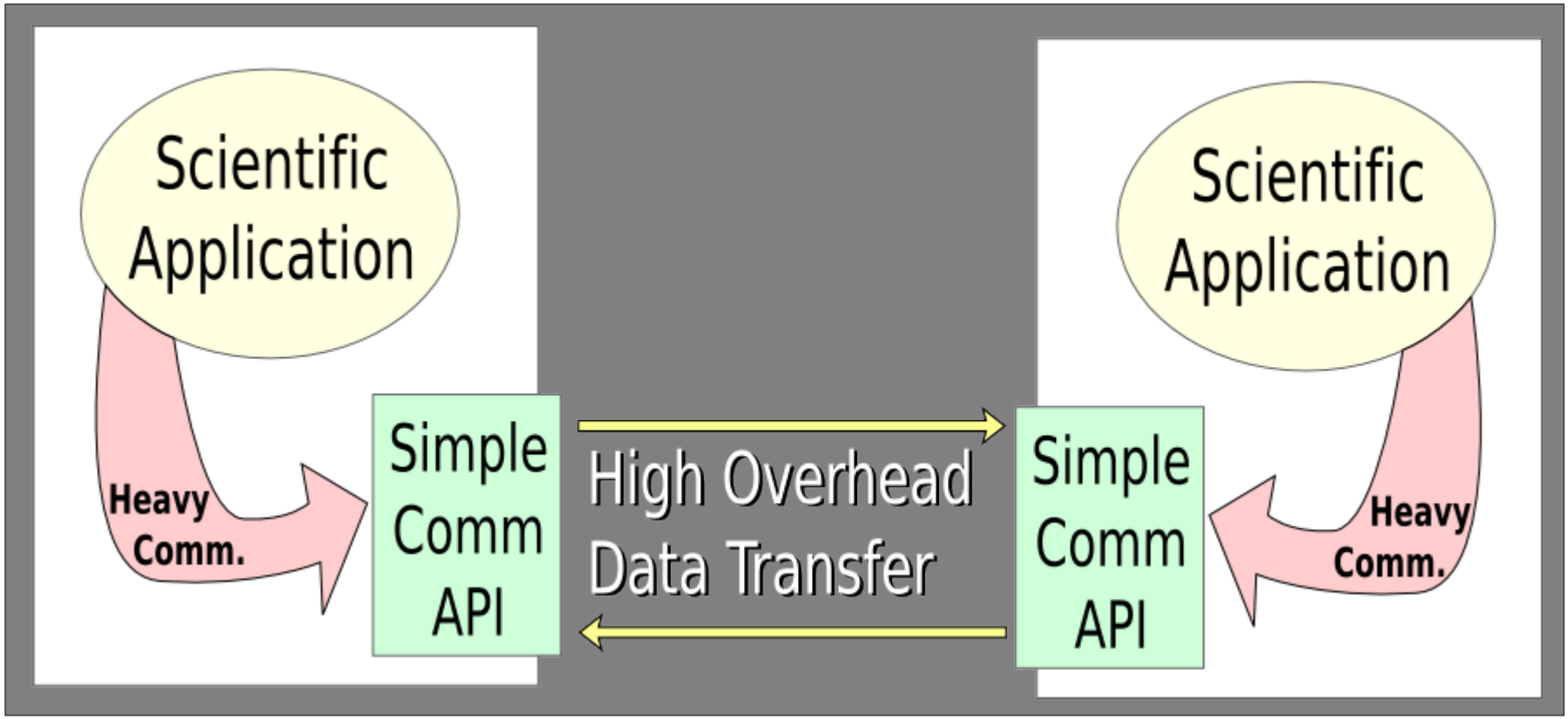
Automatic MPI application transformation with **ASPhALT**

Anthony Danalis **Lori Pollock** **Martin Swany**

University of Delaware

Problem


Scientific Application
Heavy Comm.
Simple Comm API
High Overhead Data Transfer
Simple Comm API
Scientific Application
Heavy Comm.

Overall Research Goal

Requirements:

- ✔ Achieve high-performance communication
- ✔ Simplify the MPI code developers write

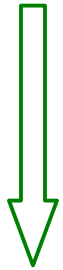
Have your cake
+
Eat your cake

Overall Research Goal

Requirements:

- ✔ Achieve high-performance communication
- ✔ Simplify the MPI code developers write

Have your cake
+
Eat your cake



Automatic cake making machine

Overall Research Goal

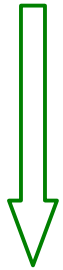
Requirements:

- ✔ Achieve high-performance communication
- ✔ Simplify the MPI code developers write

Proposed Solution:

An **automatic** system that **transforms** simple communication code into efficient code.

Have your cake
+
Eat your cake



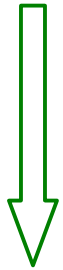
Automatic cake making machine

Overall Research Goal

Requirements:

- Achieve high-performance communication
- Simplify the MPI code developers write

Have your cake
+
Eat your cake



Automatic cake making machine

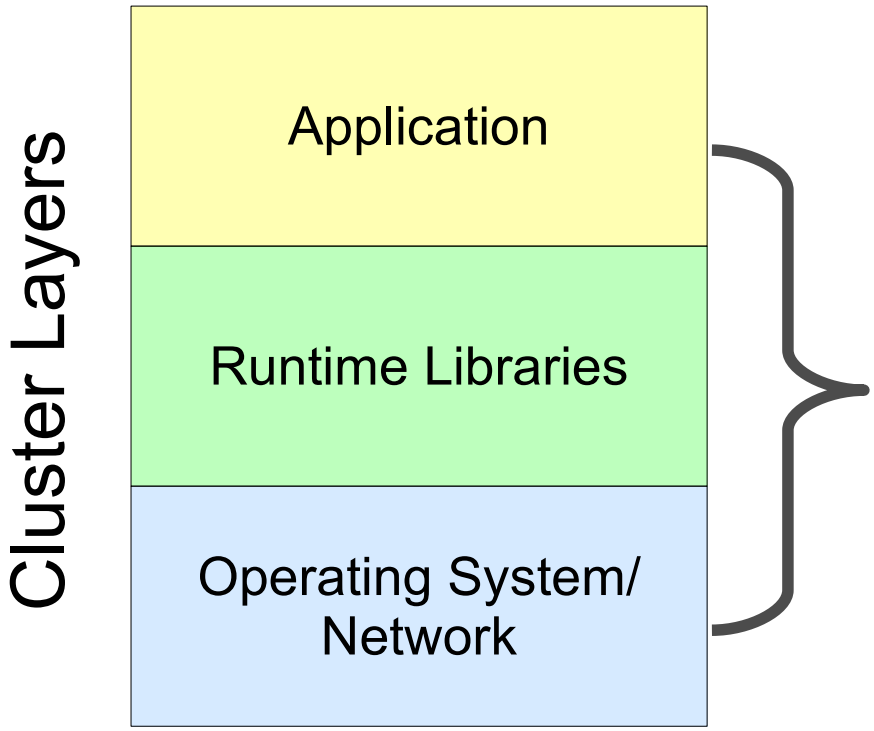
Proposed Solution:

An **automatic** system that **transforms** simple communication code into efficient code.

Side-effect:

