MPI-aware compiler optimizations for improving communication-computation overlap

Anthony Danalis, ICL – University of Tennessee
Lori Pollock, University of Delaware
Martin Swany, University of Delaware
John Cavazos, University of Delaware
Motivating Example

doi = 1, 100
    sB[i] = A[i]+i
    B[i] = C[i]-i
end do
Motivating Example

do i = 1, 100
   sB[i] = A[i]+i
end do

do i = 1, 100
   B[i] = C[i]-i
end do
Motivating Example

```
do i = 1, 100
    sB[i] = A[i]+i
end do

do i = 1, 100
    B[i] = C[i]-i
end do
```

```
do i = 1, 100
    sB[i] = A[i]+i
    foo(i)
    B[i] = C[i]-i
end do
```
do i = 1, 100
    sB[i] = A[i]+i
end do

do i = 1, 100
    B[i] = C[i]-i
end do

do i = 1, 100
    sB[i] = A[i]+i
    foo(i)
    B[i] = C[i]-i
end do
do $i = 1, 100$
    $sB[i] = A[i]+i$
end do

do $i = 1, 100$
    $B[i] = C[i]-i$
end do

Compilers avoid transforming code with calls to unknown library functions
Motivating Example

do i = 1, 100
  sB[i] = A[i]+i
end do

do i = 1, 100
  sB[i] = A[i]+i
  mpi_isend(sB[i],1,r[i])
  mpi_wait(r[i])
  B[i] = C[i]-i
end do
Motivating Example

```
do i = 1, 100
   sB[i] = A[i]+i
end do

do i = 1, 100
   B[i] = C[i]-i
end do
```

```
do i = 1, 100
   sB[i] = A[i]+i
   mpi_isend(sB[i],1,r[i])
   mpi_wait(r[i])
   B[i] = C[i]-i
end do
```
Motivating Example

```fortran
do i = 1, 100
    sB[i] = A[i]+i
end do

do i = 1, 100
    B[i] = C[i]-i
end do

mpire_isend(sB[i],1,r[i])
mpire_wait(r[i])
```

Motivating Example

\[
\begin{align*}
\text{do } i = 1, 100 \\
\quad & sB[i] = A[i] + i \\
\text{end do}
\end{align*}
\]

\[
\begin{align*}
\text{do } i = 1, 100 \\
\quad & B[i] = C[i] - i \\
\text{end do}
\end{align*}
\]

\[
\begin{align*}
\text{do } i = 1, 100 \\
\quad & sB[i] = A[i] + i \\
\quad & mpi_isend(sB[i], 1, r[i]) \\
\text{end do}
\end{align*}
\]

\[
\begin{align*}
\text{do } i = 1, 100 \\
\quad & mpi_wait(r[i]) \\
\text{end do}
\end{align*}
\]

\[
\begin{align*}
\text{do } i = 1, 100 \\
\quad & B[i] = C[i] - i \\
\text{end do}
\end{align*}
\]
Motivating Example

do i = 1, 100
   sB[i] = A[i]+i
end do

do i = 1, 100
   B[i] = C[i]-i
end do

mpi_isend(sB[i],1,r[i])

mpi_wait(r[i])

B[i] = C[i]-i
end do
Motivating Example

do i = 1, 100
   sB[i] = A[i]+i
end do

do i = 1, 100
   B[i] = C[i]-i
end do

do i = 1, 100
   mpi_isend(sB[i],1,r[i])
end do

do i = 1, 100
   mpi_wait(r[i])
end do
Thesis

Enabling compilers to **safely optimize** MPI programs is **achievable and profitable**
Why not any other library?

- MPI has unparalleled penetration
- Communication performance critical for HPC
- Communication importance due to growth
- MPI IS NOT A LIBRARY
Why not any other library?

