# Halo Exchanges and Tausch

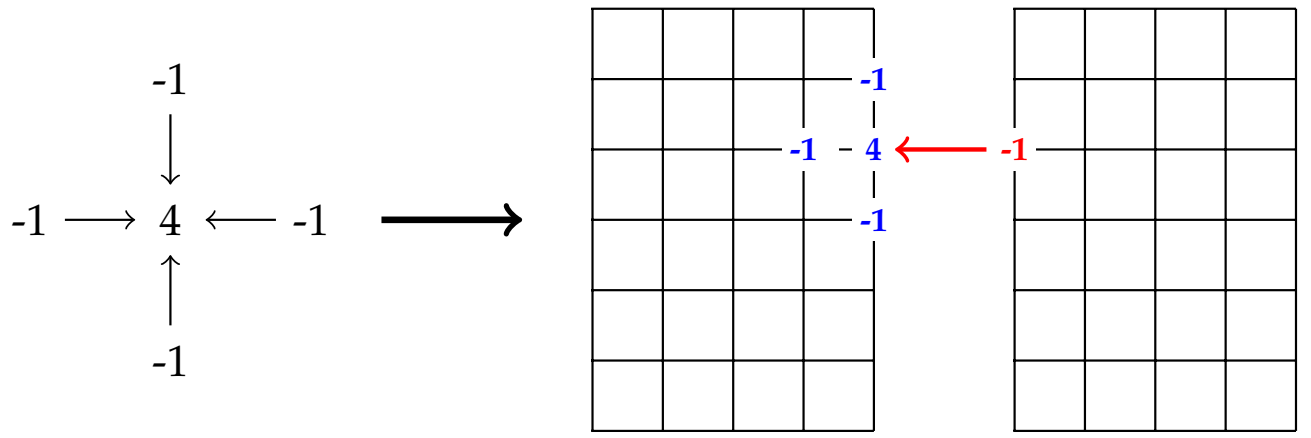
Lukas Spies

Scientific Computing Seminar
University of Illinois

2nd October 2018

## Sample problem:

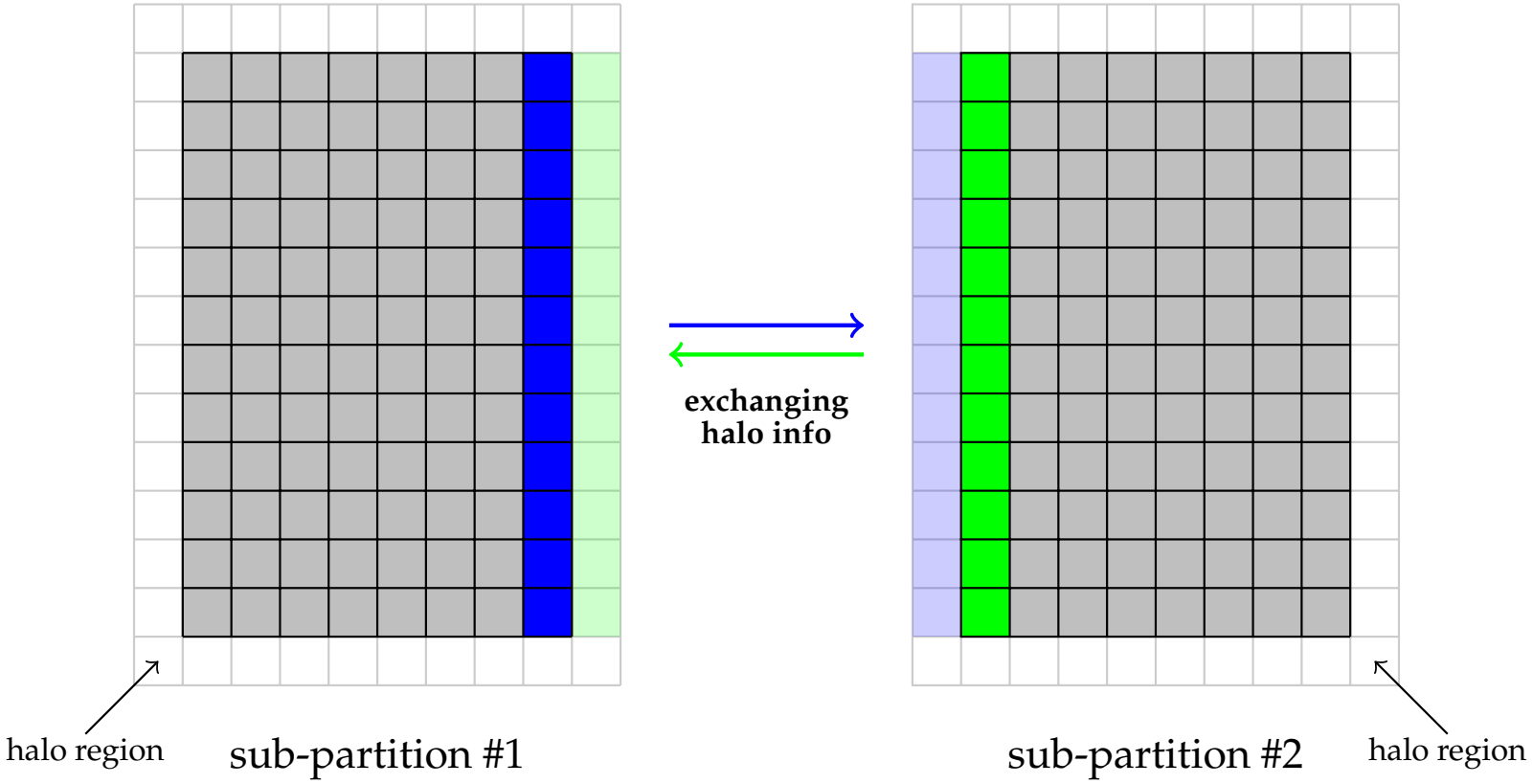
5-point stencil on two sub-partitions of some domain




