Praktikum für verteilte und parallele Systeme
WS 2008/09

MPI on Cell

Zwischenbericht

Lukas Laschitz, Stefan Hofbauer und
Matthias Janetschek

MOC-Implementierung

- Environmental Management
- Groups, Context and Communicators
- Datentypen

Blocking Communication

MOC_Send (void *buf, int count, MOC_Datatype datatype, int dest, int tag, MOC_Comm comm)

MOC_Bsend (void *buf, int count, MOC_Datatype datatype, int dest, int tag, MOC_Comm comm)

MOC_Ssend (void *buf, int count, MOC_Datatype datatype, int dest, int tag, MOC_Comm comm)

MOC_Recv (void *buf, int count, MOC_Datatype datatype, int source, int tag, MOC_Comm comm, MOC_Status *status)

MOC_Probe (int source, int tag, MOC_Comm comm, MOC_Status *status)

MOC_Iprobe (int source, int tag, MOC_Comm comm, int *flag, MOC_Status *status)

Send-Receive

MOC_Sendrecv (void *sendbuf, int sendcount, MOC_Datatype sendtype, int dest, int sendtag,
void *recvbuf, int recvcount, MOC_Datatype recvtype, int source, int recvtag,
MOC_Comm comm, MOC_Status *status)

MOC_Sendrecv_replace (void *buf, int count, MOC_Datatype datatype, int dest, int sendtag, int
source, int recvtag, MOC_Comm comm, MOC_Status *status)

Non-blocking Communication

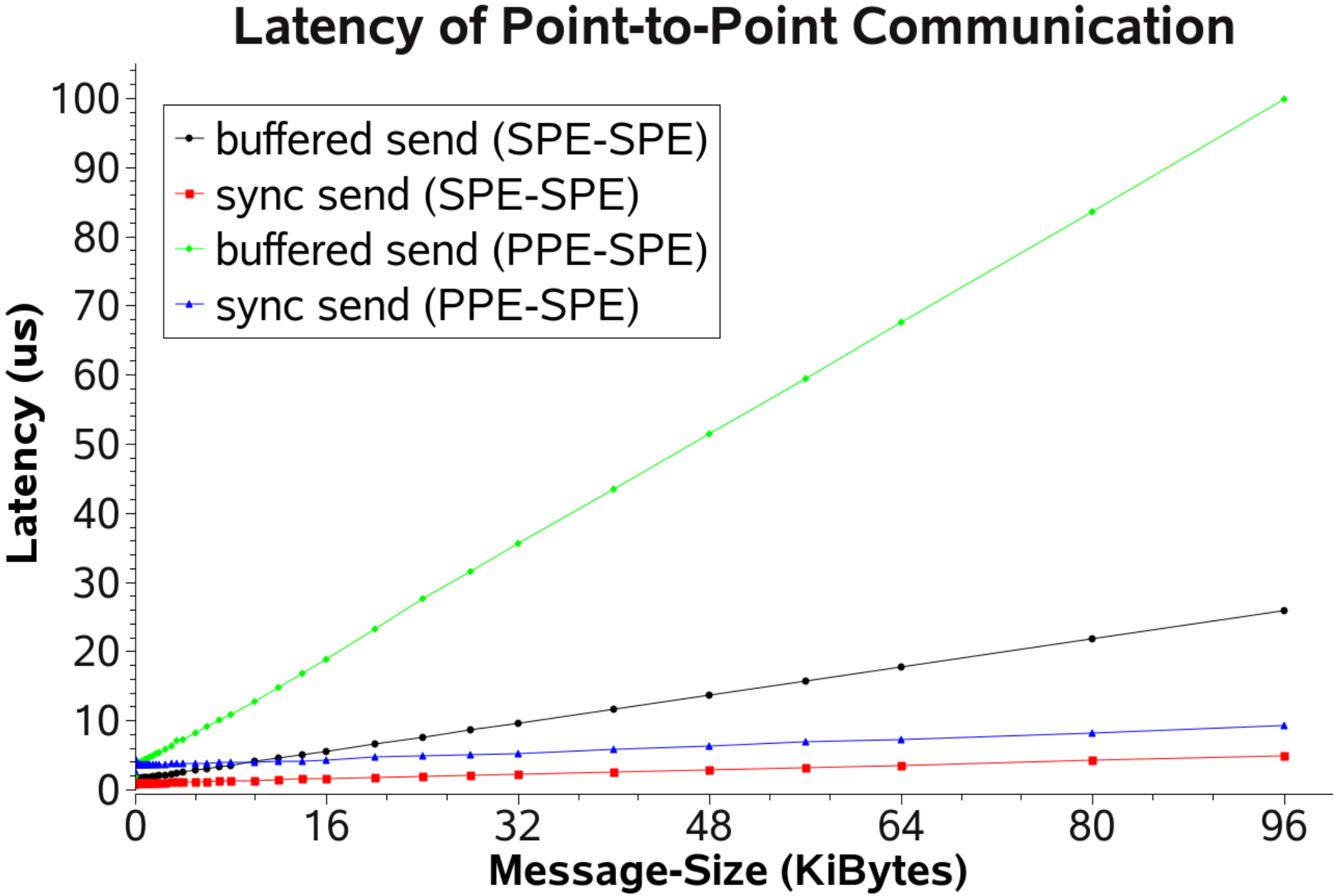
MOC_ISsend (void *buf, int count, MOC_Datatype datatype, int dest, int tag, MOC_Comm comm, MOC_Request *request)