- MPI has unparalleled penetration
- Communication performance critical for HPC
- Communication importance due to grow
- MPI IS NOT A LIBRARY

MPI is a standard with well defined behavior
Compiler Transformation Groups

- Blocking to Non-Blocking MPI calls

- Communication Library Specific Transformations
  - Utilize specialized libraries in specialized environments

- Collective Call Decomposition
  - Convert a collective into multiple async. point-to-point operations

- Code Motion for Overlap Window Expansion
  - Overlap Window: code region between Isend/Irecv and Wait

- Variable Cloning
  - Vectorize/expand scalars/arrays to relax data dependencies

- LNO to create Independent Code Blocks
  - Break loops to create communication-independent computation
Safety Analysis

- MPI function equivalence rules

- Application-layer data flow effects
  - Variables passed as parameters to MPI calls

- Library-layer data flow effects
  - Memory altered due to library internal side-effects

- Control flow related rules for code motion
  - Legal locations to move an MPI call

- MPI function segmentation rules
  - How to break an MPI call to two or more
MPI function equivalence rules

<table>
<thead>
<tr>
<th>blocking function</th>
<th>non-blocking function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(buf, cnt, type, dst, tag, comm)</td>
<td>MPI_Isend(buf, cnt, type, dst, tag, comm, req)</td>
</tr>
<tr>
<td>MPI_Recv(buf, cnt, type, src, tag, comm, stat)</td>
<td>MPI_Irecv(buf, cnt, type, src, tag, comm, req)</td>
</tr>
<tr>
<td></td>
<td>MPI_Wait(req, stat)</td>
</tr>
</tbody>
</table>

 wait
## Data Flow Analysis

(Application-layer)

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(buf, cnt, type, dst, tag, com)</td>
<td>DEF: $\emptyset$</td>
<td>USE: { all arguments }</td>
</tr>
<tr>
<td>MPI_Recv(buf, cnt, type, src, tag, com, stat)</td>
<td>DEF: { buf, stat }</td>
<td>USE: { cnt, type, src, tag, com }</td>
</tr>
<tr>
<td>MPI_Isend(buf, cnt, type, dst, tag, com, req)</td>
<td>DEF: { req }</td>
<td>USE: { all arguments except “req” }</td>
</tr>
<tr>
<td>MPI_Irecv(buf, cnt, type, src, tag, com, req)</td>
<td>DEF: { buf, req }</td>
<td>USE: { cnt, type, src, tag, com }</td>
</tr>
<tr>
<td>MPI_Wait(req, stat)</td>
<td>DEF: { req, stat }</td>
<td>USE: { req }</td>
</tr>
</tbody>
</table>
Is this information sufficient?

MPI_Irecv(B1, 10, MPI_INT, 0, 0, comm_world, &rq1)
MPI_Irecv(B2, 10, MPI_INT, 0, 0, comm_world, &rq2)

safe to re-order?

call mpi_irecv(B(3,1:10), 10, MPI_INT, 0, 0, comm_world, rq, err)
call mpi_wait(rq, stat, err)

safe?

tmp(1:10) = B(3,1:10)
call mpi_irecv(tmp, 10, MPI_INT, 0, 0, comm_world, rq, err)
B(3,1:10) = tmp(1:10)
call mpi_wait(rq, stat, err)

MPI's side-effects on receive buffer last until mpi_wait()
Is this information sufficient?

MPI_Irecv(B1, 10, MPI_INT, 0, 0, comm_world, &rq1)
MPI_Irecv(B2, 10, MPI_INT, 0, 0, comm_world, &rq2)

No true dependency yet
re-ordering is incorrect

call mpi_irecv(B(3,1:10), 10, MPI_INT, 0, 0, comm_world, rq, err)
call mpi_wait(rq, stat, err)

tmp(1:10) = B(3,1:10)
call mpi_irecv(tmp, 10, MPI_INT, 0, 0, comm_world, rq, err)
B(3,1:10) = tmp(1:10)
call mpi_wait(rq, stat, err)

MPI's side-effects on receive buffer last until
mpi_wait()
Is this information sufficient?

MPI_Irecv(B1, 10, MPI_INT, 0, 0, comm_world, &rq1)
MPI_Irecv(B2, 10, MPI_INT, 0, 0, comm_world, &rq2)

No true dependency yet re-ordering is incorrect

call mpi_recev(B, 10, MPI_INT, 0, 0, comm_world, rq, err)
call mpi_wait(rq, stat, err)
A(1:10) = B(1:10)

MPI's side-effects on receive buffer last until mpi_wait()

call mpi_recev(B, 10, MPI_INT, 0, 0, comm_world, rq, err)
A(1:10) = B(1:10)
call mpi_wait(rq, stat, err)

Safe to move?
Is this information sufficient?