5-point stencil sub-partition #1 sub-partition #2

Stencil weights known, node values unknown
$\Rightarrow$ Need for communication!

**Solution:** Store extra values as halo around domain




**New problem:** Need to keep updating halos

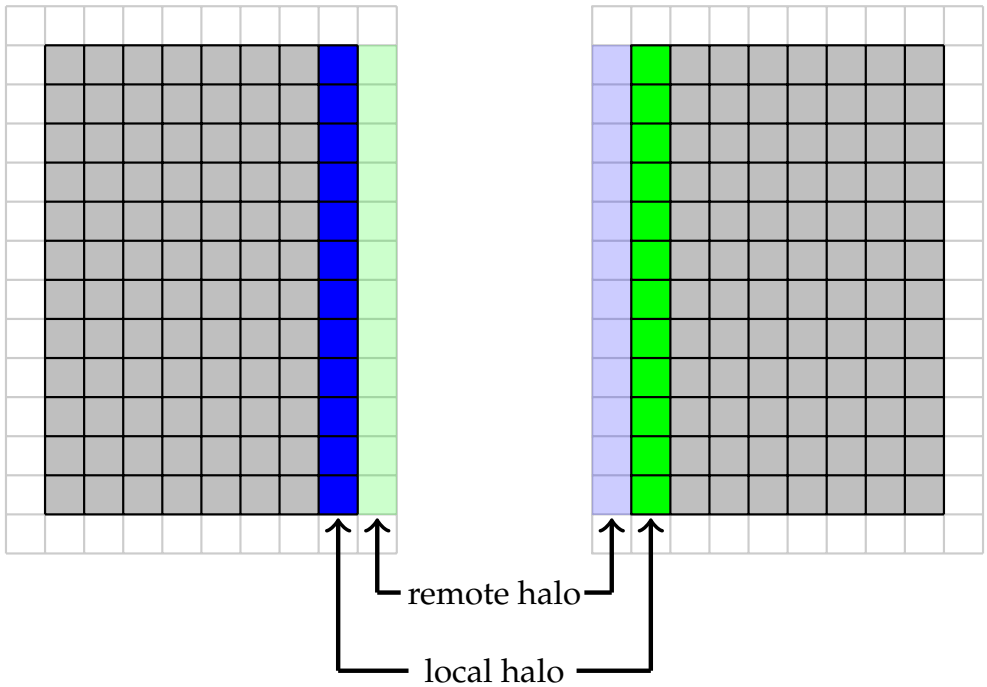
Existing Solutions:

- Libraries like MSG (previously used in Cedar)
  $\rightarrow$ complicated API, not designed for parallel computing
- Custom implementation
  $\rightarrow$ development work needs to be repeated for every project, takes time away from actual development

$\Rightarrow$ Tausch to the rescue!