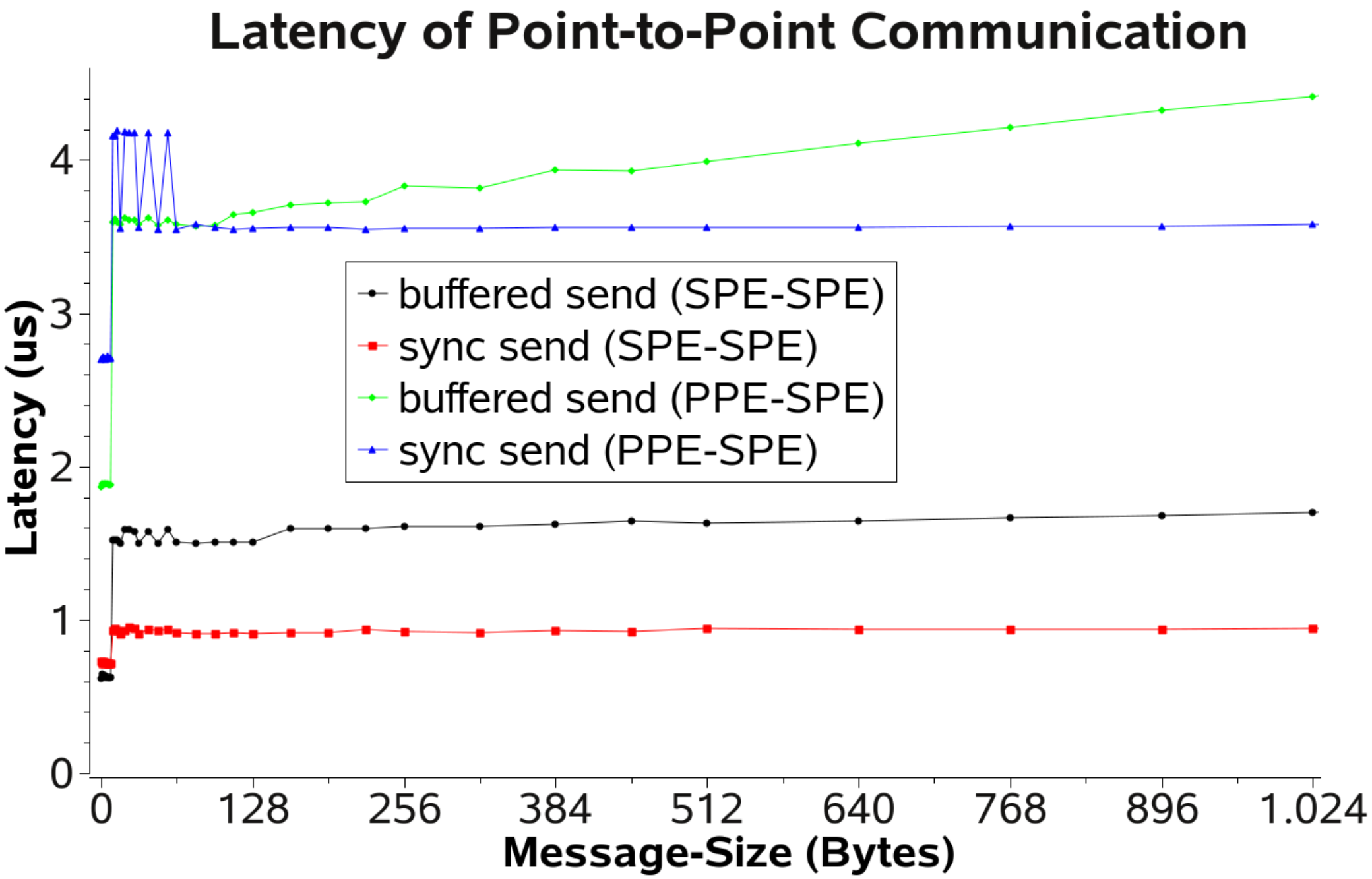
MOC_ISend (void *buf, int count, MOC_Datatype datatype, int dest, int tag, MOC_Comm comm, MOC_Request *request)