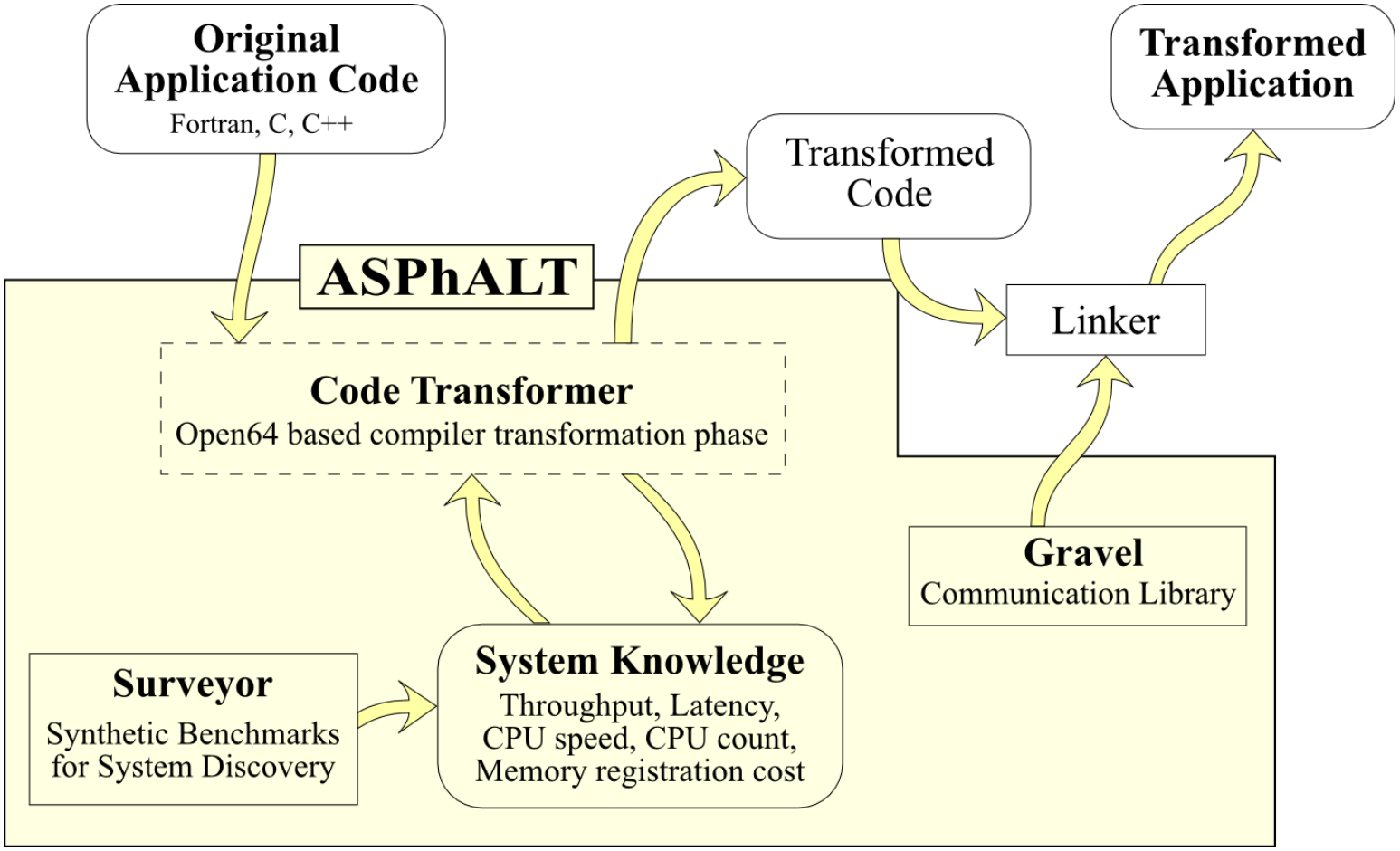
Enables **legacy parallel MPI applications** to scale, even if written **without any knowledge** of this system

Overall Research Goal



ASPhALT:
Information from multiple layers contributes to source optimization

Our Framework : ASPhALT*



Why/How transform the code?

Example 1: Synchronous Transfer to Asynchronous Transfer

Original Synch. transfer

```
array[N]

do i = 1, N
   array[i] = ...
enddo
mpi_Isend( array[1], N, sreq )
mpi_recv( ... )
mpi_wait( sreq )

do i = 1, N
   other[i] = ...
enddo
```

Computation

Communication

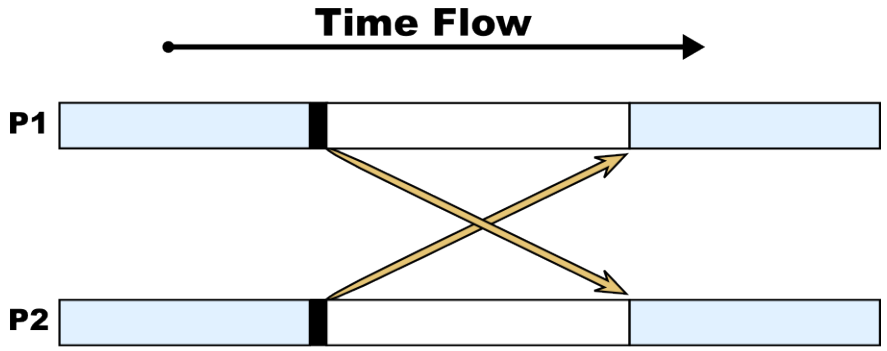
Computation (independent)

Why/How transform the code?

Example 1: Synchronous Transfer to Asynchronous Transfer

Original Synch. transfer

```
array[N]

do i = 1, N
   array[i] = ...
enddo
mpi_Isend( array[1], N, sreq )
mpi_recv( ... )
mpi_wait( sreq )

do i = 1, N
   other[i] = ...
enddo
```

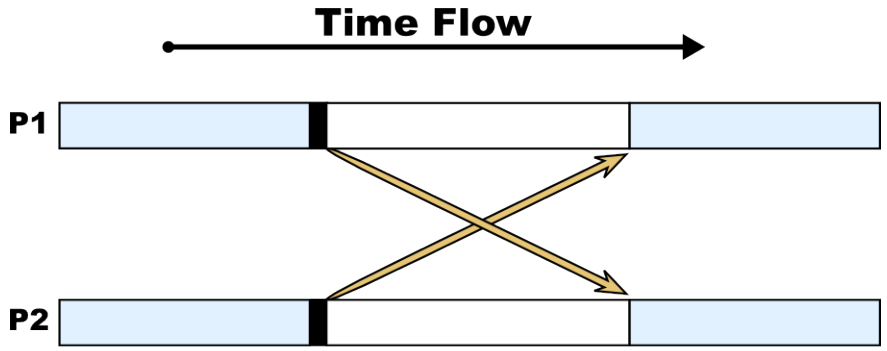


Why/How transform the code?

Example 1: Synchronous Transfer to Asynchronous Transfer

Original Synch. transfer

```
array[N]

do i = 1, N
   array[i] = ...
enddo
mpi_Isend( array[1], N, sreq )
mpi_recv( ... )
mpi_wait( sreq )

do i = 1, N
   other[i] = ...
enddo
```



Asynch. Transfer

```
array[N]

do i = 1, N
   array[i] = ...
enddo
mpi_Irecv( rreq )
mpi_Isend( array[1], N, sreq )

do i = 1, N
   other[i] = ...
enddo
mpi_wait( sreq)
mpi_wait( rreq)
```

- Computation
- Communication Initiation
- Computation (independent)
- Communication Finalization

Why/How transform the code?

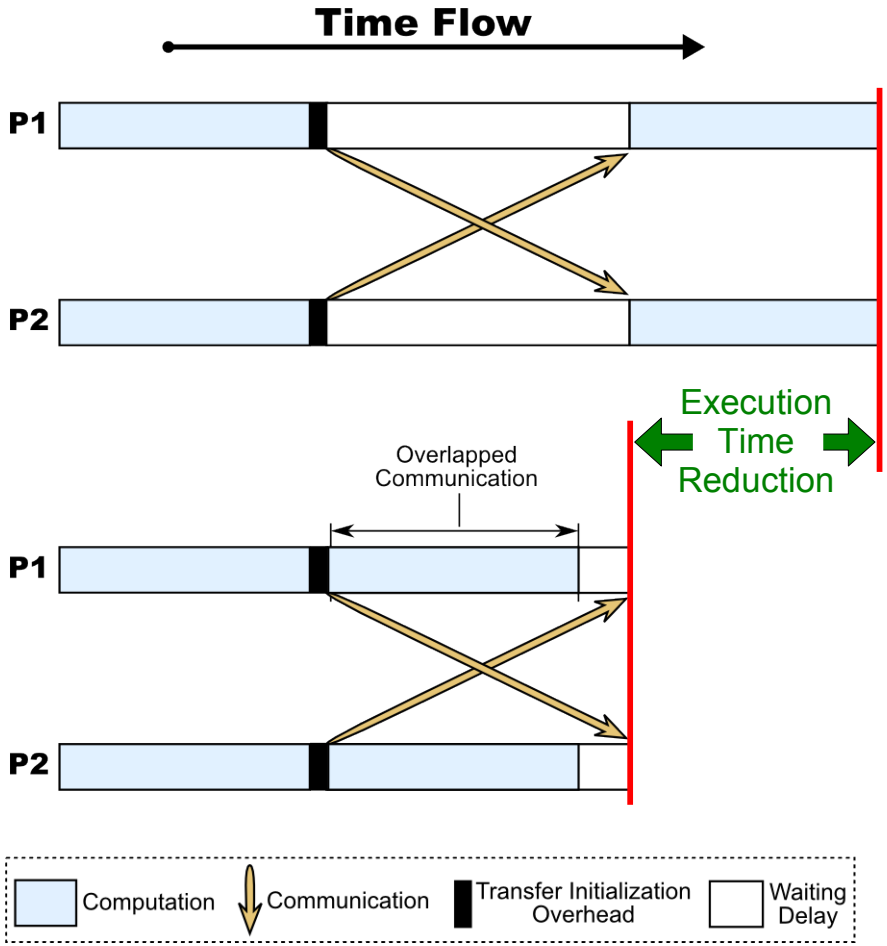
Example 1: Synchronous Transfer to Asynchronous Transfer

Original Synch. transfer

```
array[N]

do i = 1, N
   array[i] = ...
enddo
mpi_Isend( array[1], N, sreq )
mpi_recv( ... )
mpi_wait( sreq )

do i = 1, N
   other[i] = ...
enddo
```

Asynch. Transfer

```
array[N]

do i = 1, N
   array[i] = ...
enddo
mpi_Irecv( rreq )
mpi_Isend( array[1], N, sreq )

do i = 1, N
   other[i] = ...
enddo
mpi_wait( sreq)
mpi_wait( rreq)
```