MPI_Irecv(B1, 10, MPI_INT, 0, 0, comm_world, &rq1)
MPI_Irecv(B2, 10, MPI_INT, 0, 0, comm_world, &rq2)

call mpi_irecv(B, 10, MPI_INT, 0, 0, comm_world, rq, err)
call mpi_wait(rq, stat, err)
A(1:10) = B(1:10)

No true dependency yet re-ordering is incorrect

MPI's side-effects on receive buffer last until mpi_wait()
# Data Flow Analysis

(Library-layer)

<table>
<thead>
<tr>
<th>Function</th>
<th>Equation/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(buf, cnt, typ, dst, tag, com)</td>
<td>outMsg[dst][tag][com][0:cnt-1] = buf[0:cnt-1]</td>
</tr>
</tbody>
</table>
| MPI_Recv(buf, cnt, typ, src, tag, com, stat) | buf[0:N-1] = inMsg[src][tag][com][0:N-1]  
 inMsg[src][tag][com][0:N-1] = artifVar |
| MPI_Isend(buf, cnt, typ, dst, tag, com, req) | outMsg[dst][tag][com][0:cnt-1] = buf[0:cnt-1]  
 buf[0:count-1] += artifVar  
 whichBuf[req] = buf |
| MPI_Irecv(buf, cnt, typ, src, tag, com, req) | buf[0:N-1] = inMsg[src][tag][com][0:N-1] + artifVar  
 inMsg[src][tag][com][0:N-1] = artifVar  
 whichBuf[req] = buf |
| MPI_Wait(req, stat)       | whichBuf[req][0:N-1] -= artifVar |
## Data Flow Analysis

**(Library-layer)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(buf, cnt, typ, dst, tag, com)</td>
<td><strong>outMsg</strong>[dst][tag][com][0:cnt-1] = buf[0:cnt-1]</td>
</tr>
</tbody>
</table>
| MPI_Recv(buf, cnt, typ, src, tag, com, stat) | buf[0:N-1] = **inMsg**[src][tag][com][0:N-1]  
**inMsg**[src][tag][com][0:N-1] = artifVar |
| MPI_Isend(buf, cnt, typ, dst, tag, com, req) | **outMsg**[dst][tag][com][0:cnt-1] = buf[0:cnt-1]  
buf[0:count-1] += artifVar  
**whichBuf**[req] = buf |
| MPI_Irecv(buf, cnt, typ, src, tag, com, req) | buf[0:N-1] = **inMsg**[src][tag][com][0:N-1] + artifVar  
**inMsg**[src][tag][com][0:N-1] = artifVar  
**whichBuf**[req] = buf |
| MPI_Wait(req, stat) | **whichBuf**[req][0:N-1] -= artifVar |

**outMsg, inMsg, whichBuf, artifVar**: special variables, not subject to optimization (volatile++)
Control flow rules

An MPI call should not be:

- Introduced into an execution path
- Removed from an execution path
- Called more/less times than originally

Legal to move from La to Lb iff:

- La dominates Lb & Lb post-dominates La
- Lb dominates La & La post-dominates Lb
Control flow rules

An MPI call should not be:

- Introduced into an execution path
- Removed from an execution path
- Called more/less times than originally

Legal to move from location \( L_a \) to \( L_b \) if:

\( L_a \) dominates \( L_b \) & \( L_b \) post-dominates \( L_a \)

\( L_b \) dominates \( L_a \) & \( L_a \) post-dominates \( L_b \)
Function segmentation rules

Synchronous calls:

✓ MPI_Send(..., count, ...) == N * MPI_Send(..., count/N, ...)
✓ Number of Recv()s should be equal to number of Send()s
✓ “count” comes from the Send() not the Recv()

Asynchronous calls:

✓ MPI_Wait() should also be called “N” times (2*N if both async)
✓ “req” and “status” should be vectorized to avoid dependencies
Optimization Algorithm

Input:
- A function of the MPI program
- The set of data transfers with message size > THRESHOLD
- Summarized IPA information on source of message buffers

attempt preliminary transformations

foreach datatransfer

DO

blocking to non-blocking communication

comm. library specific transformations

attempt expanding overlap window

TRUE

overlapped comp exec time > data transfer time?

FALSE

is variable cloning beneficial?

TRUE

attempt variable cloning

FALSE

is loop fission beneficial?

TRUE

attempt loop fission

FALSE

is CCTP beneficial?

TRUE

attempt CCTP

FALSE

DO

attempt expanding overlap window

FALSE

is loop peeling beneficial?

