The Role of Idle Waves in Modeling and Optimization of Parallel Programs

Ayesha Afzal$^{1,2}$, Georg Hager$^1$, Gerhard Wellein$^{1,2}$

$^1$Erlangen National High Performance Computing Center
$^2$Department of Computer Science, University of Erlangen-Nürnberg Germany

NHR PerfLab Seminar
April 26, 2022
Challenges of white-box performance modelling in HPC

Vision: white-box first-principle performance modelling

Why it’s not a realistic model?

Why it can go in either direction?
Challenges of white-box performance modelling in HPC

It’s intricated (bottlenecks interact, systems are noisy, etc.)
Motivation

Process 0

\( T_{\text{exec}} \) | \( T_{\text{non-exec}} \) | \( T_{\text{exec}} \) | \( T_{\text{non-exec}} \) | \( \cdots \)

Process 1

\( T_{\text{exec}} \) | \( T_{\text{non-exec}} \) | \( T_{\text{exec}} \) | \( T_{\text{non-exec}} \) | \( \cdots \)

“Lock-step” behavior at start

No network contention or load imbalance

Memory-bound MPI-parallel programs: timeline view
Motivation

Memory-bound MPI-parallel programs: timeline view

“Lock-step” behavior at start
No network contention or load imbalance

“Snap-in” desynchronization with communication overlap after some evolution
Motivating example (1)

Time step 1

Wall-clock time [s]

Computational wavefront

MPI parallel Lattice Boltzmann fluid solver

302^3 lattice cell, 8 GB data set, non-blocking, 1D domain decomposition, distance-1 communication, 10 Emmy@RRZE sockets @2.2 GHz
Motivating example (2)

MPI parallel STREAM TRIAD

Wall-clock time normalized to slowest process

Spontaneous symmetry breaking, communication overlap, why?
Under what conditions?

Emmy@RRZE sockets @2.2 GHz, non-temporal stores, bi-dir, open chain, distance-1 communication

3.1 Execution characteristics
HPC workloads have a wide spectrum of requirements regarding code execution towards resources of the parallel computing platform. The most straightforward categorization is whether the workload is sensitive to certain resource bottlenecks, such as memory bandwidth. Since we restrict ourselves to scalable code here, we run the traditionally memory-bound algorithms such as stencil updates or SpMV with one MPI process per contention domain (typically a ccNUMA node). This is not a problem for the microbenchmarks since we deliberately choose an in-core workload there.

3.2 Categorization of communication characteristics
Here we briefly describe the different communication characteristics under investigation. We start by assuming a “P2P-homogeneous” situation where all processes (except boundary processes in case of open boundary conditions) have the same communication partners and characteristics. We will later lift this restriction and cover more general patterns.

Communication topology
Communication topology is a consequence of the physical problem underlying the numerical method and of the algorithm (discretization, geometry). It boils down to the question “which other processes does rank $i$ communicate with?” and is characterized by a topology matrix (see Figure 1 for examples of compact and noncompact topologies).

In a compact topology, each process communicates with a dense, continuous array of neighbors with distances $d = \pm 1, \pm 2, ..., \pm j$. The topology matrix comprises a dense band around the main diagonal. In a noncompact topology, each process communicates with processes that are not arranged as a continuous block, e.g., $d = \pm 1, \pm j$. In both variants, the topology matrix can be symmetric or asymmetric.

$P_i$
Node 0
Socket 0
$P_{i-1}$
Node 1
Socket 1
$P_{i+1}$

Fig. 1: Compact and non-compact communication topologies with bidirectional open chain characteristics.
Lessons learned (1): Impact of idle wave on overlap

Propagation speed of idle wave

Overlap amplified / damped?