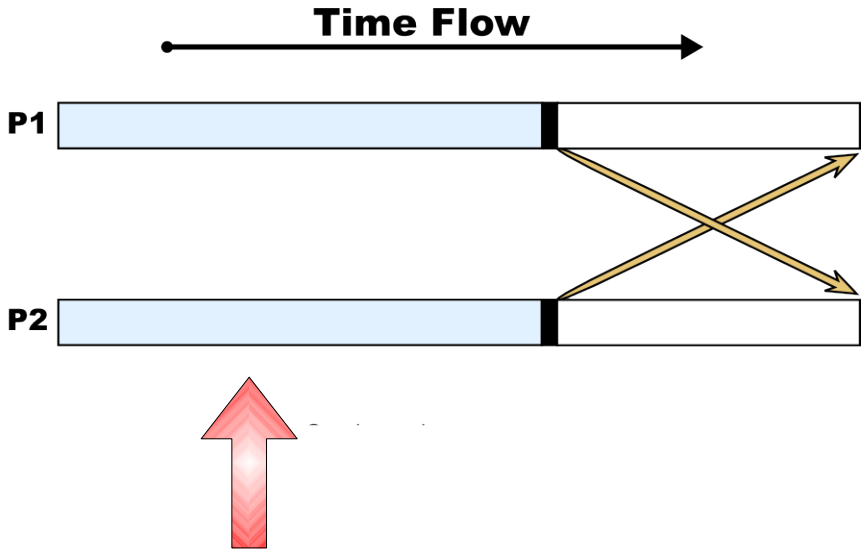


Why/How transform the code?

Example 2: Loop Tiling (prepushing)

Original code

```
array[N]

do i = 1, N
   array[i] = ...
enddo

mpi_Irecv( rreq )
mpi_send( array[1], N )
mpi_wait( rreq )
```

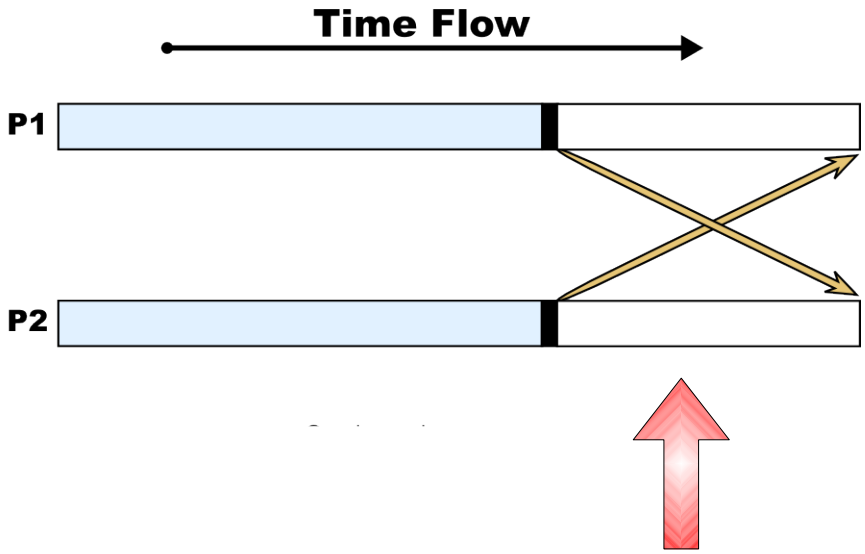


Why/How transform the code?

Example 2: Loop Tiling (prepushing)

Original code

```
array[N]

do i = 1, N
   array[i] = ...
enddo

mpi_Irecv( rreq )
mpi_send( array[1], N )
mpi_wait( rreq )
```



Why/How transform the code?

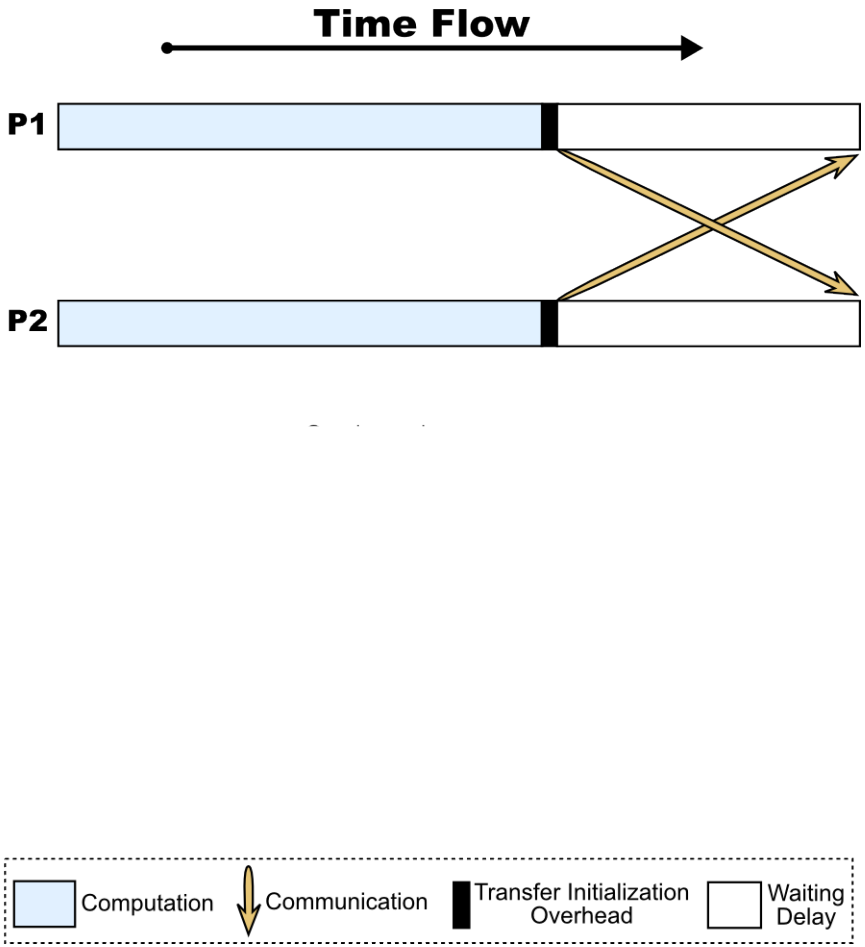
Example 2: Loop Tiling (prepushing)

Original code

```
array[N]

do i = 1, N
   array[i] = ...
enddo

mpi_Irecv( rreq )
mpi_send( array[1], N )
mpi_wait( rreq )
```

Tiled code

```
array[N]

do T = 1, N, S
   do i = T, T+S-1
      array[i] = ...
   enddo

   mpi_Irecv( rreq[T] )
   mpi_Isend( array[T], S, sreq[T] )
   if( T > 1 ) then
      mpi_wait( sreq[T-1] )
      mpi_wait( rreq[T-1] )
   endif
enddo
```



Why/How transform the code?

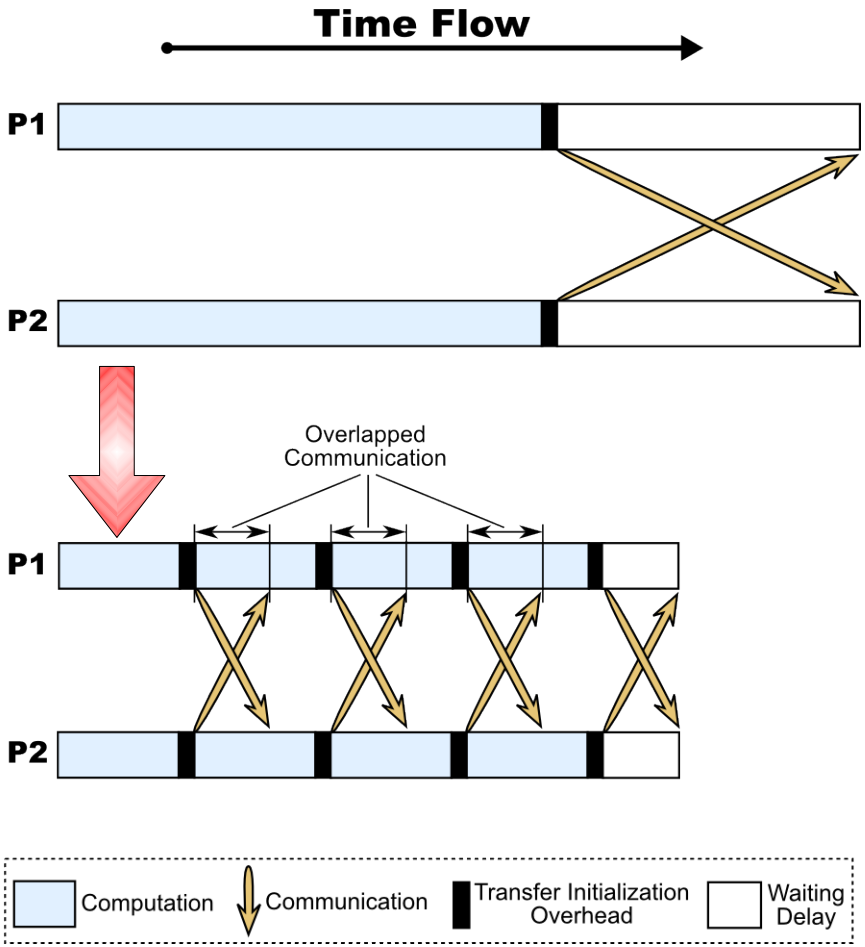
Example 2: Loop Tiling (prepushing)

Original code

```
array[N]

do i = 1, N
   array[i] = ...
enddo

mpi_Irecv( rreq )
mpi_send( array[1], N )
mpi_wait( rreq )
```

Tiled code

```
array[N]

do T = 1, N, S
   do i = T, T+S-1
      array[i] = ...
   enddo

   mpi_Irecv( rreq[T] )
   mpi_Isend( array[T], S, sreq[T] )
   if( T > 1 ) then
      mpi_wait( sreq[T-1] )
      mpi_wait( rreq[T-1] )
   endif
enddo
```



Why/How transform the code?

