Automatic Code Motion to Extend MPI Nonblocking Overlap Window

Van Man Nguyen[1-4], Emmanuelle Saillard[2], Julien Jaeger[1,3],
Denis Barthou[2,4], and Patrick Carribault[1,3]

[1] CEA, DAM, DIF, F-91297, Arpajon, France
[2] Inria, Bordeaux, France
[3] Laboratoire en Informatique Haute Performance pour le Calcul et la simulation
[4] Bordeaux Institute of Technology, U. of Bordeaux, LaBRI, Bordeaux, France

Introduction

- **Need to overlap communications with computations**
  - Asynchronous communications
- **MPI nonblocking communications**
  - Init (**`MPI_I*`**) & Completion calls (**`MPI_Wait*`**)
  - Overlap communication times with computation times: insert operations between the init & the completion calls
  - Usage is still marginal & complex

Introduction

- Automatic transformation blocking → nonblocking

- Motivating Example
  - Code motion to expose more overlapping possibilities

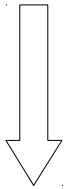
```
1  MPI_Alltoall(d1, sendcount, MPI_BYTE, d2,
2      recvcount, MPI_BYTE, MPI_COMM_WORLD);
3  matrix_multiply(a, b, res, matrix_size);
4  touch(d1);
5  matrix_multiply(a2, b2, res2, matrix_size);
```

Introduction

- Automatic transformation blocking → nonblocking

- Motivating Example
  - Code motion to expose more overlapping possibilities

```
1  MPI_Alltoall(d1, sendcount, MPI_BYTE, d2,
2      recvcount, MPI_BYTE, MPI_COMM_WORLD);
3  matrix_multiply(a, b, res, matrix_size);
4  touch(d1);
5  matrix_multiply(a2, b2, res2, matrix_size);
```



```
1  MPI_Request req;
2  MPI_Ialltoall(d1, sendcount, MPI_BYTE, d2,
3      recvcount, MPI_BYTE, MPI_COMM_WORLD, &req);
4  matrix_multiply(a, b, res, matrix_size);
5  matrix_multiply(a2, b2, res2, matrix_size);
6  MPI_Wait(&req, MPI_STATUS_IGNORE)
7  touch(d1);
```

Overlapping interval

Table of Contents

Achieving Communication-Computation Overlap

Related work

- Automatic transformation of blocking MPI calls into nonblocking
- Code motion
  - Place the init & completion calls as far as possible from each other

- Steps:
  1. Find the communications arguments & their dependencies
  2. Find an appropriate insertion point for the init & completion calls
     → Nearest data dependency
     → MPI call order
     → Control flow dependency
  3. Insert nonblocking calls

Resolving Data Dependencies (Related Work)

- Find last statement before the MPI call using the communication arguments
- Find first statement after the MPI call using the communication arguments

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Finalize();
```

deps = {

Resolving Data Dependencies (Related Work)

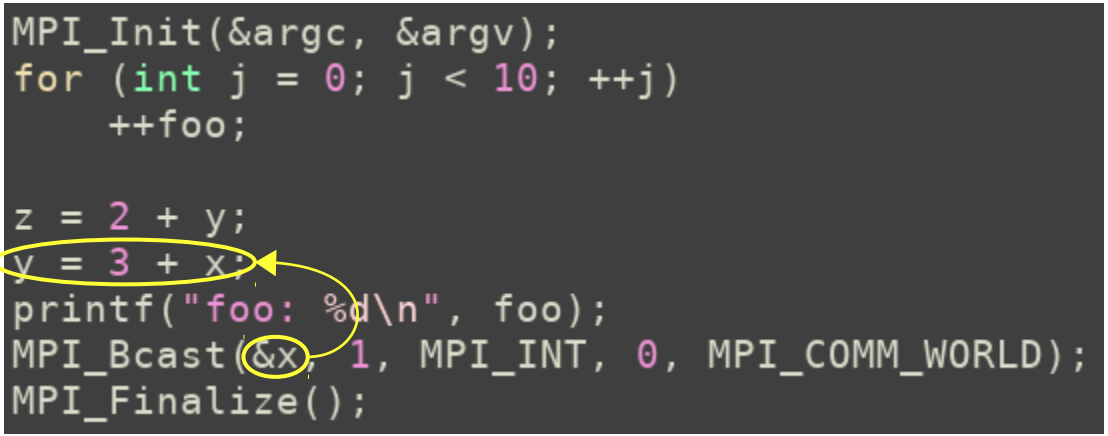
- Find last statement before the MPI call using the communication arguments
- Find first statement after the MPI call using the communication arguments

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Finalize();
```

deps = {

Resolving Data Dependencies (Related Work)