Glossary:

- **remote halo:** values needed by one sub-partition that are computed on another sub-partition.
- **local halo:** values computed by one sub-partition that are needed by another sub-partition.



Tausch overview:

1. **Defining halo info:**
   addLocalHaloInfo()
   addRemoteHaloInfo()
2. **Extract values of halo:**
   packSendBuffer()
3. **Send/Recv values:**
   send()
   recv()
4. **Move received values back into main buffer:**
   unpackRecvBuffer()

```cpp
#include "tausch.h"

int main(int argc, char** argv) {

    MPI_Init(&argc, &argv);

    Tausch<double> *tausch = new Tausch<double>(MPI_DOUBLE, MPI_COMM_WORLD);

    (...)

    int haloidSend = tausch->addLocalHaloInfo(sendindices);
    int haloidRecv = tausch->addRemoteHaloInfo(recvindices);

    (...)

    tausch->packSendBuffer(haloidSend, 0, buf);
    tausch->send(haloidSend, 1, right);

    tausch->recv(haloidRecv, 1, left);
    tausch->unpackRecvBuffer(haloidRecv, 0, buf);

    (...)

    MPI_Finalize();

    return 0;

}
```

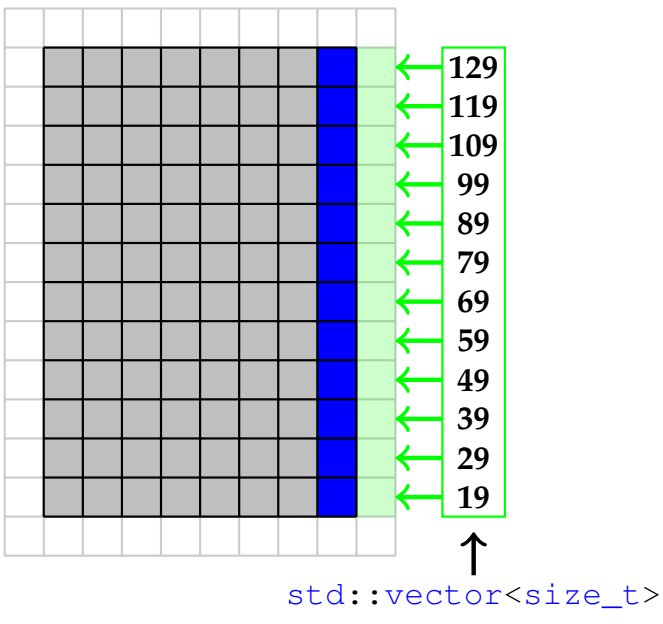
```
...

Tausch<double> *tausch = new Tausch<double>(MPI_DOUBLE, MPI_COMM_WORLD);

...
```

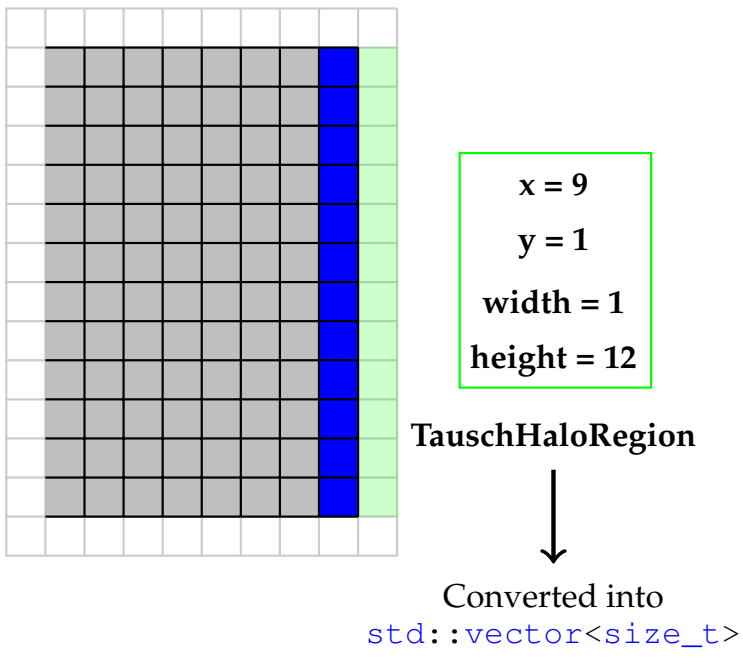
Constructing new Tausch object:

- templated library: can be used with any data type supported by MPI
- To duplicate or not to duplicate communicator

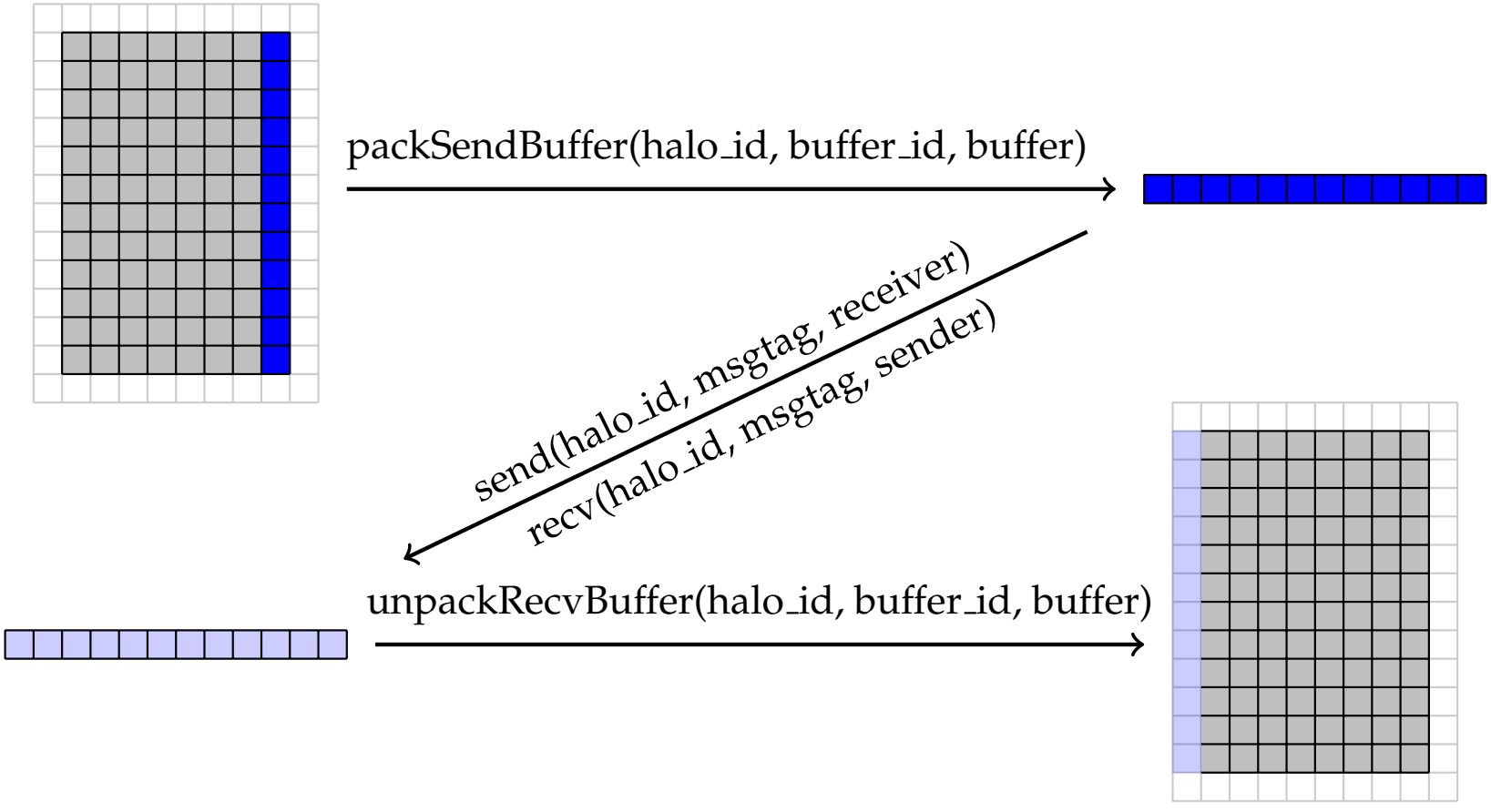
```
...
int haloidSend = tausch->addLocalHaloInfo(sendindices);
int haloidRecv = tausch->addRemoteHaloInfo(recvindices);
...
```