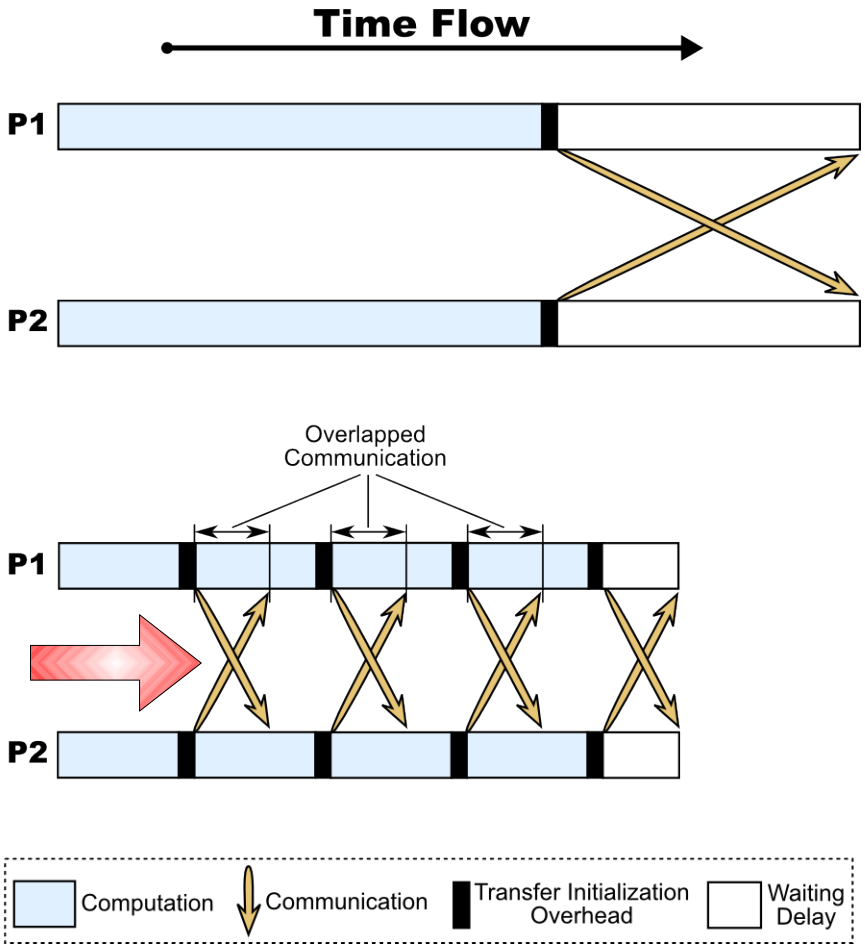
Example 2: Loop Tiling (prepushing)

Original code

```
array[N]

do i = 1, N
   array[i] = ...
enddo

mpi_Irecv( rreq )
mpi_send( array[1], N )
mpi_wait( rreq )
```

Tiled code

```
array[N]

do T = 1, N, S
   do i = T, T+S-1
      array[i] = ...
   enddo

   mpi_Irecv( rreq[T] )
   mpi_Isend( array[T], S, sreq[T] )
   if( T > 1 ) then
      mpi_wait( sreq[T-1] )
      mpi_wait( rreq[T-1] )
   endif
enddo
```

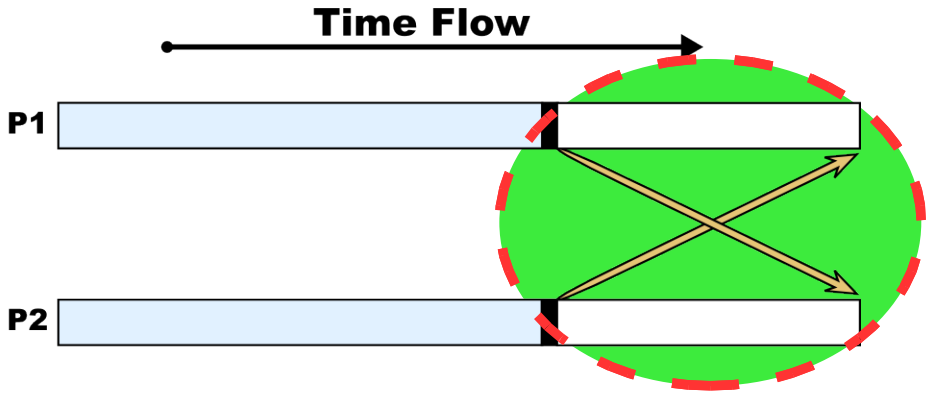


Why/How transform the code?

Example 2: Loop Tiling (prepushing)

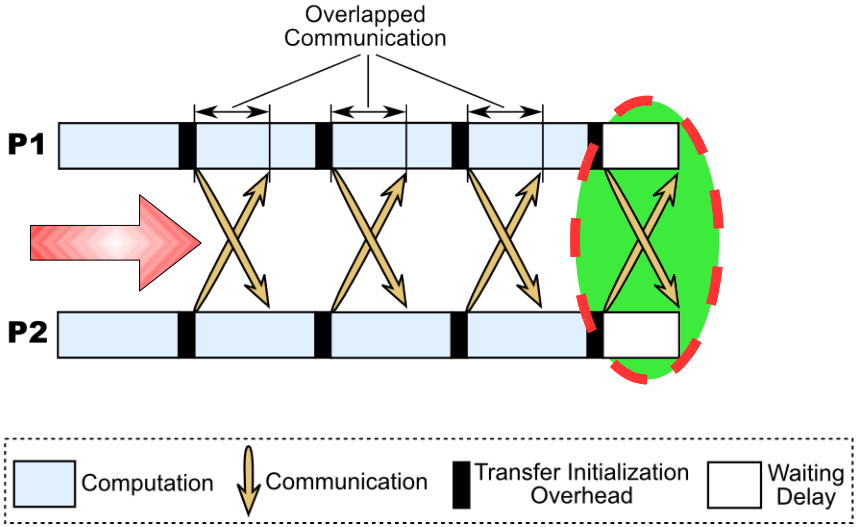
Original code

```
array[N]

do i = 1, N
   array[i] = ...
enddo

mpi_Irecv( rreq )
mpi_send( array[1], N )
mpi_wait( rreq )
```



Tiled code

```
array[N]

do T = 1, N, S
   do i = T, T+S-1
      array[i] = ...
   enddo

   mpi_Irecv( rreq[T] )
   mpi_Isend( array[T], S, sreq[T] )
   if( T > 1 ) then
      mpi_wait( sreq[T-1] )
      mpi_wait( rreq[T-1] )
   endif
enddo
```