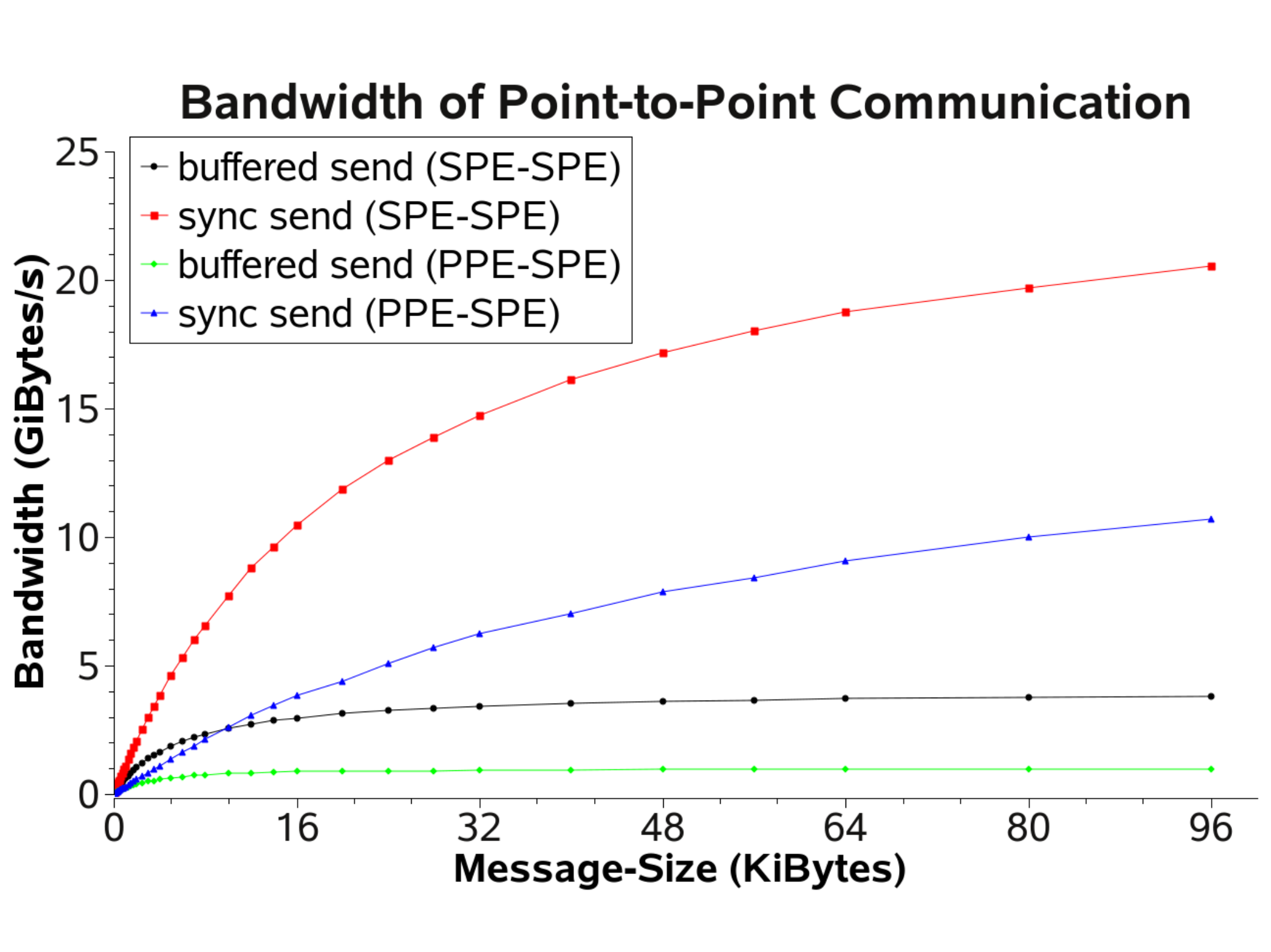
? MOC_IRecv (void *buf, int count, MOC_Datatype datatype, int source, int tag, MOC_Comm comm, MOC_Request *request) ?

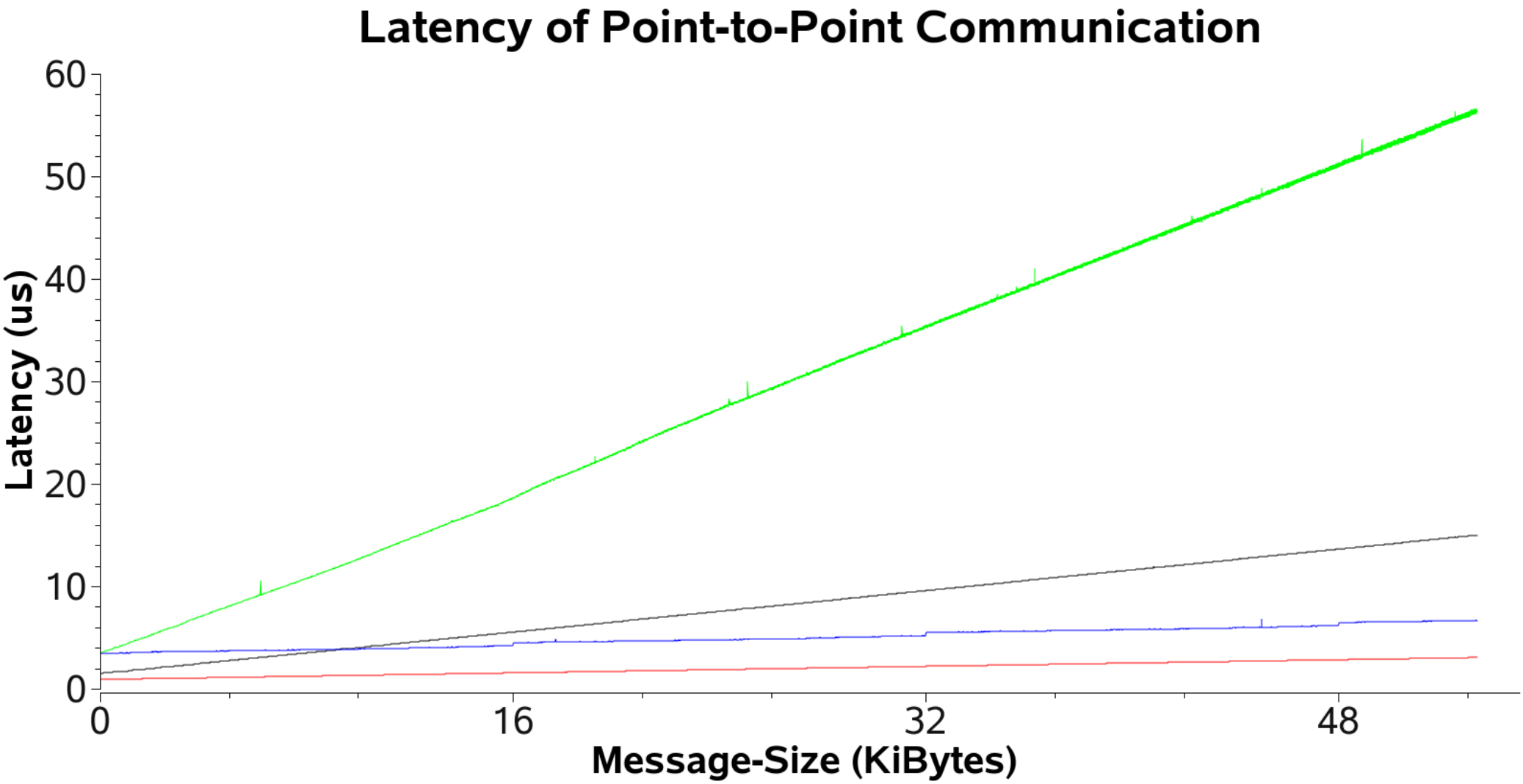
MOC_Test (MOC_Request *request, int *flag, MOC_Status *status)