# Two ways to specify halo indices:



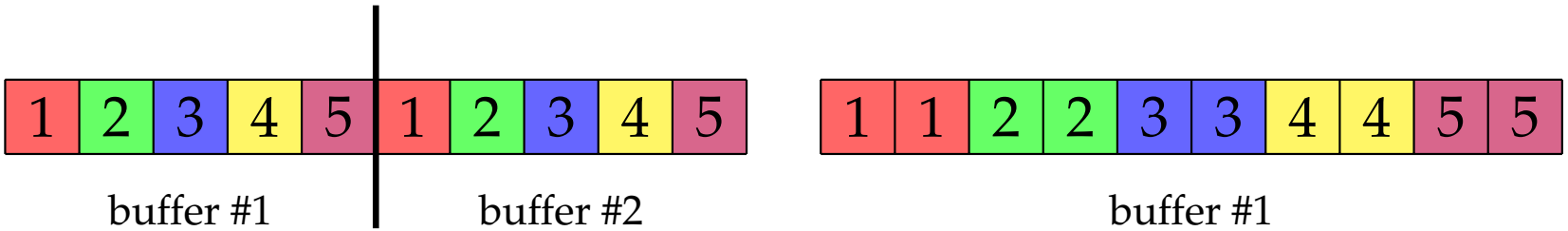



```
...
tausch->packSendBuffer(haloidSend, 0, buf);
tausch->send(haloidSend, 1, right);

tausch->recv(haloidRecv, 1, left);
tausch->unpackRecvBuffer(haloidRecv, 0, buf);
...
```



Some design considerations:

- Using MPI persistent communication decreases overhead on repeated calls to send/recv.
- Multiple buffers can be combined into one message
- Multiple values per point can be combined into one message
- Aligned memory buffers



Performance model in 2D:

$$\mathbf{T_{perf} = 2nT_{b2h,nseq} + 2nT_{b2h,seq} + 4nT_{MPI,send/recv} + 2nT_{h2b,seq} + 2nT_{h2b,nseq}}$$

$T_{perf}$ := Performance prediction

$n$ := number of points in one dimension

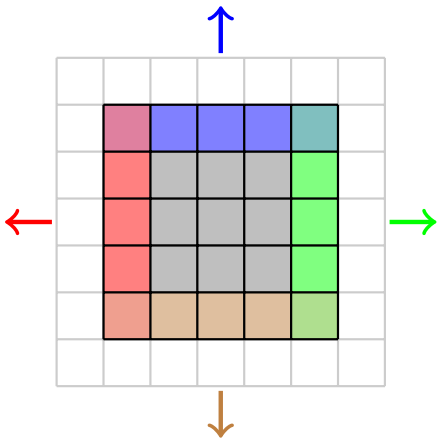
$T_{b2h,seq}$ := Time to copy one value from buffer to halo, sequential memory access

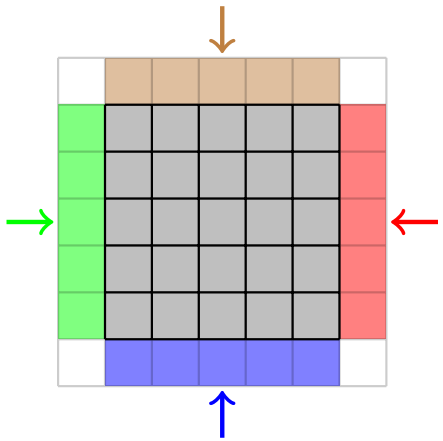
$T_{b2h,nseq}$ := Time to copy one value from buffer to halo, non-sequential memory access