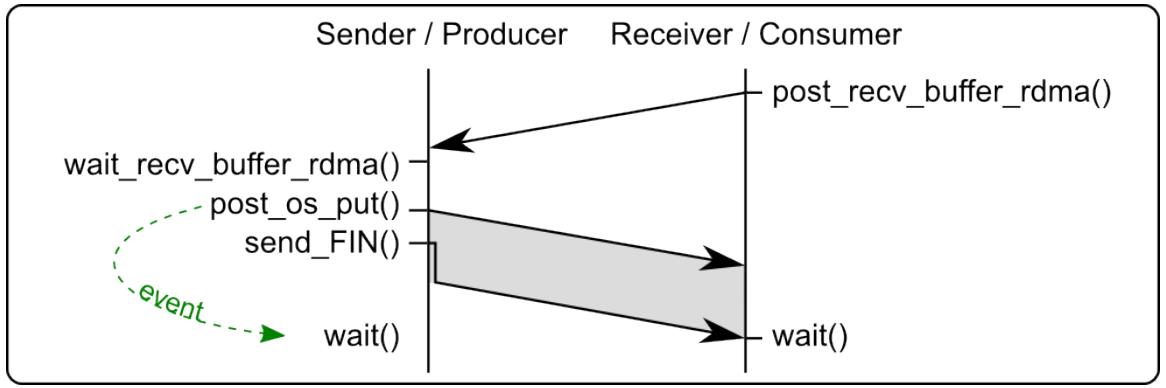


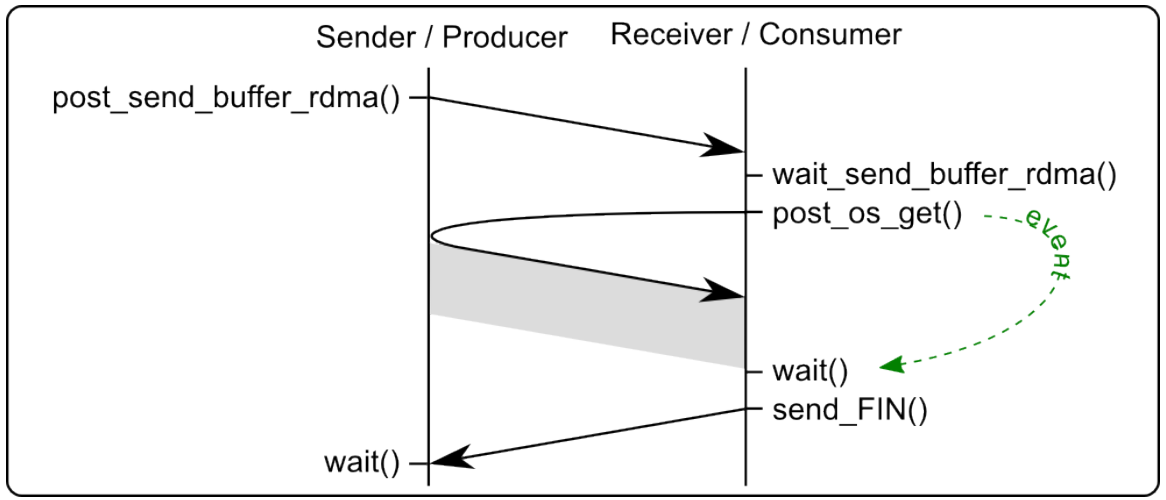
Utilizing *Gravel* to increase overlap

Gravel:

- Thin library with no additional costs
- Provide access to useful low level features (RDMA)
- Provide function separation (Handshake / Data Transfer)
- Abstract hardware/protocol/language details
- Utilize existing state of the art APIs
  - uDAPL

Gravel Protocols





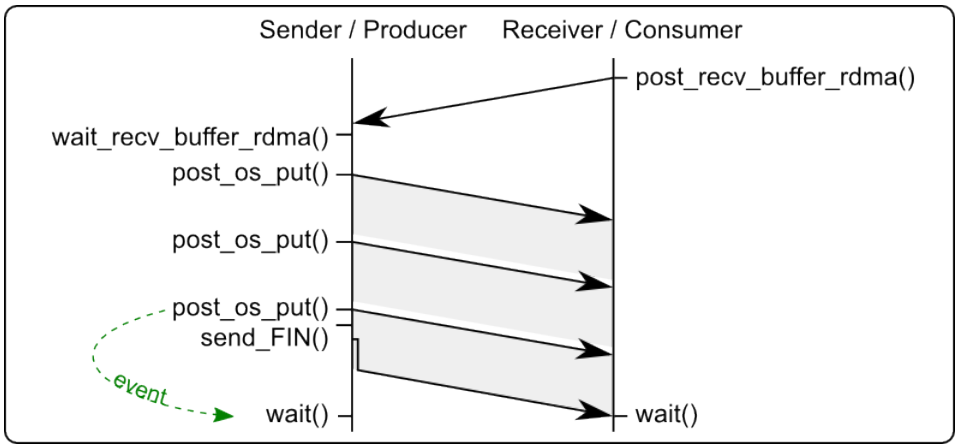
Advanced use of *Gravel*

Before

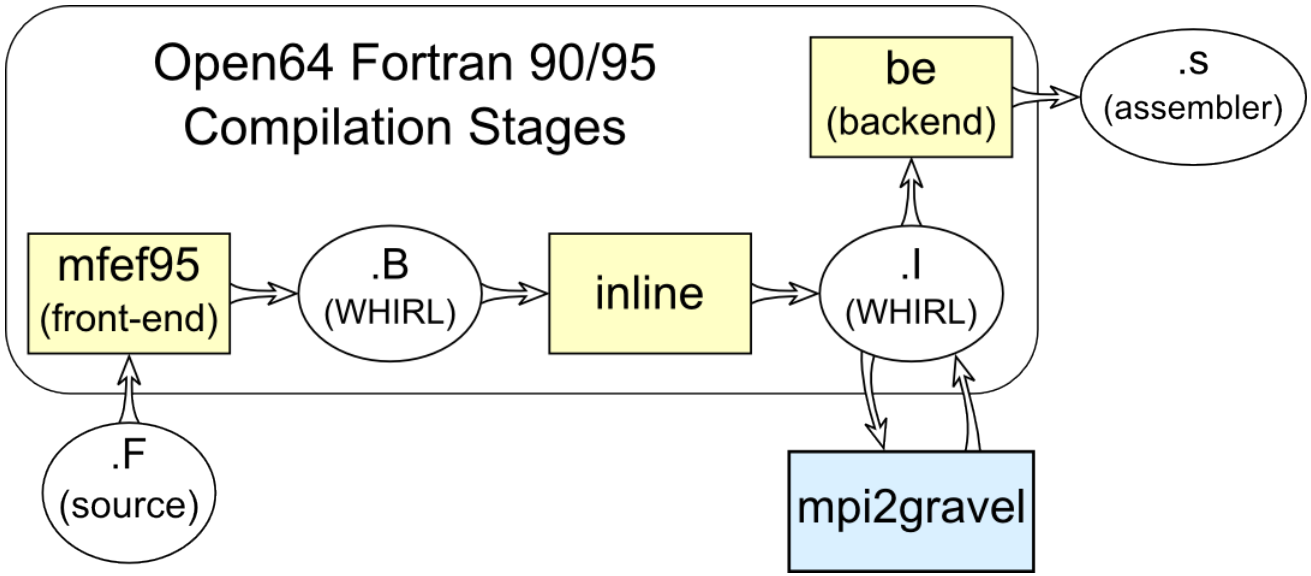
```
mpi_irecv()

do i=1,N
   sBuf[ i ] = ...
enddo

mpi_isend()
mpi_waitall()
```

After

```
gravel_post_recv_buffer_rdma()
do T=1, N, K
   do i=T, T+K-1
      sBuf[ i ] = ...
   enddo
   if( T == 1 ) then
      gravel_wait_recv_buffer_rdma()
   endif
   gravel_post_os_put()
enddo
gravel_send_fin(next, ierr)
gravel_waitall()
```



Automatic Transformer



Automation Challenges

- Inter-procedural Analysis
- Data Dependence Analysis
- Array Access Analysis
- Profitability Analysis

Automation Challenges

- Inter-procedural Analysis

Communication might take place in a separate function.

```
iex = 1
call exchange_1( v,k,iex )

do j = jend, jst, -1
   do i = iend, ist, -1
      do m = 1, 5
           tv( m, i, j ) =
>     omega * (  udz( m, 1, i, j ) * v( 1, i, j, k+1 )
>              + udz( m, 2, i, j ) * v( 2, i, j, k+1 )
>              + udz( m, 3, i, j ) * v( 3, i, j, k+1 )
>              + udz( m, 4, i, j ) * v( 4, i, j, k+1 )
>              + udz( m, 5, i, j ) * v( 5, i, j, k+1 ) )
      end do
   end do
end do
```

Automation Challenges

- Inter-procedural Analysis

Communication might take place in a separate function.

Inlining can alleviate the problem (but might create others)

```
call MPI_RECV( dum(1,exc1_jst), ... )
do j=exc1_jst,exc1_jend
  v(1,exc1_nx+1,j,k) = dum(1,j)
  v(2,exc1_nx+1,j,k) = dum(2,j)
  v(3,exc1_nx+1,j,k) = dum(3,j)
  v(4,exc1_nx+1,j,k) = dum(4,j)
  v(5,exc1_nx+1,j,k) = dum(5,j)
enddo
```

exchange_1(v, k, ...)

```
 do j = jend, jst, -1
   do i = iend, ist, -1
     do m = 1, 5
          tv( m, i, j ) =
>     omega * (  udz( m, 1, i, j ) * v( 1, i, j, k+1 )
>              + udz( m, 2, i, j ) * v( 2, i, j, k+1 )
>              + udz( m, 3, i, j ) * v( 3, i, j, k+1 )
>              + udz( m, 4, i, j ) * v( 4, i, j, k+1 )
>              + udz( m, 5, i, j ) * v( 5, i, j, k+1 ) )
     end do
   end do
 end do
```

