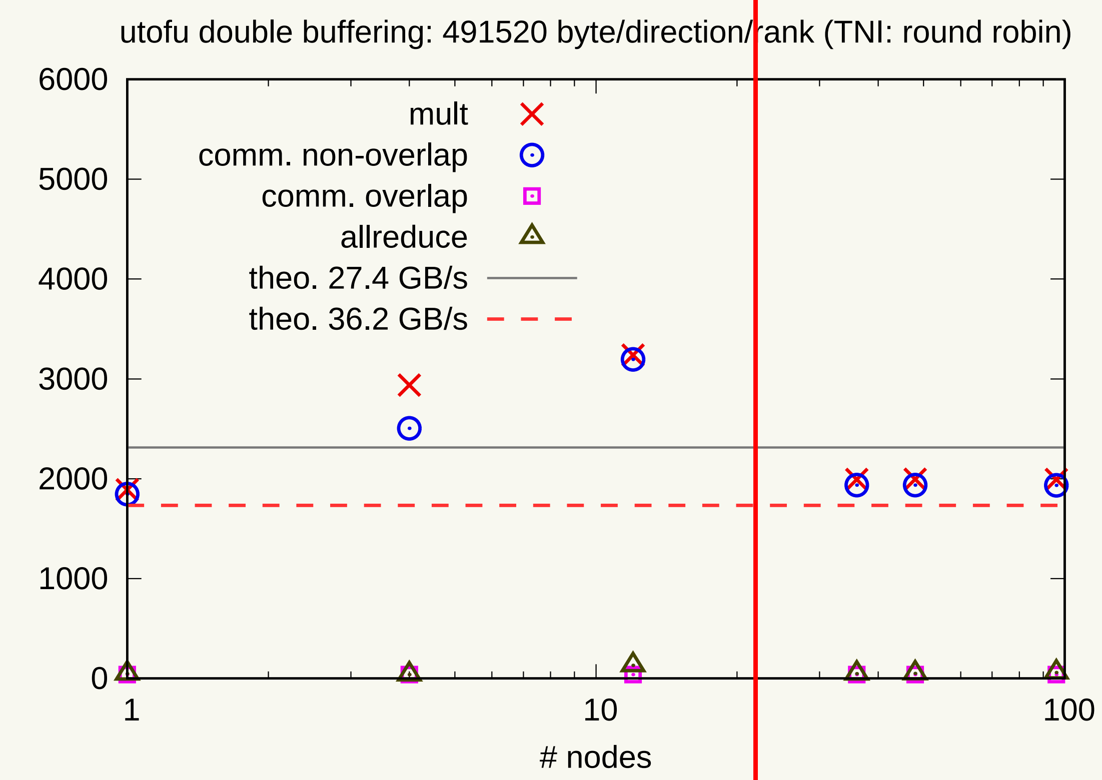




uTofu double buffering (round robin TNI)