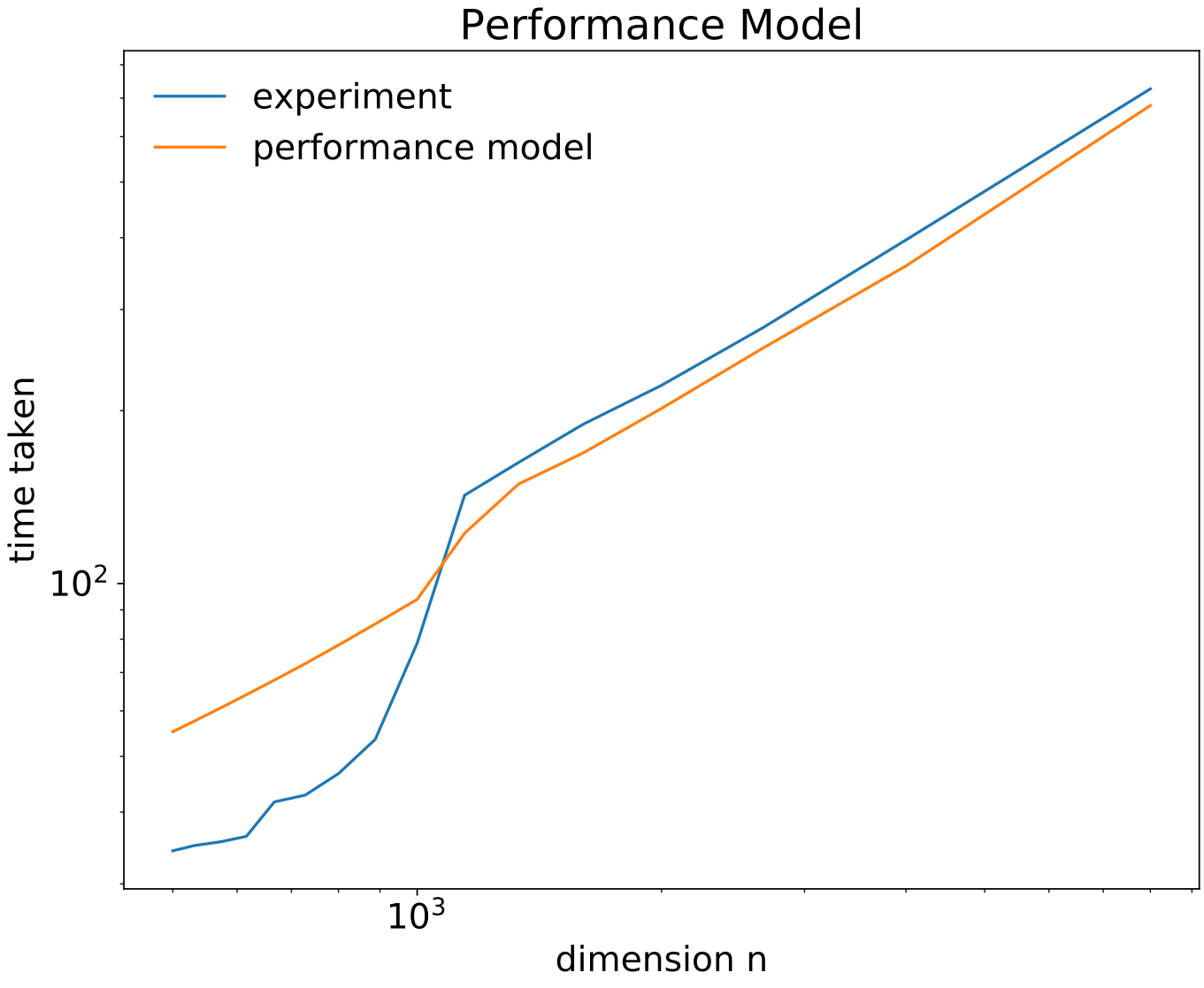
$T_{h2b,seq}$ := Time to copy one value from halo to buffer, sequential memory access