\[ T_{\text{exec}} \quad T_{\text{non-exec}} \quad T_{\text{exec}} \quad T_{\text{non-exec}} \]

\[ T_{\text{exec}} \quad T_{\text{non-exec}} \quad T_{\text{exec}} \quad T_{\text{non-exec}} \]
Idle wave: delay propagation across processes in each iteration

\[ v_{\text{min}} = \kappa \times \sigma \times \frac{1}{\text{iter}} \times \frac{1}{T_{\text{comp}} + T_{\text{comm}}} \times \frac{\text{iter}}{s} \]

\[ \kappa = \frac{j(j+1)}{2} \text{ or } (j + |i|) \text{ or } j \]

\[ j/i: \text{ longest /shorter-distance partner} \]

DOI: 10.1103/PhysRevE.91.013306

resource-scalable MPI programs
Communication pattern and concurrency: compact

\[ \text{Dimensionless } \kappa \]

distance in processes travelled in one time step by the idle wave
Communication pattern and concurrency: compact

Dimensionless \( \kappa \) distance in processes travelled in one time step by the idle wave

Ayesha Afzal <ayesha.afzal@fau.de> | ISC High Performance 2021 Digital

June 30, 2021
mark is popular for ranking supercomputers beyond the ubiquitous LINPACK. Here we

Figure

largest smaller integer (© studies covered so far. It turns out that all connections apart from the longest-distance case (b) where the topology matrix is narrower.

a double-precision Jacobi smoother using Cartesian domain decomposition and two

Stencil smoother with halo exchange

MWSDir.

ference between eager and rendezvous mode does not impact the other variants beyond the innermost loop. In all these cases, the wave propagation speed doubles in rendezvous mode. However, in the single-wavecase (MWMDim) this leads to a double-precision Jacobi smoother using Cartesian domain decomposition and two

Stencil smoother with halo exchange

MWSDir.

ference between eager and rendezvous mode does not impact the other variants beyond the innermost loop. In all these cases, the wave propagation speed doubles in rendezvous mode. However, in the single-wavecase (MWMDim) this leads to

the propagation speed is

\[ \text{propagation speed} = \frac{\text{distance}}{\text{time}} \]

factor. For the case in

we restrict ourselves to a manageable subset of options that nevertheless cover a

more general

More complex patterns

fourth option are corner cases with minimum and maximum number of

the two directions per dimension (i.e., positive and negative). For each direction (e.g.,

ber of split-waits:

while

end while

MPI_Waitall

MPI_Irecv

MPI_Send

MPI_Isend

MPI_Irecv

MPI_Sendrecv

1: while \( d \leq \text{dims} \) do
2: \( \text{while } \text{dir} \leq \text{bi} \) do
3: \( \text{MPI}_\text{Isend} \)elfare
4: \( \text{MPI}_\text{Irecv} \)elfare
5: \( \text{end while} \)
6: \( \text{end while} \)
7: \( \text{MPI}_\text{Waitall} \)

Refraction effect

30 60 90 119
0
10
20
30
40
50
60
70
80
90
100
110
119
Receiver rank
Sender rank
(a)

1.4 1.6 1.8 2 2.2 2.4
0
5
10
20
30
40
50
60
70
80
90
100
110
119
Time [s]
Rank
0
50
119
N AMASS
(b)
Idle wave propagation: 3D Jacobi smoother

In practice, the elimination or the survival of the wave may be desirable depending on wave progression. Orange color shows with Cartesian domain decomposition (interior processes), and MWSDir concurrency applies just like in the stencil example.

The per-process problem size is small enough for eager mode, but communication time is a relevant contribution to the overall runtime.

Few MPI programs use point-to-point communications only. Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

For collective communications are certainly heavily permeable to a traveling wave. The number of communication partners varies between

In case (a) we get

For (b) we get

For (c) we get

The effects we discuss here are summarized in Figure 6.

It was shown that idle waves can lead to automatic

The results are summarized in Figure 5.