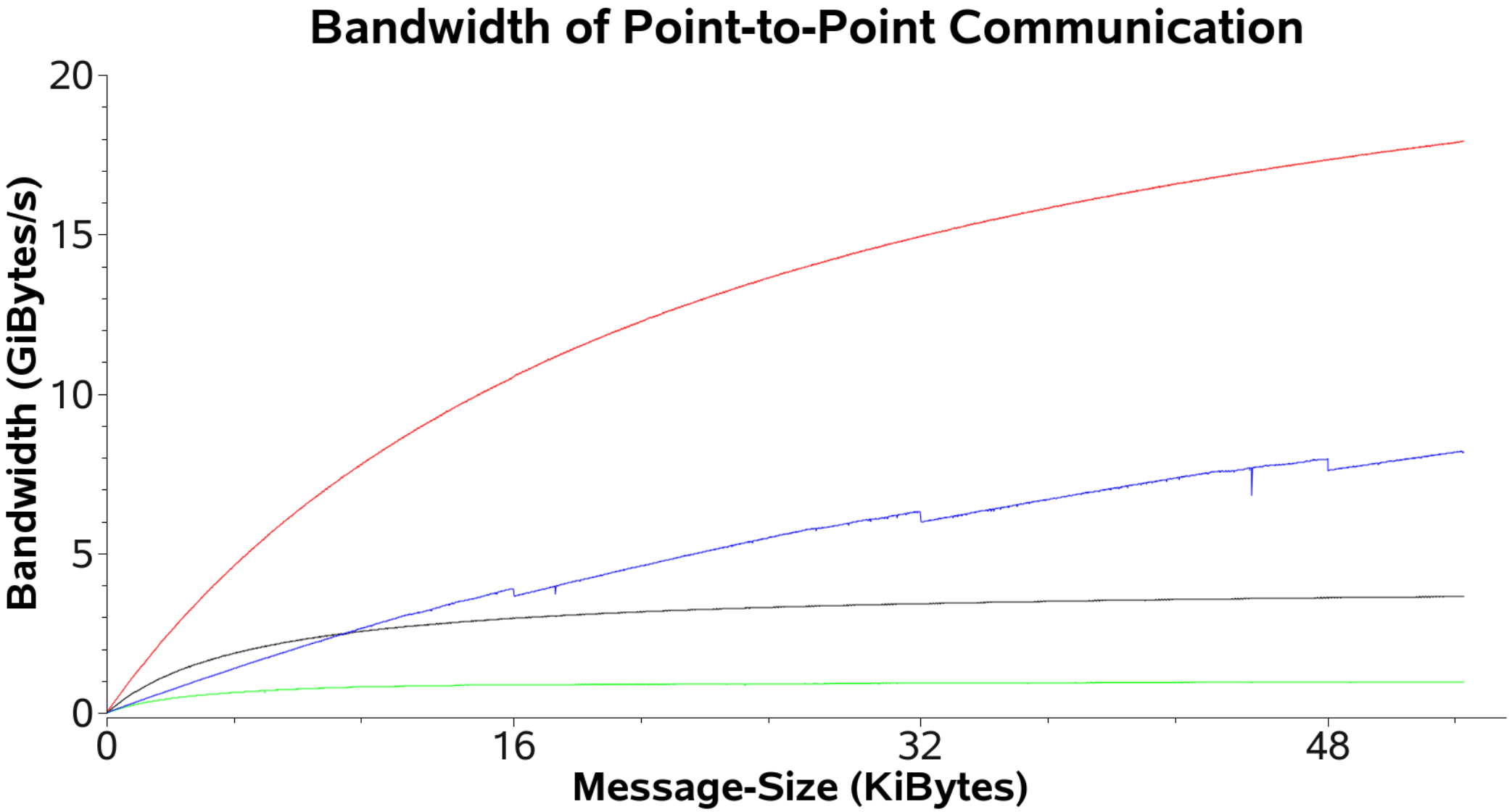
MOC_Wait (MOC_Request *request, MOC_Status *status)