Automation Challenges

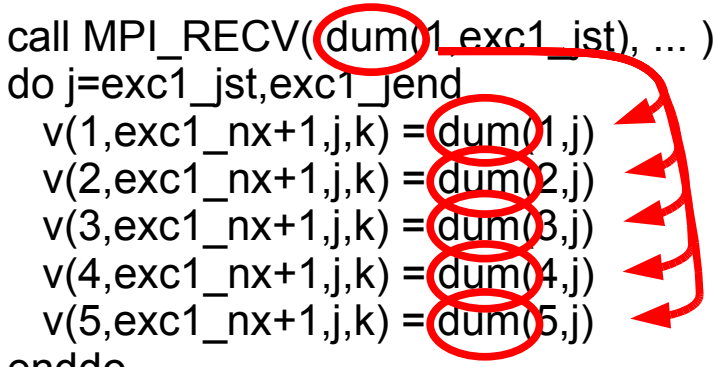
- Data Dependence Analysis

Used to find (in)dependent computation

```
call MPI_RECV( dum(1,exc1_jst), ... )
do j=exc1_jst,exc1_jend
  v(1,exc1_nx+1,j,k) = dum(1,j)
  v(2,exc1_nx+1,j,k) = dum(2,j)
  v(3,exc1_nx+1,j,k) = dum(3,j)
  v(4,exc1_nx+1,j,k) = dum(4,j)
  v(5,exc1_nx+1,j,k) = dum(5,j)
enddo

 do j = jend, jst, -1
   do i = iend, ist, -1
     do m = 1, 5
          tv( m, i, j ) =
>     omega * (  udz( m, 1, i, j ) * v( 1, i, j, k+1 )
>            + udz( m, 2, i, j ) * v( 2, i, j, k+1 )
>            + udz( m, 3, i, j ) * v( 3, i, j, k+1 )
>            + udz( m, 4, i, j ) * v( 4, i, j, k+1 )
>            + udz( m, 5, i, j ) * v( 5, i, j, k+1 ) )
     end do
   end do
 end do
```

Automation Challenges

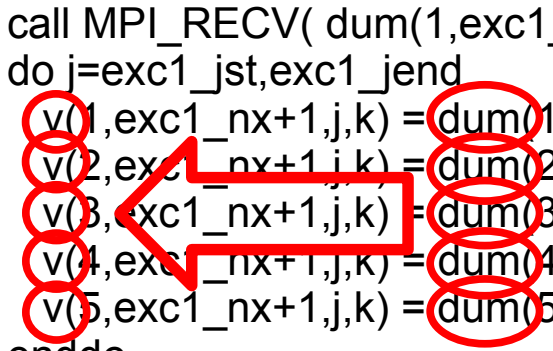
- Data Dependence Analysis




```
 do j = jend, jst, -1
    do i = iend, ist, -1
       do m = 1, 5
            tv( m, i, j ) =
>      omega * (  udz( m, 1, i, j ) * v( 1, i, j, k+1 )
>              + udz( m, 2, i, j ) * v( 2, i, j, k+1 )
>              + udz( m, 3, i, j ) * v( 3, i, j, k+1 )
>              + udz( m, 4, i, j ) * v( 4, i, j, k+1 )
>              + udz( m, 5, i, j ) * v( 5, i, j, k+1 ) )
       end do
    end do
 end do
```

Automation Challenges

- Data Dependence Analysis



```
call MPI_RECV( dum(1,exc1_jst), ... )
do j=exc1_jst,exc1_jend
  v(1,exc1_nx+1,j,k) = dum(1,j)
  v(2,exc1_nx+1,j,k) = dum(2,j)
  v(3,exc1_nx+1,j,k) = dum(3,j)
  v(4,exc1_nx+1,j,k) = dum(4,j)
  v(5,exc1_nx+1,j,k) = dum(5,j)
enddo
```

```
 do j = jend, jst, -1
    do i = iend, ist, -1
       do m = 1, 5
            tv( m, i, j ) =
>       omega * (  udz( m, 1, i, j ) * v( 1, i, j, k+1 )
>              + udz( m, 2, i, j ) * v( 2, i, j, k+1 )
>              + udz( m, 3, i, j ) * v( 3, i, j, k+1 )
>              + udz( m, 4, i, j ) * v( 4, i, j, k+1 )
>              + udz( m, 5, i, j ) * v( 5, i, j, k+1 ) )
       end do
    end do
 end do
```

Automation Challenges

- Data Dependence Analysis

```
call MPI_RECV( dum(1,exc1_jst), ... )
do j=exc1_jst,exc1_jend
  v(1,exc1_nx+1,j,k) = dum(1,j)
  v(2,exc1_nx+1,j,k) = dum(2,j)
  v(3,exc1_nx+1,j,k) = dum(3,j)
  v(4,exc1_nx+1,j,k) = dum(4,j)
  v(5,exc1_nx+1,j,k) = dum(5,j)
enddo

 do j = jend, jst, -1
    do i = iend, ist, -1
       do m = 1, 5
            tv( m, i, j ) =
>      omega * (  udz( m, 1, i, j ) * v( 1, i, j, k+1 )
>              + udz( m, 2, i, j ) * v( 2, i, j, k+1 )
>              + udz( m, 3, i, j ) * v( 3, i, j, k+1 )
>              + udz( m, 4, i, j ) * v( 4, i, j, k+1 )
>              + udz( m, 5, i, j ) * v( 5, i, j, k+1 ) )
       end do
    end do
 end do
```

Automation Challenges

- Array Access Analysis

Independent regions of a single array

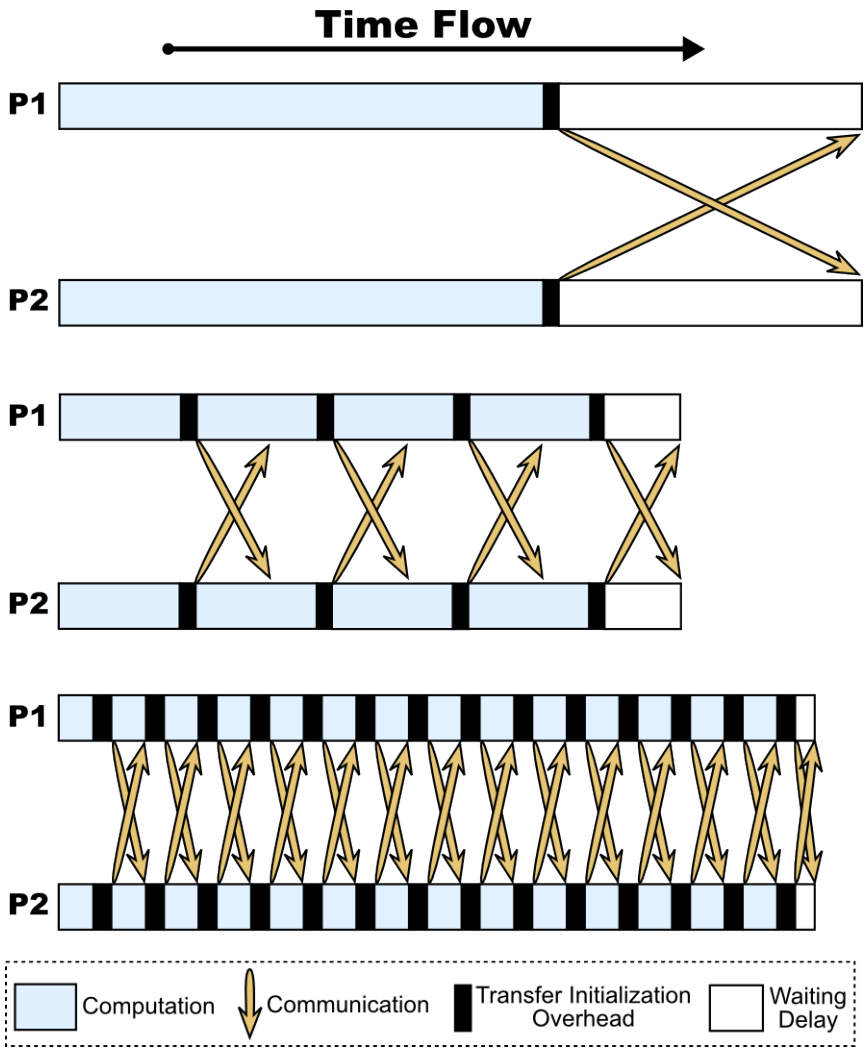
```
call MPI_RECV( dum(1,exc1_jst), ... )
do j=exc1_jst,exc1_jend
  v(1,exc1_nx+1,j,k) = dum(1,j)
  v(2,exc1_nx+1,j,k) = dum(2,j)
  v(3,exc1_nx+1,j,k) = dum(3,j)
  v(4,exc1_nx+1,j,k) = dum(4,j)
  v(5,exc1_nx+1,j,k) = dum(5,j)
enddo

 do j = jend, jst, -1
   do i = iend, ist, -1
     do m = 1, 5
          tv( m, i, j ) =
>     omega * (  udz( m, 1, i, j ) * v( 1, i, j, k+1 )
>              + udz( m, 2, i, j ) * v( 2, i, j, k+1 )
>              + udz( m, 3, i, j ) * v( 3, i, j, k+1 )
>              + udz( m, 4, i, j ) * v( 4, i, j, k+1 )
>              + udz( m, 5, i, j ) * v( 5, i, j, k+1 ) )
     end do
   end do
 end do
```