Few MPI programs use point-to-point communications only. Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be

Concerning idle wave propagation during multiple back-to-back sparse matrix-vector multiplications using the HPCG matrix, which emerges from a sparse linear system, the question arises which collective routines may be
Idle wave propagation: 3D Jacobi smoother

One MPI process per ccNUMA
Process grid: 2 x 6 x 10

Time [s]

Rank

Sender rank

Receiver rank

Msg [KB]

One MPI process per ccNUMA
Process grid: 2 x 6 x 10
Idle wave propagation: sparse MVM with HPCG matrix

Top row: topology matrix picturing a gather operation, is also applicable to documented settings are available in case of communication volumes. Bottom row: problem size of HPCG matrix with a multiplication (SpMV) using the

Globally synchronizing primitives (symmetry across main diagonals) and bidirectional halo exchange are necessarily synchronizing, and indeed the idle wave can pass the collective, which appears like a global, compact communication block through which the wave travels with maximum speed.

Examples of necessarily synchronizing collectives (see the discussion of inhomogeneous communication above).

With Intel MPI, the environment variable can be set to a value that selects a particular implementation variant for the collective. Eleven 

Examples of necessarily synchronizing collectives (see the discussion of inhomogeneous communication above).

Finally, Figure 3 illustrates how the interaction of 

Globally synchronizing primitives (symmetry across main diagonals) and bidirectional halo exchange are necessarily synchronizing, and indeed the idle wave can pass the collective, which appears like a global, compact communication block through which the wave travels with maximum speed.

Examples of necessarily synchronizing collectives (see the discussion of inhomogeneous communication above).

With Intel MPI, the environment variable can be set to a value that selects a particular implementation variant for the collective. Eleven
Idle wave propagation: sparse MVM with HPCG matrix

Examples of necessarily synchronizing collectives
- Reductions
- Gather
- Scatter
- Broadcast

Examples of non-synchronizing collectives
- Gather
- Scatter
- Broadcast

If the survival of idle waves is desirable, avoid synchronizing collective calls.

The process of finding the optimal parameter settings can be automated.
Idle wave propagation: sparse MVM with HPCG matrix

One MPI process per coreNUMA
Process grid: 4 x 5 x 5

Time [ms]

Receiver rank M

Sender rank

Rank

Msg [B]
Idle wave propagation: Adaptive Mesh Refinement - miniAMR

![Graph of communication patterns over time]

- Number of total communication partners
- Number of unique communication partners

Sender rank vs Receiver rank over Time intervals:

- Left Side: Sender rank vs Receiver rank
- Middle: Time profile
- Right Side: Receiver rank vs Time
Lessons learned: Impact of idle wave on overlap

Afzal et al.

DOI:10.1109/CLUSTER.2019.8890995

IEEE Cluster 2019

Propagation speed of idle wave

Overlap amplified / damped?

$T_{\text{exec}}$ $T_{\text{non-exec}}$ $T_{\text{exec}}$ $T_{\text{non-exec}}$

$T_{\text{exec}}$ $T_{\text{non-exec}}$ $T_{\text{exec}}$ $T_{\text{non-exec}}$

$T_{\text{exec}}$ $T_{\text{non-exec}}$ $T_{\text{exec}}$ $T_{\text{non-exec}}$
Lessons learned: Impact of idle wave on overlap

Propagation speed of idle wave

Overlap amplified / damped?

\[ T_{\text{exec}} \quad T_{\text{non-exec}} \quad T_{\text{exec}} \quad T_{\text{non-exec}} \ldots \]

\[ T_{\text{exec}} \quad T_{\text{non-exec}} \quad T_{\text{exec}} \quad T_{\text{non-exec}} \ldots \]

Triple idle wave speed

Wall-clock time [s]

-40 rank/s, \( r = 0.999 \)

+121 rank/s, \( r = 0.968 \)
Lessons learned: Impact of idle wave on overlap

Afzal et al.

DOI:10.1109/CLUSTER.2019.8890995

IEEE Cluster 2019

Propagation speed of idle wave

Overlap amplified / damped?