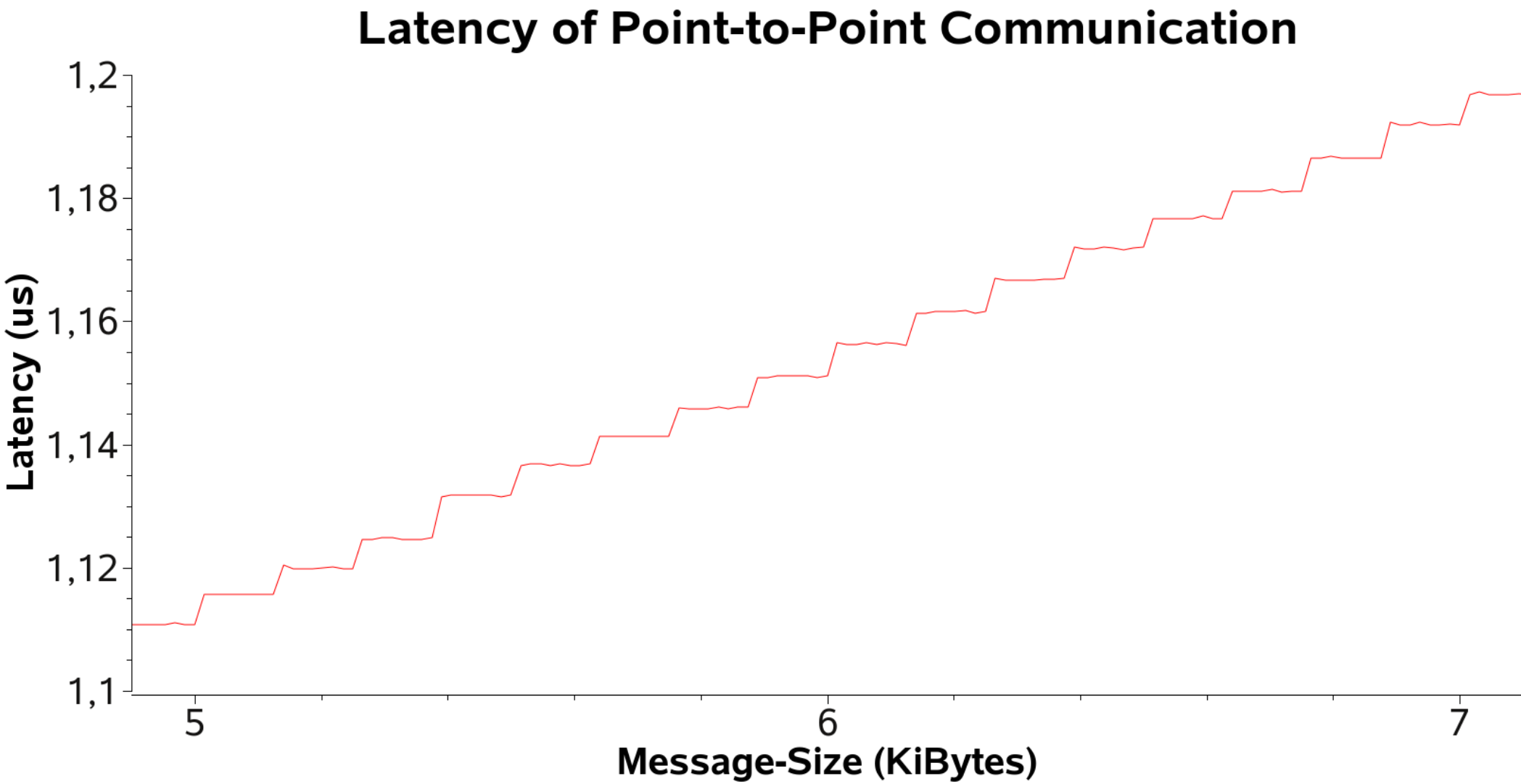

Latency of Point-to-Point Communication
buffered send (SPE-SPE)
sync send (SPE-SPE)
buffered send (PPE-SPE)
sync send (PPE-SPE)
Latency (us)
0
10
20
30
40
50
60
70
80
90
100
0
16
32
48
64
80
96
Message-Size (KiBytes)