Independent

Automation Challenges

- Profitability Analysis

Tile/Buffer Size Matters



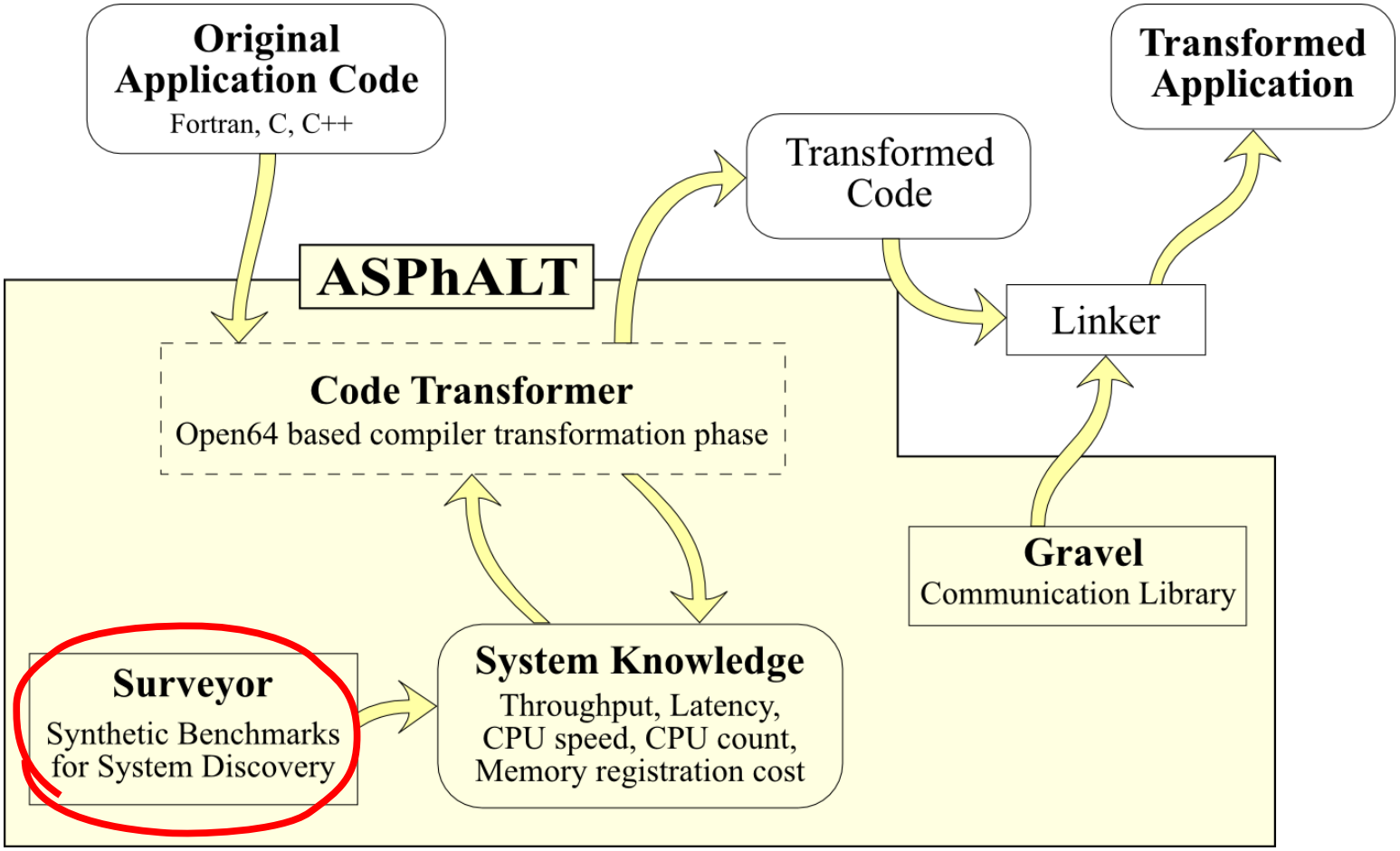
✗ Too Big

✔ Just Right

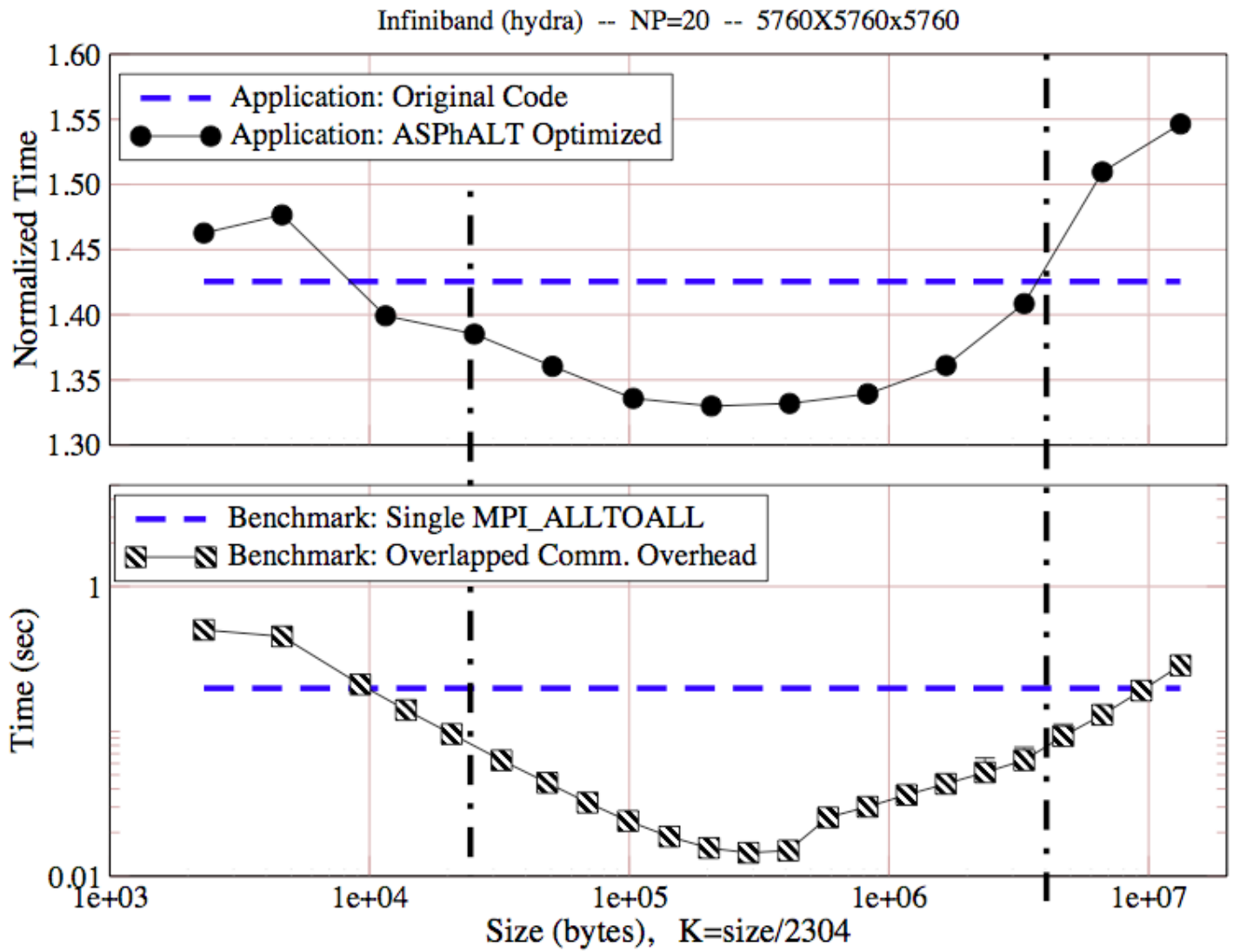
✗ Too Small

Questions ?