Idle wave decay

$T_{\text{exec}}$ $T_{\text{non-exec}}$ $T_{\text{exec}}$ $T_{\text{non-exec}}$

$T_{\text{exec}}$ $T_{\text{non-exec}}$ $T_{\text{exec}}$ $T_{\text{non-exec}}$
Non-linear interactions of idle wave

(a) equal  
(b) half  
(c) random
Non-linear interactions of idle wave with system topology

1 KiB Msg

Emmy
2 x 1 x 10
(12 domains)

SuperMUC-NG
2 x 1 x 24
(5 domains)

Hawk
2 x 4 x 16
(30 domains)

Single Node:
sockets per node x
ccNUMA domain per socket x
physical cores per domain

Sender rank

Receiver rank

Ayesha Afzal <ayesha.afzal@fau.de> | NHR PerfLab Seminar 2022

April 26, 2022 22
Lessons learned: Impact of idle wave on overlap

Afzal et al.

IEEE Cluster 2019

DOI:10.1109/CLUSTER.2019.8890995

ISC HPC 2021

DOI: 10.1007/978-3-030-78713-4_19
**Lessons learned: Impact of idle wave on overlap**

**Afzal et al.**

IEEE Cluster 2019

DOI: 10.1109/CLUSTER.2019.8890995

ISC HPC 2021

DOI: 10.1007/978-3-030-78713-4_19

---

**Contention** (i.e., saturation point in memory bandwidth bottleneck)

**Overlap amplified / damped?**

**Idle wave decay**

**Propagation speed of idle wave**

Possible bottlenecks:
- memory
- cache
- on-/inter-chip network
- link b/w host & accelerator