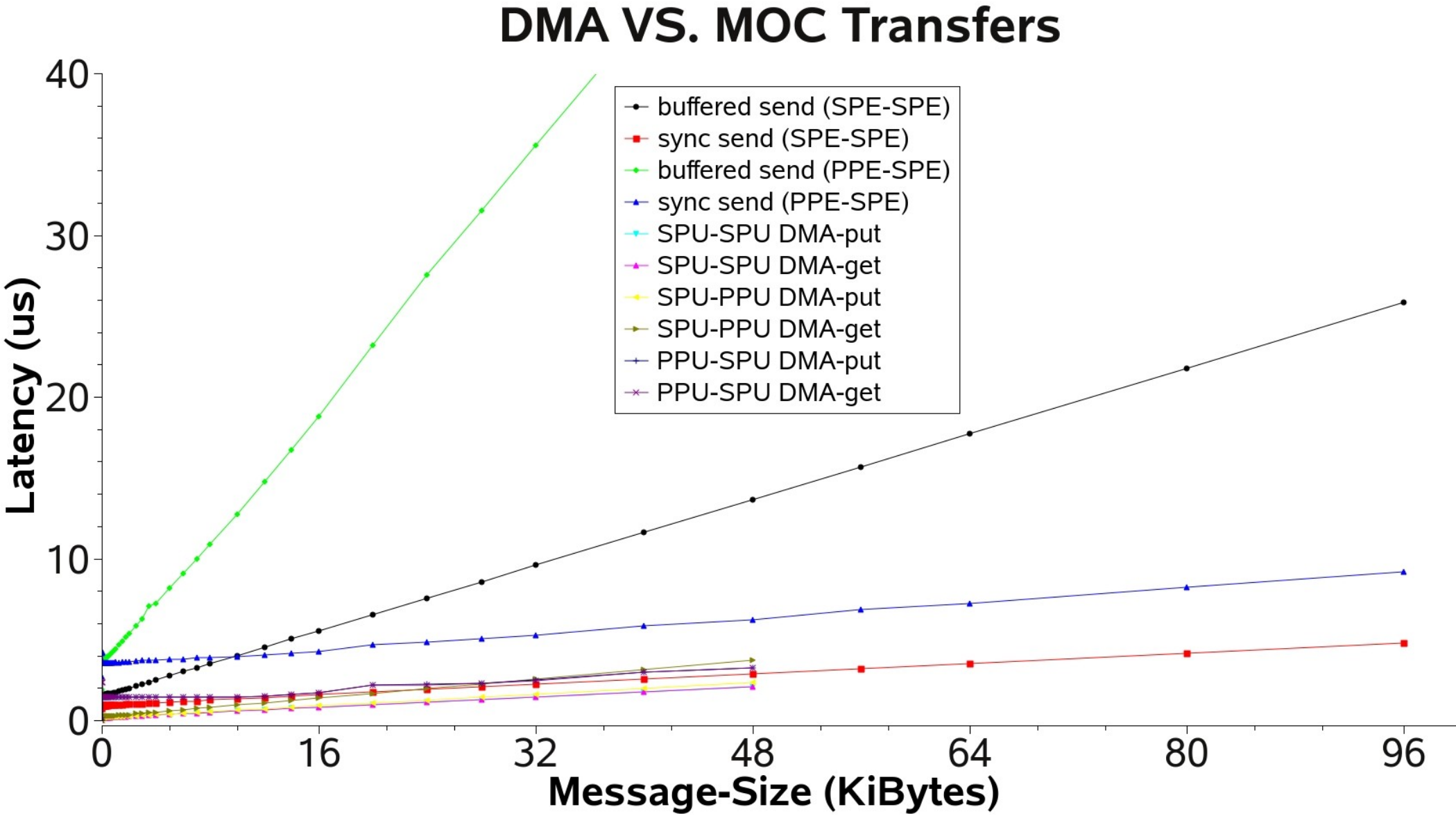

Latency of Point-to-Point Communication
Latency (us)
0
1
2
3
4
buffered send (SPE-SPE)
sync send (SPE-SPE)
buffered send (PPE-SPE)
sync send (PPE-SPE)
0
128
256
384
512
640
768
896
1.024
Message-Size (Bytes)

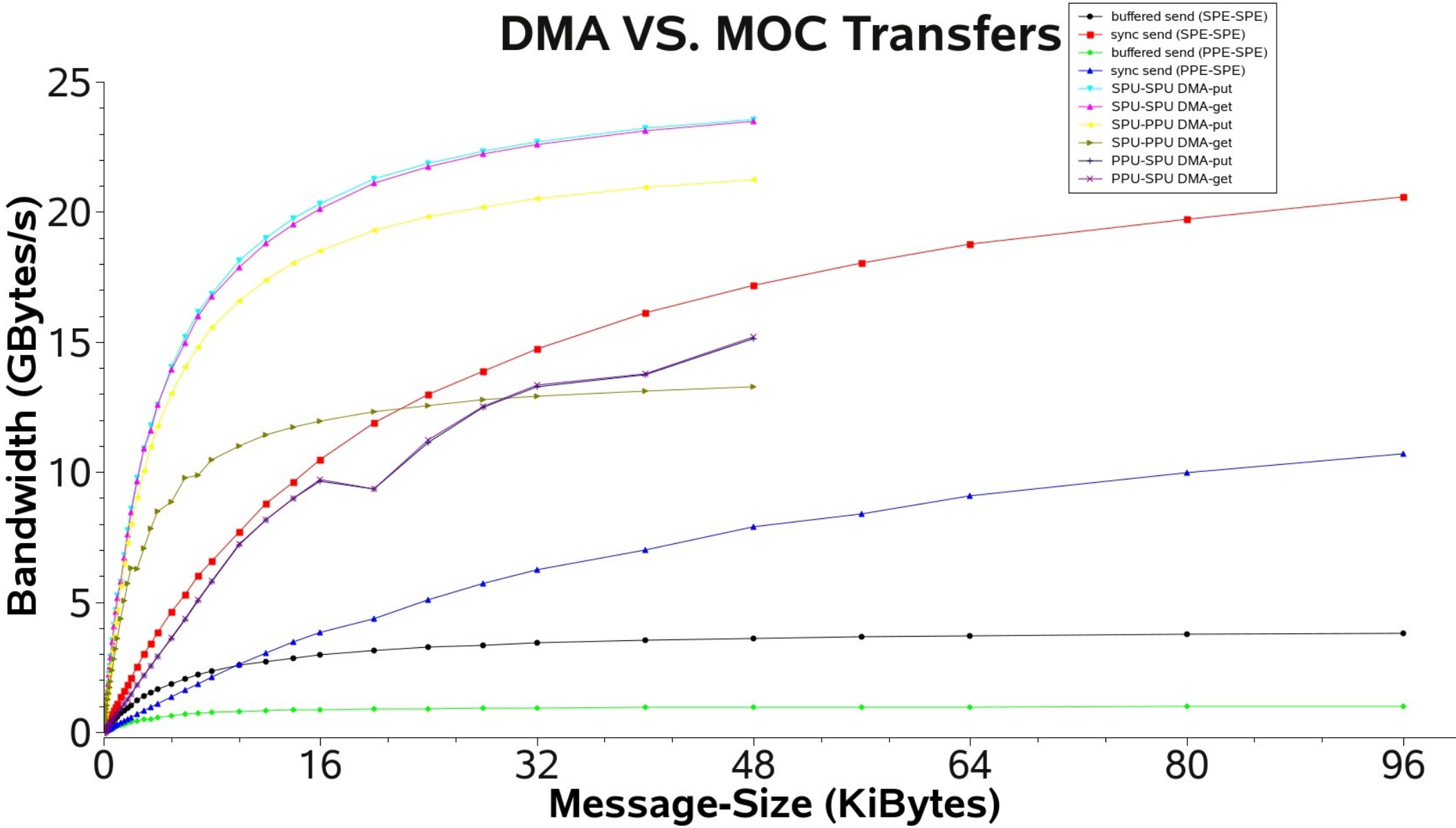

Bandwidth of Point-to-Point Communication
buffered send (SPE-SPE)
sync send (SPE-SPE)
buffered send (PPE-SPE)
sync send (PPE-SPE)
Bandwidth (GiBytes/s)
0
5
10
15
20
25
0
16
32
48
64
80
96
Message-Size (KiBytes)