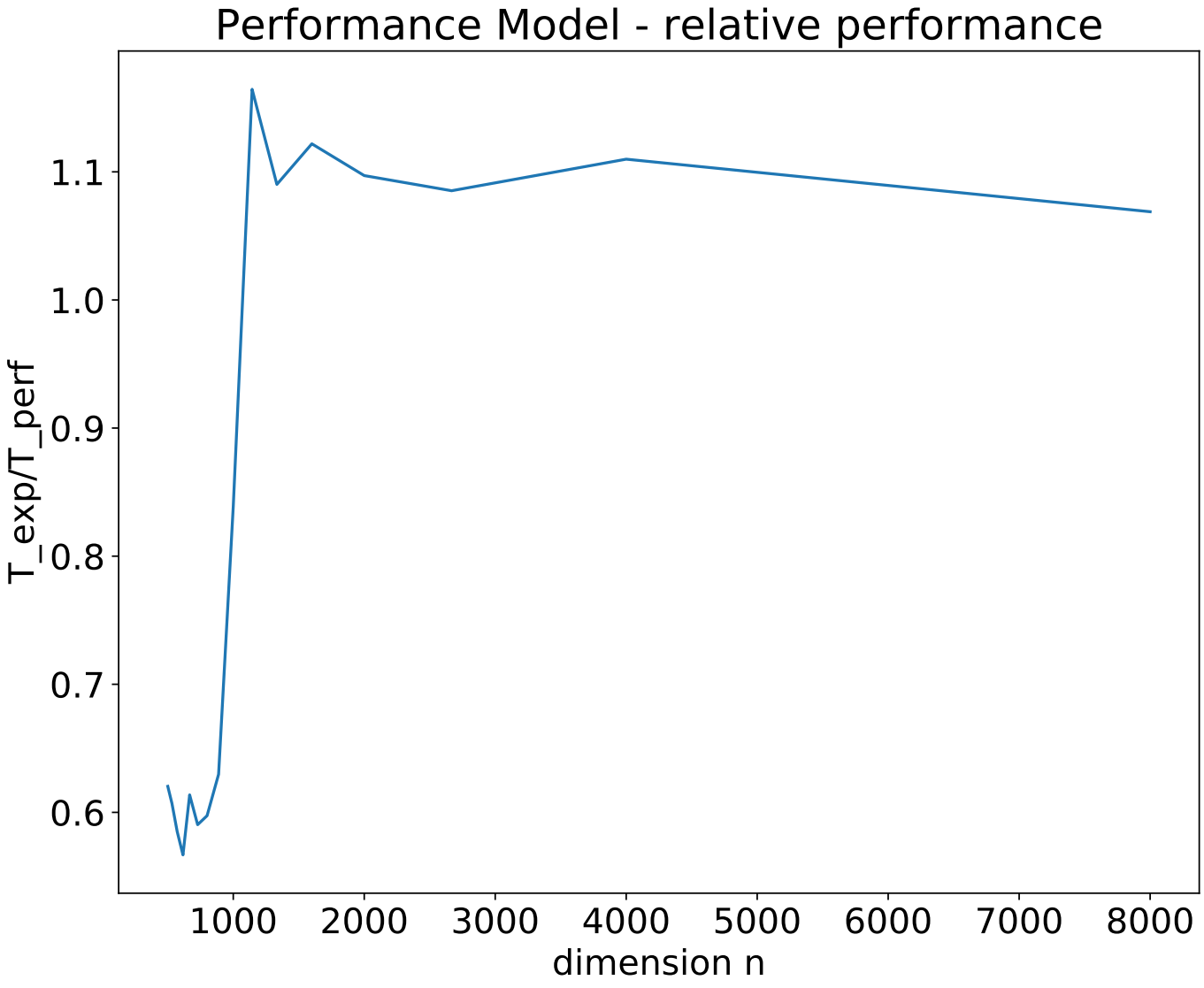
$T_{h2b,nseq}$ := Time to copy one value from halo to buffer, non-sequential memory access

$T_{MPI,send/recv}$ := Time to send one value using MPI communication






Performance Model
experiment
performance model
time taken
$10^2$
$10^3$
dimension n


Performance Model - relative performance
T_exp/T_perf
1.1
1.0
0.9
0.8
0.7
0.6
1000
2000
3000
4000
5000
6000
7000
8000
dimension n

Current status and future work:

- Tausch is used already by Cedar, integration into PlasCom2 in progress
- Performance model demonstrates achievement of expected performance
- API very simple and straightforward $\rightarrow$ easy integration
- CPU/GPU communication still incomplete/unverified
- Small paper planned