TRUE

attempt loop peeling

DONE

Output: Transformed MPI function
Optimization Algorithm

Input:
- The set of data transfers with message size > THRESHOLD
- Summarized IPA information on source of message buffers

attempt preliminary transformations

foreach datatransfer

DO

blocking to non-blocking communication

comm. library specific transformations

attempt expanding overlap window

Overlapped comp exec time > data transfer time?

TRUE

is variable cloning beneficial?

TRUE

attempt variable cloning

FALSE

FALSE

is loop fission beneficial?

TRUE

attempt loop fission

FALSE

is CCTP beneficial?

TRUE

attempt CCTP

FALSE

FALSE

is loop peeling beneficial?

TRUE

attempt loop peeling

DONE

Output: Transformed MPI function

send-receive matching is assumed to be provided as input through annotations
Optimization Algorithm

Iterate over different data-transfers
Optimization Algorithm

Input:
- A function of the MPI program
- The set of data transfers with message size > THRESHOLD
- Summarized IPA information on source of message buffers

1. attempt preliminary transformations
2. foreach datatransfer
   a. blocking to non-blocking communication
   b. comm. library specific transformations
   c. attempt expanding overlap window
      - TRUE: Overlapped comp exec time > data transfer time?
         - TRUE: attempt variable cloning
         - FALSE: attempt loop fission
         - FALSE: attempt CCTP
      - FALSE: attempt loop fission
      - FALSE: attempt CCTP
   d. is loop fission beneficial?
      - TRUE: attempt loop fission
      - FALSE: is CCTP beneficial?
      - TRUE: attempt CCTP
      - FALSE: is loop peeling beneficial?
      - TRUE: attempt loop peeling
      - FALSE: is loop peeling beneficial?
      - TRUE: attempt loop peeling
      - FALSE: attempt expanding overlap window
3. foreach datatransfer
   a. attempt expanding overlap window
   b. is loop peeling beneficial?
      - TRUE: attempt loop peeling
      - FALSE: attempt expanding overlap window

Output: Transformed MPI function

Iterate over different optimizations
Optimization Algorithm

A function of the MPI program

Input:
1. The set of data transfers with message size > THRESHOLD
2. Summarized IPA information on source of message buffers

attempt preliminary transformations

foreach datatransfer

DO

blocking to non-blocking communication

comm. library specific transformations

attempt expanding overlap window

DONE

TRUE

Overlapped comp exec time > data transfer time?

FALSE

is variable cloning beneficial?

FALSE

is loop fission beneficial?

FALSE

is CCTP beneficial?

TRUE

attempt variable cloning

attempt loop fission

attempt CCTP

Iterate over different data-transfers

Output: Transformed MPI function
do i = 1, 100
    sB[i] = A[i]+i
end do
mpi_irecv(rB,...)
mpi_send(sB,...)
mpi_wait()
do i = 1, 100
    B[i] = C[i]-i
end do
mpi_irecv(rB,...)
do i = 1, 100
    sB[i] = A[i]+i
end do
mpi_send(sB,...)
mpi_wait()
do i = 1, 100
    sB[i] = A[i] + i
end do
mpi_irecv(rB, ...)

mpi_send(sB, ...)
mpi_wait()

do i = 1, 100
    B[i] = C[i] - i
end do
mpi_wait()
Overlap Window Expansion

\begin{verbatim}
do i = 1, 100
    sB[i] = A[i]+i
end do

mpi_irecv(rB,...)
mpi_send(sB, ...)

mpi_wait()

do i = 1, 100
    B[i] = C[i]-i
end do
\end{verbatim}
do i = 1, 100
    sB[i] = rB[i]+i
end do
mpi_irecv(rB,...)
mpi_send(sB, ...)
mpi_wait()
do i = 1, 100
    B[i] = rB[i]-i
end do
Variable Cloning

do i = 1, 100
    sB[i] = rB[i]+i
end do
mpi_irecv(rB1,...)
mpi_send(sB, ...)
mpi_wait()
do i = 1, 100
    B[i] = rB[i]-i
end do
Variable Cloning

do i = 1, 100
    sB[i] = rB[i]+i
end do
mpi_irecv(rB1,...)
mpi_send(sB, ...)
mpi_wait()
do i = 1, 100
    B[i] = rB[i]-i
end do
Variable Cloning