---

**Performance**

- Number of MPI processes

**Resource**

- Scalable
- Bottleneck

```
T_exec T_non-exec T_exec T_non-exec
T_exec T_non-exec T_exec T_non-exec

...```

---

Ayesha Afzal <ayesha.afzal@fau.de> | NHR PerfLab Seminar 2022

April 26, 2022
Scalable versus contented processes

Ayesha Afzal <ayesha.afzal@fau.de> | NHR PerfLab Seminar 2022

4 Emmy 10-core sockets @2.2 GHz, non-temporal stores, bi-dir, open chain, distance-1 communication
Scalable versus contented processes

Scalable processes → No overlap
Strongly contented processes → Maximum overlap

SuperMUC-NG @2.3 GHz, non-temporal stores, bi-dir, 1024 B, close chain, distance-1 communication

A(\cdot) = B(\cdot) + s \cdot C(\cdot)
Lessons learned: Impact of idle wave on overlap

Afzal et al.  

IEEE Cluster 2019  

DOI: 10.1109/CLUSTER.2019.8890995

ISC HPC 2021  

DOI: 10.1007/978-3-030-78713-5_20

DOI: 10.1109/CLUSTER.2019.8890995

ISC HPC 2020

DOI: 10.1007/978-3-030-78713-5_20
Lessons learned: Scope

Algorithmic dependency: some collectives can be permeable to idle waves
Lessons learned: Scope

Spatial multi-tasking in GPUs

Multi-phase programs (HPCG)

Task-parallel programs

<table>
<thead>
<tr>
<th>Permutated rank</th>
<th>Permuted rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.29</td>
</tr>
<tr>
<td>1</td>
<td>0.295</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
</tr>
<tr>
<td>3</td>
<td>0.305</td>
</tr>
</tbody>
</table>

DOI: 10.1002/cpe.6816

Afzal et al.

CCPE 2022
4.1.1 Shape of computational wavefronts

The shape of computational wavefronts is crucial for optimizing the performance of sparse matrix-vector multiplication (SpMVM) operations. The computational wavefronts are a result of the communication and computation in parallel systems. They are influenced by the distribution of nonzeros across processes and the communication patterns between them.

The nonzeros are typically concentrated in blocks of matrix rows, and this distribution can affect the computational and communication load on different scales. The wavefronts describe the progression of computations and communications over time.

4.1.2 Analysis

The analysis of computational wavefronts involves understanding the distribution of nonzeros across processes and the resulting communication patterns. This helps in optimizing the workload distribution and minimizing communication overhead.

Two different implementations were tested for the SpMVM operations:

- **SPLIT** mode, where each process can compute the part of the SpMVM for blocks of matrix rows (and corresponding LHS and RHS memory).
- **NON-SPLIT** mode, where a fully developed system settles in the vicinity of the performance saturation.

In the MPI-parallel SpMVM implementation, contiguous local and remote SpMVM kernels are called. This requires nonblocking MPI calls before the local SpMVM and MPI_Wait for the remote SpMVM.

The matrices were generated using the scalable matrix library, and the communication characteristics of distributed-memory collections were considered.

For large matrices, the performance of sparse matrix-vector multiplication (SpMVM) is sufficient as to intrude the slowest socket. The matrices were generated using the scalable matrix library. In (a) of Figs. 3 shows that the actual time for the call does not play a role beyond (for Meggie) and 18 nodes (for SuperMUC-NG), where a fully developed system, where a fully developed system, where a fully developed system.

The behavior of the split (c) and non-split (d) variants is shown that the actual time for the call does not play a role. It shows that there is considerable residual GLUP/s, which shows that there is considerable residual GLUP/s, which shows that there is considerable residual GLUP/s.
Performance: MPI parallel Sparse Matrix-Vector Multiplication

- **LARGE COMMUNICATION OVERHEAD**
  - High matrix bandwidth (pe)
  - SLOW IDLE WAVE
    - Low matrix bandwidth (ep)
  - SLOW SYNCHRONIZED BASELINE
    - Non-split communication scheme
  - Higher speedup with automatic overlap

\[
D[\%] = \frac{P_{\text{barrier-free}} - P_{\text{barrier}}}{|P_{\text{barrier}}|} \times 100
\]
MPI processes occur automatically, i.e., it is not provoking communication is symmetric throughout, excluding boundaries, with \( y \) and local domain size for

\[
\begin{align*}
\text{Fig. 13: Average, minimum, and maximum aggregated time spent in}
\end{align*}
\]

\[
\text{better speed-up with less-synchronizing collective variant even if it not the most efficient implementation}
\]

\[
\begin{align*}
\text{Large amount of time spent in the MPI library is not necessarily harmful if overlapping useful work}
\end{align*}
\]

\[
\begin{align*}