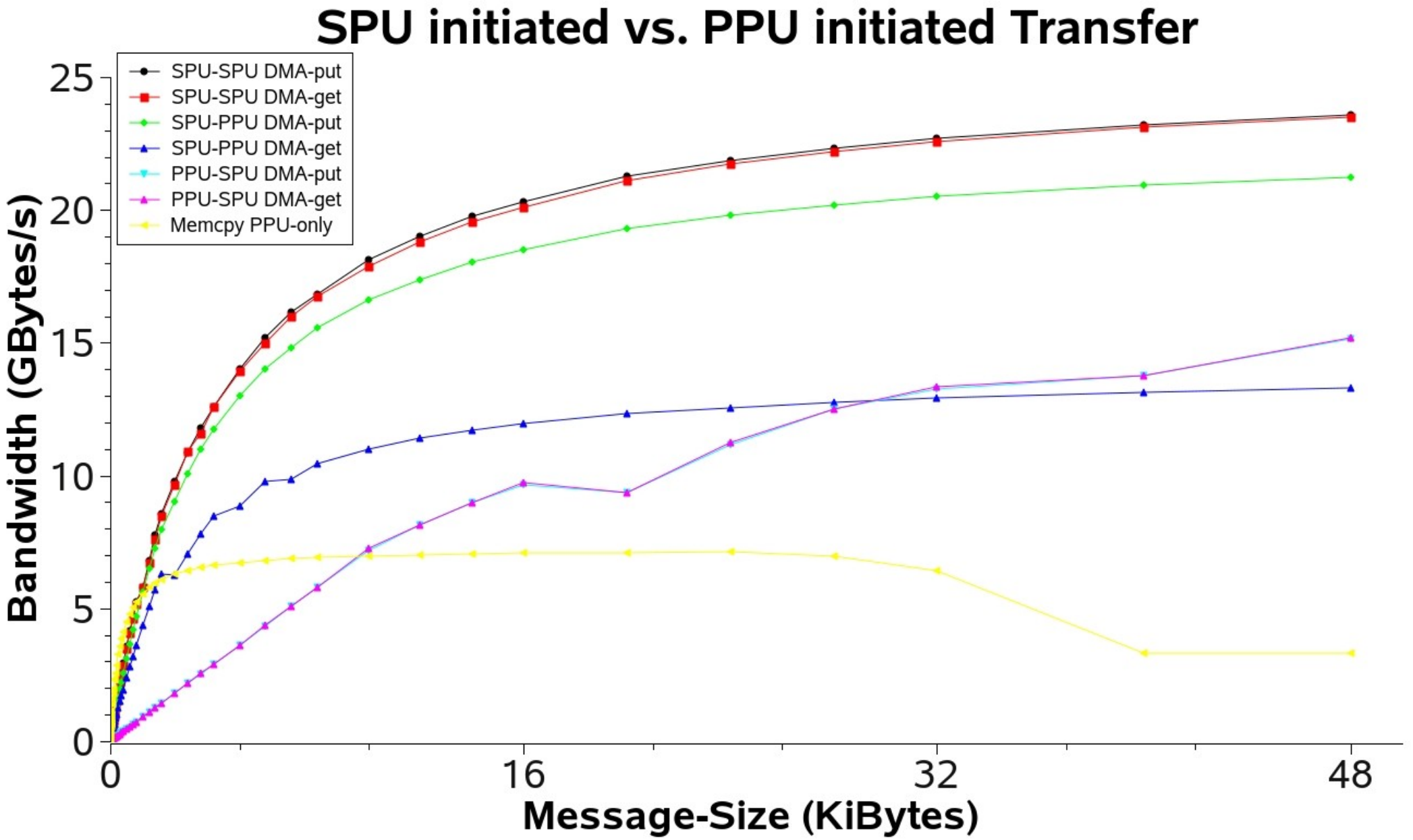

Latency of Point-to-Point Communication
Latency (us)
60
50
40
30
20
10
0
0
16
32
48
Message-Size (KiBytes)