do i = 1, 100
   sB[i] = rB[i]+i
end do
mpi_irecv(rB1,...)
mpi_send(sB, ...)
mpi_wait()

do i = 1, 100
   B[i] = rB1[i]-i
end do
do i = 1, 100
    sB[i] = rB[i] + i
end do
mpi_irecv(rB1, ...)
mpi_send(sB, ...)
mpi_wait()
do i = 1, 100
    B[i] = rB1[i] - i
end do
Loop Fission

do i = 1, 100
  sB[i] = A[i]+i
  mpi_isend(sB[i],r)
  mpi_wait(r)
  B[i] = C[i]-i
end do

do i = 1, 100
  sB[i] = A[i]+i
  mpi_isend(sB[i],r[i])
end do

do i = 1, 100
  mpi_wait(r[i])
end do

do i = 1, 100
  B[i] = C[i]-i
end do
do i = 1, 100
  sB[i] = A[i]+i
  mpi_isend(sB[i], r)
  mpi_wait(r)
  B[i] = C[i]-i
end do

do i = 1, 100
  sB[i] = A[i]+i
  mpi_isend(sB[i], r[i])
end do

do i = 1, 100
  mpi_wait(r[i])
end do

do i = 1, 100
  B[i] = C[i]-i
end do
CCTP
Communication and Computation Tiling & Pipelining

\begin{verbatim}
do T=1,N,K
  mpi_irecv(rB[T],K,rr[T/K])
  do i = T, T+K-1
    sB[i] = …
  end do
  mpi_isend(sB[T],K,sr[T/K])
  if( T > 1 )
    mpi_wait(sr[T/K-1])
    mpi_wait(rr[T/K-1])
  end if
end do
mpi_wait(sr[T/K-1])
mpi_wait(rr[T/K-1])
\end{verbatim}
CCTP
Communication and Computation Tiling & Pipelining

\begin{align*}
\text{mpi\_irecv}(rB,N,rr) \\
do \ i = 1, N \\
\quad \text{sB}[i] = \ldots \\
\end{align*}

\begin{align*}
\text{mpi\_isend}(sB,N,\text{sr}) \\
\text{mpi\_wait}(\text{sr}) \\
\text{mpi\_wait}(rr) \\
\end{align*}

Loop tiling (strip mining)

\begin{align*}
\text{do \ [T=1,N,K]} \\
\text{mpi\_irecv}(rB[T],K,rr[T/K]) \\
\quad \text{do \ i = T, T+K-1} \\
\quad \quad \text{sB}[i] = \ldots \\
\quad \quad \text{end do} \\
\text{mpi\_isend}(sB[T],K,\text{sr}[T/K]) \\
\text{if} ( \ T > 1 \ ) \\
\quad \text{mpi\_wait}(\text{sr}[T/K-1]) \\
\quad \text{mpi\_wait}(rr[T/K-1]) \\
\quad \text{end if} \\
\text{end do} \\
\text{mpi\_wait}(\text{sr}[T/K-1]) \\
\text{mpi\_wait}(rr[T/K-1])
\end{align*}
mpi_irecv(rB, N, rr)
do i = 1, N
  sB[i] = ...
end do
mpi_isend(sB, N, sr)
mpi_wait(sr)
mpi_wait(rr)

Loop tiling (strip mining)
Commun. segmentation

do T=1, N, K
  mpi_irecv(rB[T], K, rr[T/K])
do i = T, T+K-1
  sB[i] = ...
end do
mpi_isend(sB[T], K, sr[T/K])
if( T > 1 )
  mpi_wait(sr[T/K-1])
  mpi_wait(rr[T/K-1])
end if
end do
mpi_wait(sr[T/K-1])
mpi_wait(rr[T/K-1])
**CCTP**

**Communication and Computation Tiling & Pipelining**

```fortran
mpi_irecv(rB,N,rr)
do i = 1, N
   sB[i] = ...
end do
mpi_isend(sB,N,sr)
mpi_wait(sr)
mpi_wait(rr)
```