\text{SuperMUC-NG nodes. The total runtime was about}
\end{align*}
\]
Pseudo implementation of parallel LBM

1: while iter ≤ nIters do
2:    stream_collide_update (lattice, u_lid, omega);
3:    set_boundary_condition (u_lid);
4:    MPI_Isend ; *
5:    MPI_Irecv ; *
6:    MPI_Wait ;
7:    ghost_cells_update () ;
8:    if ((iter % collective_step) == 0) then
   9:        MPI_Allreduce ;
10:   end if
11:   swap (local_src_lattice, local_dst_lattice) ;
12: end while
Performance: MPI parallel LULESH proxy application

Overlap effect is swamped by dominating laggars ➔ no benefit
Performance: Hybrid parallel Chebychev Filter Diagonalization

**NON-SPLIT mode**

for $k = 0 : n_a - 1$
do
for $p = 0 : n_p - 1$
do
swap($\hat{W}, \hat{U}$);
end for
end for

**SPLIT-WAIT mode**

for $k = 0 : n_a - 1$
do
for $p = 0 : n_p - 1$
do
swap($\hat{W}, \hat{U}$);
end for
end for

**PIPELINE mode**

for $p = 0 : n_p - 1$
do
swap($\hat{W}, \hat{U}$);
comm_init($\hat{U}$);
local_kernel;
MPI_wait();
end for
for $k = 0 : n_a - 2$
do
comm_init($\hat{U}$);
kernel($k$);
MPI_wait();
end for
kernel($n_a - 1$);
end for

### Matrix

<table>
<thead>
<tr>
<th>Traits</th>
<th>Data-type</th>
<th>$n_r = n_c$</th>
<th>$n_{mx}$</th>
<th>$n_{mpe}$</th>
<th>Size [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOPI-EHN</td>
<td>$m_a \times m_p \times m_z = 128 \times 128 \times 64$</td>
<td>complex double</td>
<td>268435456</td>
<td>3487563776</td>
<td>13</td>
</tr>
<tr>
<td>SPIN26</td>
<td>28-14-1</td>
<td>double</td>
<td>10406060</td>
<td>145806400</td>
<td>14</td>
</tr>
<tr>
<td>SPIN28*</td>
<td>28-14-1</td>
<td>double</td>
<td>40116600</td>
<td>601749000</td>
<td>15</td>
</tr>
<tr>
<td>SPIN30*</td>
<td>30-15-1</td>
<td>double</td>
<td>155117520</td>
<td>2481880320</td>
<td>16</td>
</tr>
</tbody>
</table>

### Communication distances

<table>
<thead>
<tr>
<th>$m_a \times m_p \times m_z$</th>
<th>$n_{nodes} = 64$</th>
<th>$n_{nodes} = 64$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \times 1 \times 1$</td>
<td>±1, −20, −19</td>
<td>1050, 105, 0.128</td>
</tr>
<tr>
<td>$1 \times n_{nodes}^{-1}$</td>
<td>±1, ±20, ±19</td>
<td>1050, 48, 8</td>
</tr>
<tr>
<td>$n_{nodes}^{-1} \times 1$</td>
<td>±1</td>
<td>1050</td>
</tr>
<tr>
<td>$1 \times n_{nodes}$</td>
<td>±1, −20, −19</td>
<td>1050, 105, 48, 8, 0.128</td>
</tr>
<tr>
<td>$n_{nodes}^{-1} \times 1$</td>
<td>±1, ±20, ±160, ±159, −19</td>
<td>1050, 105, 48, 8, 0.128</td>
</tr>
<tr>
<td>$1 \times n_{nodes}^{-1}$</td>
<td>±1, −20, −19</td>
<td>1050, 105, 0.128</td>
</tr>
<tr>
<td>$n_{nodes}^{-1} \times 1$</td>
<td>±1, −20, −140, −19, −141</td>
<td>1050, 48, 8, 8</td>
</tr>
<tr>
<td>$2 \times n_{nodes}$</td>
<td>±1, −20, −480, −160, −481, −159, −19</td>
<td>1050, 105, 48, 8, 8, 0.128</td>
</tr>
<tr>
<td>$2 \times n_{nodes}^{-1}$</td>
<td>±1, −20, −560, −80, −561, −79, −19</td>
<td>1050, 105, 48, 8, 8, 0.128</td>
</tr>
<tr>
<td>$2 \times n_{nodes}^{-1}$</td>
<td>±1, −20, −80, −79, −81, −19</td>
<td>1050, 105, 96, 8, 0.128</td>
</tr>
</tbody>
</table>

(a) TOPI-EHN

(b) SPIN26/30

(c) SPIN28

Time step

Threads per socket

Performance (Gflop/s)

Lower code balance

Ayesha Afzal <ayesha.afzal@fau.de> | NHR PerfLab Seminar 2022

April 26, 2022
Better overlap for decomposition of slow ide wave and large communication overhead

WITH BARRIER
Pipeline mode better (non-split suffer more for more saturating case)

WITHOUT BARRIER
Non-split mode on-par with pipeline mode or even better for more saturating case

Overlapping via explicit programming techniques may not be necessary for strongly bandwidth-saturating code with large (but not dominant) communication overhead due to the presence of natural overlap by desynchronization

### Performance: Hybrid parallel Chebychev Filter Diagonalization

### Better Overlap for Decomposition of Slow Ide Wave and Large Communication Overhead

**WITH BARRIER**
- Pipeline mode better (non-split suffer more for more saturating case)

**WITHOUT BARRIER**
- Non-split mode on-par with pipeline mode or even better for more saturating case

Overlapping via explicit programming techniques may not be necessary for strongly bandwidth-saturating code with large (but not dominant) communication overhead due to the presence of natural overlap by desynchronization
DisCostiC: A DSL-based Parallel Simulation Framework

Using First-Principles Analytic Performance Models
On-going and future work

Physical Oscillator Model For Parallel Distributed Computing
On-going and future work

- Simulation Framework
- Physical Oscillator Model
- Machine Learning Techniques and Advanced Metrics for Analysis of Parallel Programs
On-going and future work

Threshold Of Applying Scheduling Techniques For Load Balancing