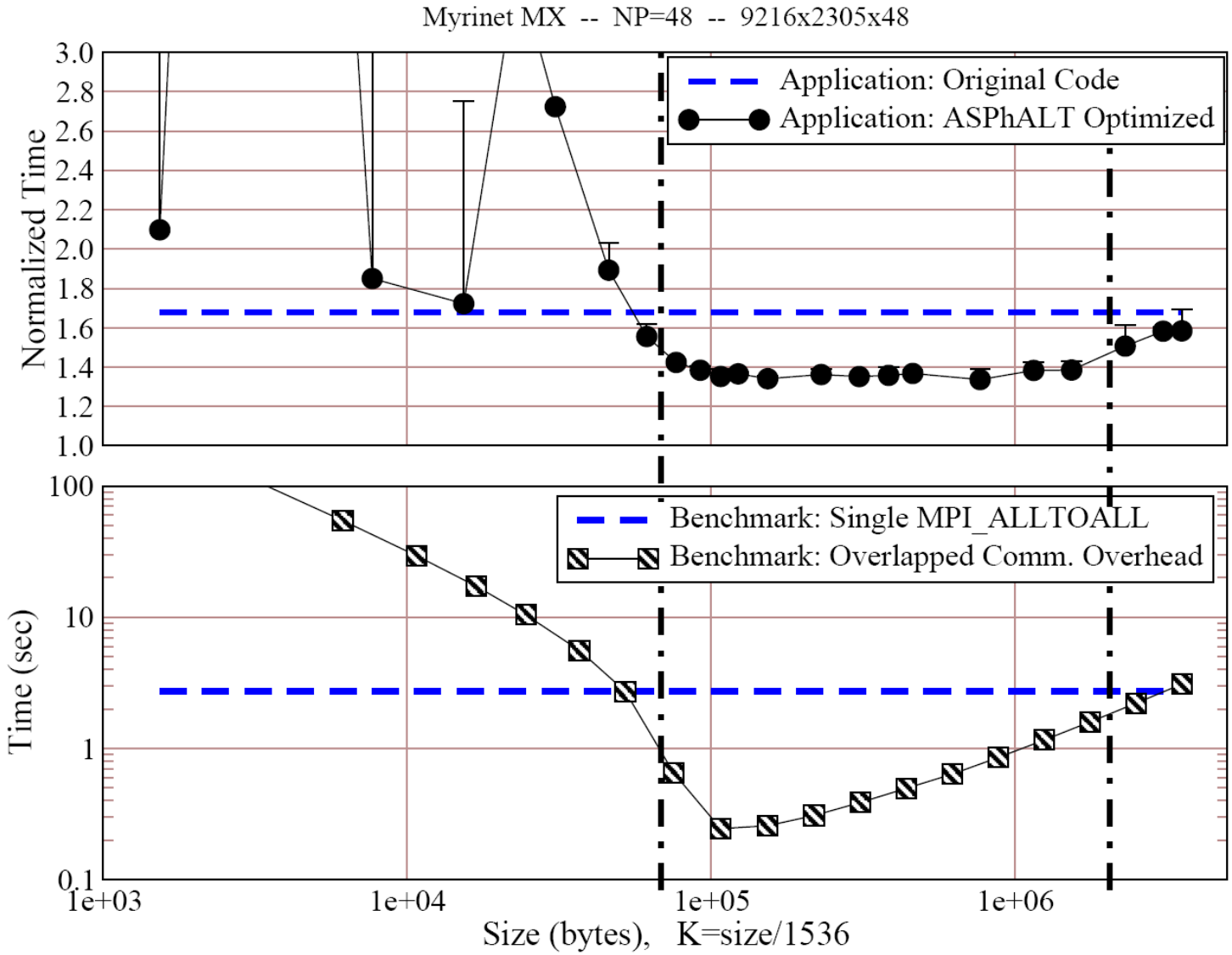
Additional Framework Components



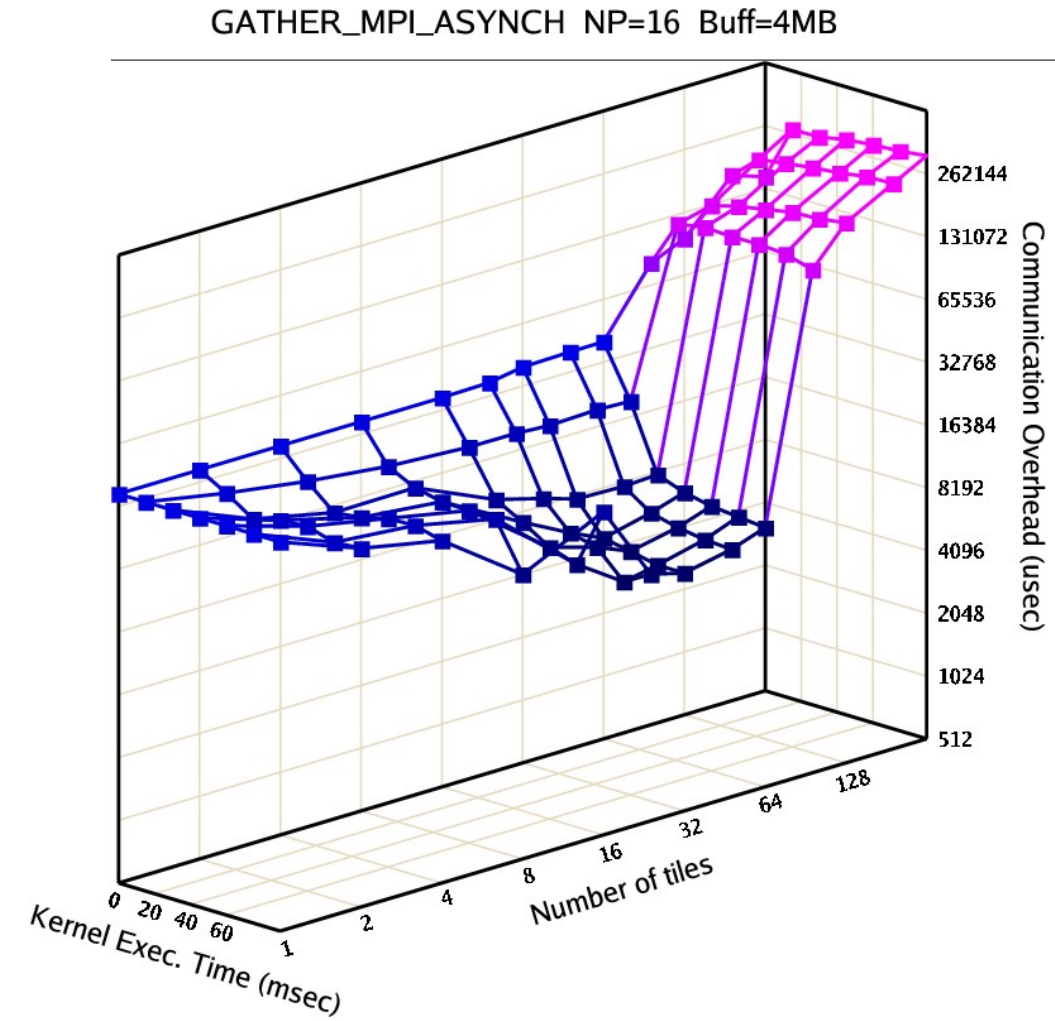
Surveyor: Using benchmark for tunning (1)

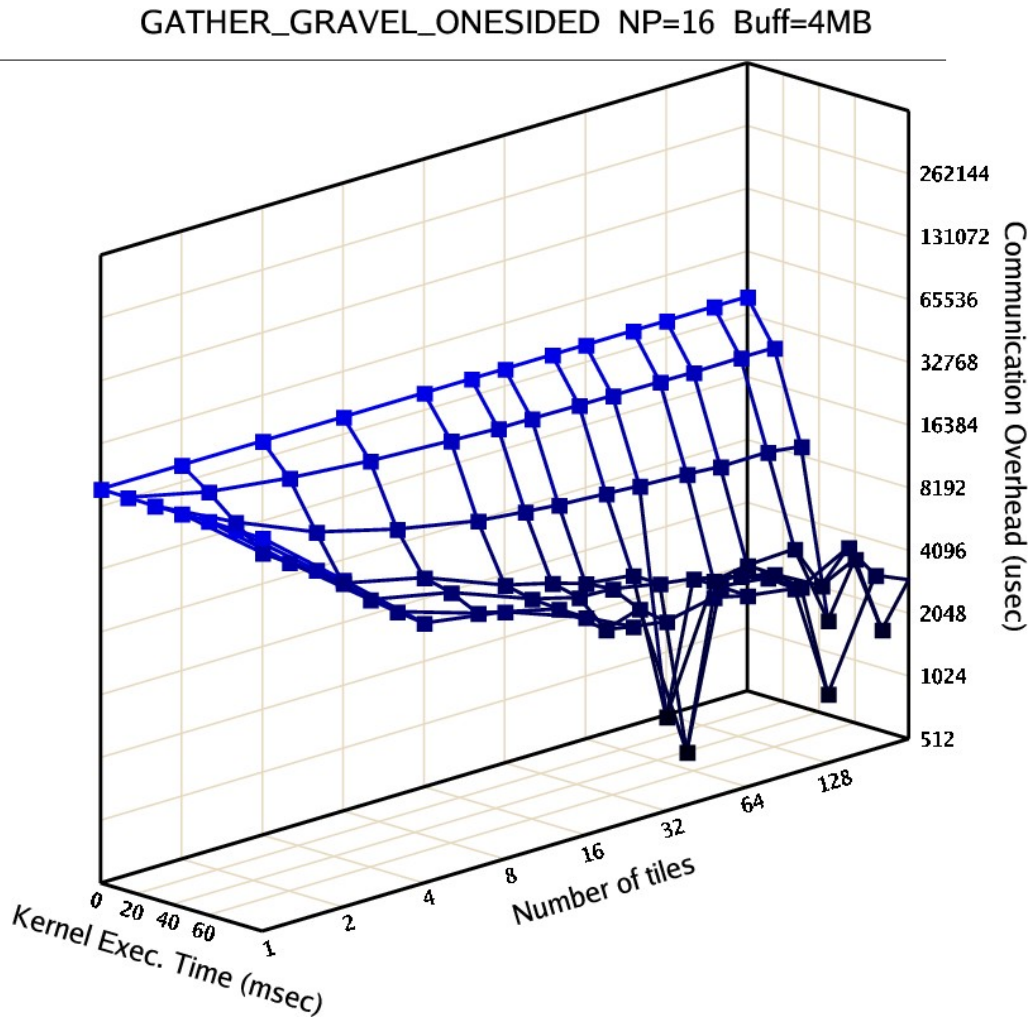


Surveyor: Using benchmark for tunning (2)



Gravel: Lightweight Communication Library (1)





Gravel: Lightweight Communication Library (2)