Loop tiling (strip mining)
Commun. Segmentation
Loop fusion

```fortran
do T=1,N,K
   mpi_irecv(rB[T],K,rr[T/K])
   do i = T, T+K-1
      sB[i] = ...
   end do
   mpi_isend(sB[T],K,sr[T/K])
   if( T > 1 )
      mpi_wait(sr[T/K-1])
   mpi_wait(rr[T/K-1])
end if
end do
mpi_wait(sr[T/K-1])
mpi_wait(rr[T/K-1])
```
CCTP
Communication and Computation Tiling & Pipelining

mpi_irecv(rB,N,rr)
do i = 1, N
    sB[i] = …
end do
mpi_isend(sB,N,sr)
mpi_wait(sr)
mpi_wait(rr)

mpi_irecv(rB[T],K,rr[T/K])
do i = T, T+K-1
    sB[i] = …
end do
mpi_isend(sB[T],K,sr[T/K])
if( T > 1 )
    mpi_wait(sr[T/K-1])
    mpi_wait(rr[T/K-1])
end if
end do
mpi_wait(sr[T/K])
mpi_wait(rr[T/K])

Loop tiling (strip mining)
Commun. Segmentation
Loop fusion
Loop alignment
CCTP
Communication and Computation Tiling & Pipelining

\[
do \ T=1, N, K \\
mpi_{irecv}(rB,N,rr) \\
do \ i = 1, N \\
\text{sB}[i] = \ldots \\
end \ do \\
mpi_{isend}(sB,N,\text{sr}) \\
mpi_{wait}(\text{sr}) \\
mpi_{wait}(rr) \\
\]

Loop tiling (strip mining)
Commun. Segmentation
Loop fusion
Loop alignment
Loop peeling
Loop Peeling

```fortran
mpi_irecv(rB,N,rr)
do i = 1, N
  do j = 1, M
    A[j,i] = ...
  end do
end do
mpi_isend(A[1,1],M,sr)
mpi_wait(sr)
mpi_wait(rr)
```
Loop Peeling

mpi_irecv(rB,N,rr)
do i = 1, N
  do j = 1, M
    A[j,i] = ...
  end do
end do
mpi_isend(A[1,1],M,sr)
mpi_wait(sr)
mpi_wait(rr)

1st

mpi_irecv(rB,N,rr)
do j = 1, M
  A[j,1] = ...
end do
do i = 2, N
  do j = 1, M
    A[j,i] = ...
  end do
end do
mpi_isend(A[1,1],M,sr)
mpi_wait(sr)
mpi_wait(rr)
Loop Peeling

```c
mpi_irecv(rB,N,rr)
do i = 1, N
    do j = 1, M
        A[j,i] = ...
    end do
end do
mpi_isend(A[1,1],M,sr)
mpi_wait(sr)
mpi_wait(rr)

mpi_irecv(rB,N,rr)
do j = 1, M
    A[j,1] = ...
end do

do i = 2, N
    do j = 1, M
        A[j,i] = ...
    end do
end do
mpi_isend(A[1,1],M,sr)
mpi_wait(sr)
mpi_wait(rr)
```
Loop Peeling

mpi_irecv(rB,N,rr)
do i = 1, N
    do j = 1, M
        A[j,i] = ...
    end do
end do
mpi_isend(A[1,1],M,sr)
mpi_wait(sr)
mpi_wait(rr)

mpi_irecv(rB,N,rr)
do j = 1, M
    A[j,1] = ...
end do
mpi_isend(A[1,1],M,sr)
do i = 2, N
    do j = 1, M
        A[j,i] = ...
    end do
end do
mpi_wait(sr)
mpi_wait(rr)
Experimental Results (NAS:LU:blts)
Experimental Results (NAS: MG)

Transformed Code Versions

- Non Blocking
- Unroll Com Loop
- Redun Store Elim
- Recv Hoise
- Recv Buf Clone
- Recv Hoist
- Send Buf Clone
- Send Wait Sink

Speedup

- MPI
- Gravel
Experimental Results (HYCOM:xcsum)

![Graph showing speedup of transformed code versions]

- **Non Blocking**
- **Recv Buffer Cloning**
- **Fission**
- **Recv Hoisting**

**Legend:**
- White bar: MPI
- Red bar: Gravel

**Y-axis (Speedup):**
- 1.0
- 1.1
- 1.2
- 1.3
- 1.4
- 1.5
- 1.6

**X-axis (Transformed Code Versions):**
- Non Blocking
- Recv Buffer Cloning
- Fission
- Recv Hoisting
Experimental Results (HYCOM: xcaget)
Summary

- Safety analysis rules that guarantee correctness
- Traditional optimizations that improve MPI communication
- Systematic ordering of optimizations by overall algorithm
- Performance can be improved even in complex applications