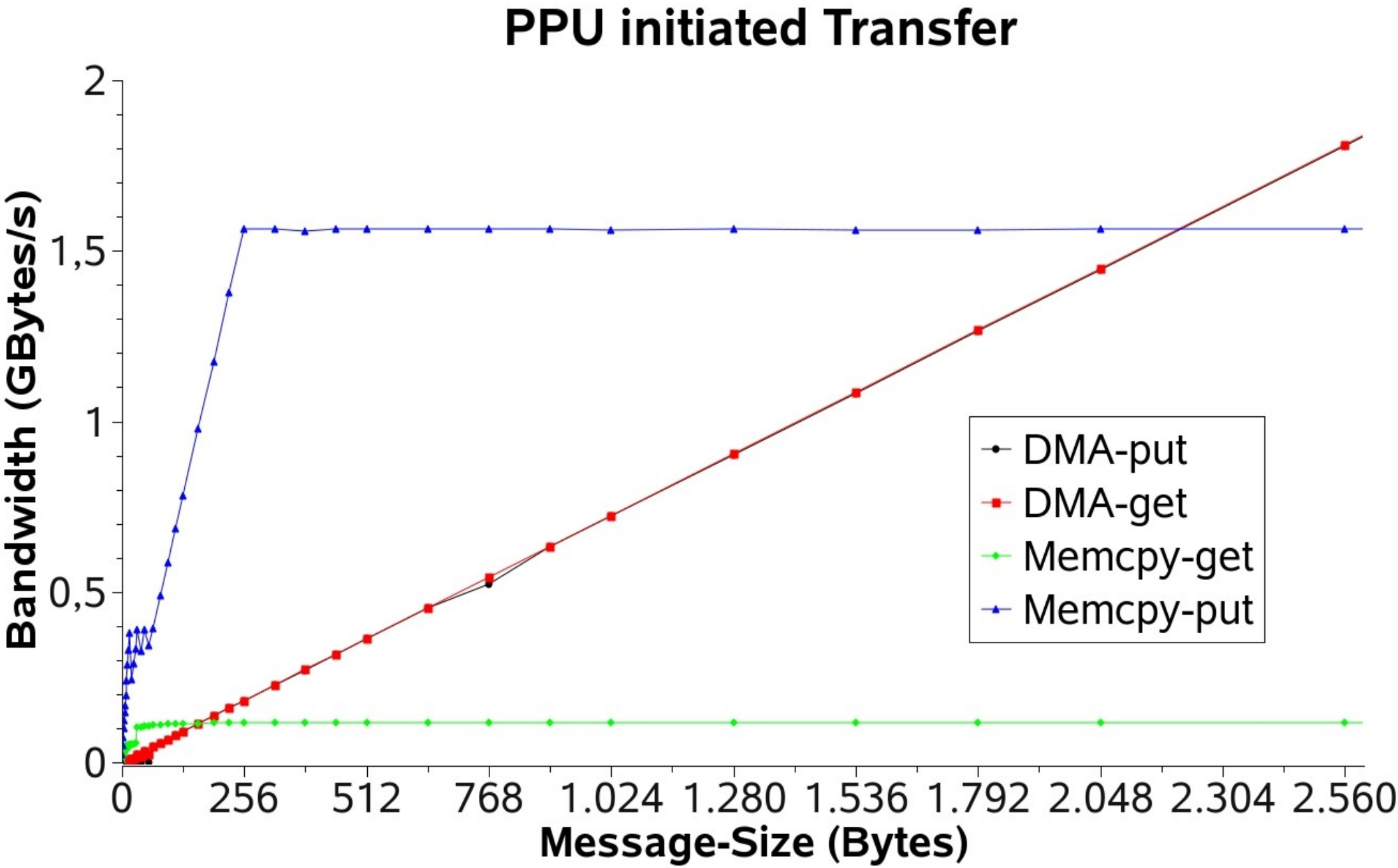

Bandwidth of Point-to-Point Communication
Bandwidth (GiBytes/s)
20
15
10
5
0
0
16
32
48
Message-Size (KiBytes)


Latency of Point-to-Point Communication
Latency (us)
1,2
1,18
1,16
1,14
1,12
1,1
5
6
7
Message-Size (KiBytes)


DMA VS. MOC Transfers
Latency (us)
Message-Size (KiBytes)
0
10
20
30
40
0
16
32
48
64
80
96
buffered send (SPE-SPE)
sync send (SPE-SPE)
buffered send (PPE-SPE)
sync send (PPE-SPE)
SPU-SPU DMA-put
SPU-SPU DMA-get
SPU-PPU DMA-put
SPU-PPU DMA-get
PPU-SPU DMA-put
PPU-SPU DMA-get


DMA VS. MOC Transfers
buffered send (SPE-SPE)
sync send (SPE-SPE)
buffered send (PPE-SPE)
sync send (PPE-SPE)
SPU-SPU DMA-put
SPU-SPU DMA-get
SPU-PPU DMA-put
SPU-PPU DMA-get
PPU-SPU DMA-put
PPU-SPU DMA-get
Bandwidth (GBytes/s)
0
5
10
15
20
25
0
16
32
48
64
80
96
Message-Size (KiBytes)


SPU initiated vs. PPU initiated Transfer
SPU-SPU DMA-put
SPU-SPU DMA-get
SPU-PPU DMA-put
SPU-PPU DMA-get
PPU-SPU DMA-put
PPU-SPU DMA-get
Memcpy PPU-only
Bandwidth (GBytes/s)
0
5
10
15
20
25
0
16
32
48
Message-Size (KiBytes)


PPU initiated Transfer
Bandwidth (GBytes/s)
2
1,5
1
0,5
0
0
256
512
768
1.024
1.280
1.536
1.792
2.048
2.304
2.560
Message-Size (Bytes)
DMA-put
DMA-get
Memcpy-get
Memcpy-put

Konsequenzen

MPI geht von Distributed-Memory aus

CBE ist aber Distributed-Shared-Memory

PPU ist Flaschenhals

==> Hyprid-Modell (MPI mit DMA)

Direct Memory Access

SPU-only:

MOC_Get_data (void *target, uint64_t source, int count, MOC_Datatype datatype)

MOC_Put_data (void *source, uint64_t target, int count, MOC_Datatype datatype)

MOC_IGet_data (void *target, uint64_t source, int count, MOC_Datatype datatype,
MOC_Request *request)

MOC_IPut_data (void *source, uint64_t target, int count, MOC_Datatype datatype,
MOC_Request *request)

MOC_Translate_LS_to_EA (void *ls, uint64_t *ea)

Direct Memory Access

PPU-only:

MOC_Get_data (void *target, uint32_t source, int count, MOC_Datatype datatype, int dest)

MOC_Put_data (void *source, uint32_t target, int count, MOC_Datatype datatype, int src)

MOC_IGet_data (void *target, uint32_t source, int count, MOC_Datatype datatype, int dest,
MOC_Request *request)

MOC_IPut_data (void *source, uint32_t target, int count, MOC_Datatype datatype, int src,
MOC_Request *request)

MOC_Translate_EA_to_LS (uint64_t ea, int rank, uint32_t *ls)

Collective Communication

- PPU == Flaschenhals (aber Daten kommen von PPU und gehen am Ende zur PPU)
- Was Sinn macht:
  - Barrier
  - Broadcast
- Statt Gather / Scatter / ... besser DMA

Danke für die Aufmerksamkeit