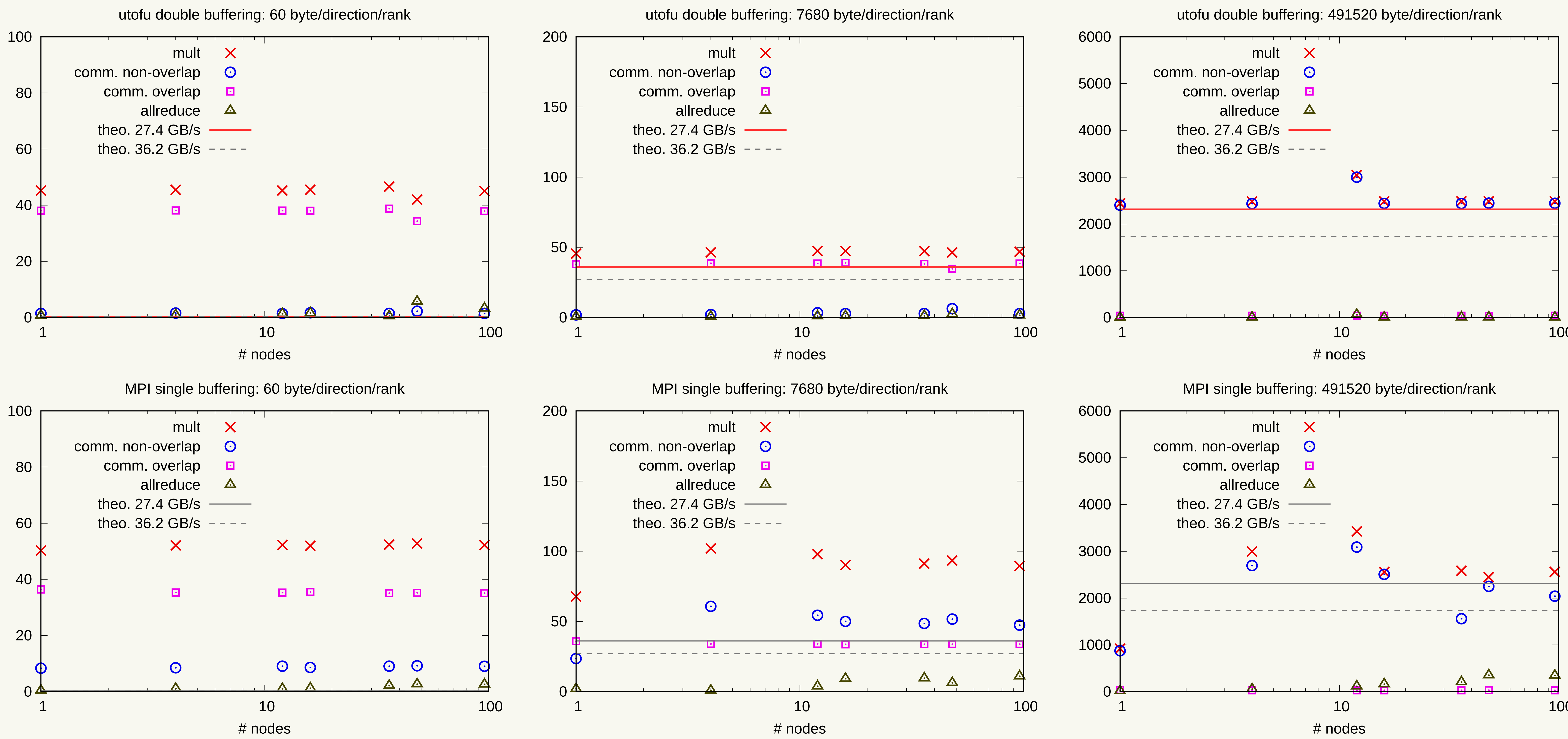






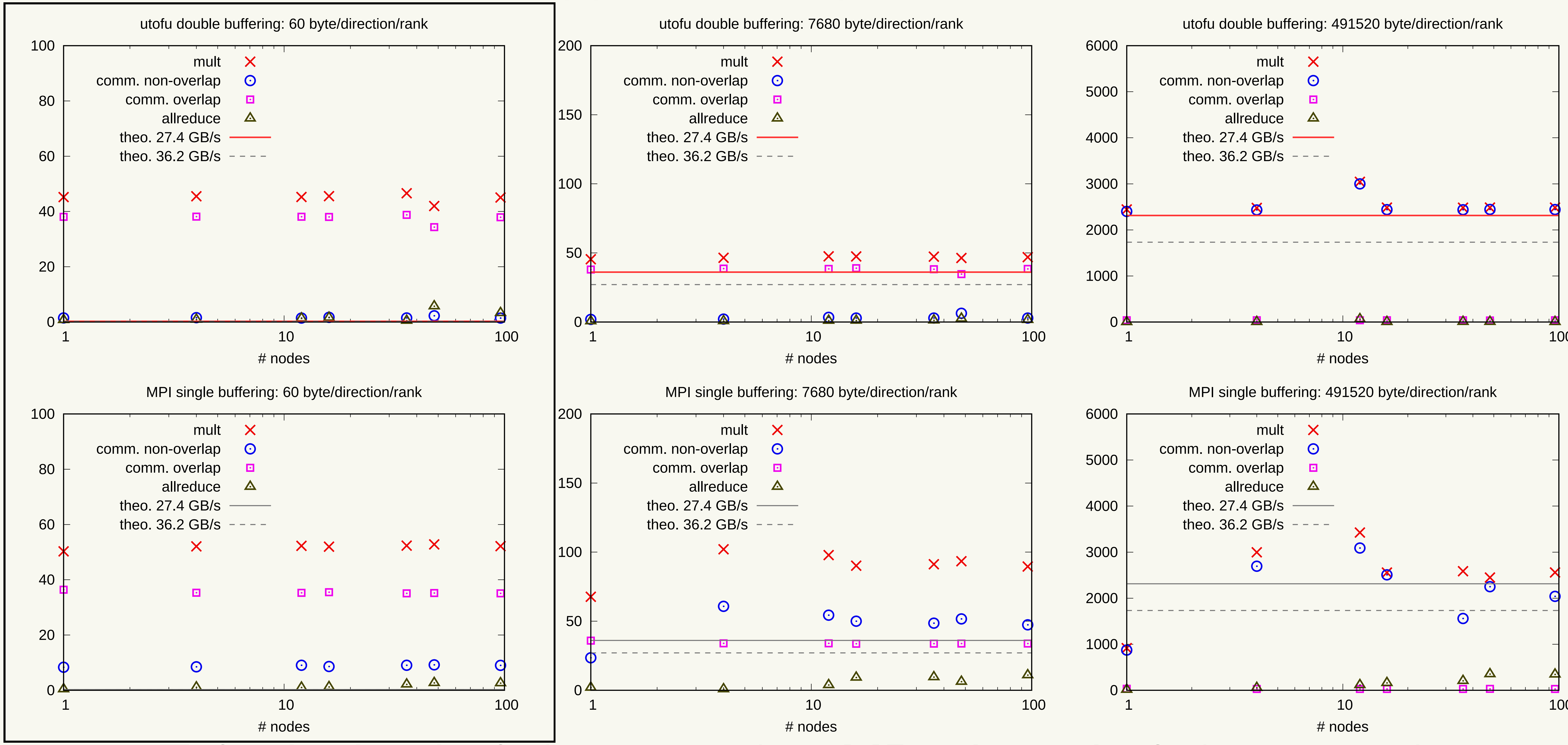
weak scaling with different data size

uTofu double buffering (upper) and mpi single buffering (lower)



weak scaling with different data size

uTofu double buffering (upper) and mpi single buffering (lower)



mult: uTofu double buf. (42 msec.) < MPI single buf. (52 msec.)