- Find last statement before the MPI call using the communication arguments
- Find first statement after the MPI call using the communication arguments




```
deps = {
y = 3 + x ;
}
```

Transforming Blocking Calls (Related Work)

- Inserting the initialization call

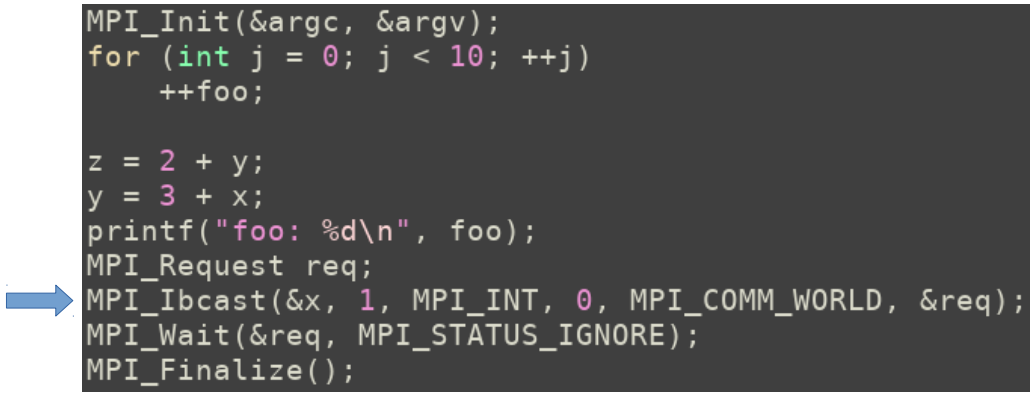
```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req
}
```

Transforming Blocking Calls (Related Work)

- Inserting the initialization call



```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req
}
```

Transforming Blocking Calls (Related Work)

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req
}
```

Transforming Blocking Calls (Related Work)

- Inserting the initialization call

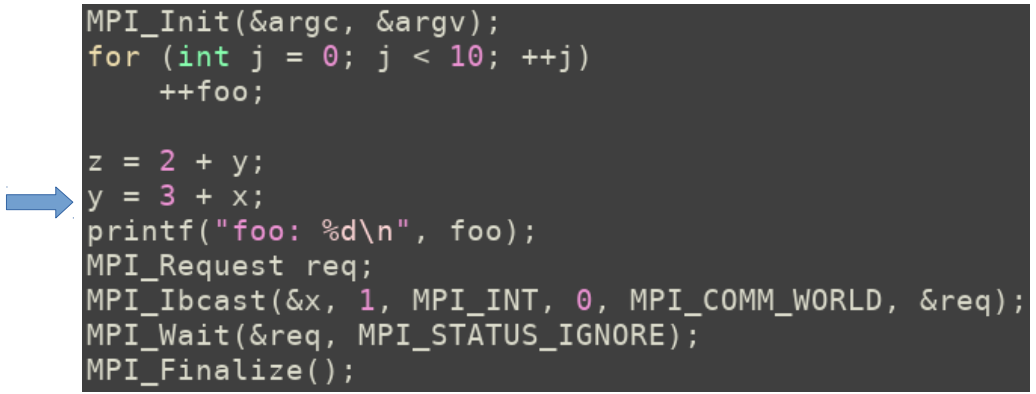
```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req
}
```

Transforming Blocking Calls (Related Work)

- Inserting the initialization call



```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

deps = {
y = 3 + x ;
}

to_move = {
MPI_Ibcast ;
MPI_Request req
}

Transforming Blocking Calls (Related Work)

- Inserting the initialization call

Insertion location
for MPI_Ibcast

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

deps = {
`y = 3 + x ;`
}

to_move = {
MPI_Ibcast ;
MPI_Request req
}

Overview of our Approach

- Automatic transformation of blocking MPI calls into nonblocking
- Code motion
  - Place the init & completion calls as far as possible from each other
  - Apply code motion to the dependency slices as well
  - => Wide overlapping interval
- Steps:
  1. Find the communications arguments & their dependencies
  2. Find an appropriate insertion point for the init & completion calls
     ~~→ Nearest dependency~~
     → MPI call order
     → Control flow scope dependency
  3. Move the dependency slices & insert nonblocking calls

Overview of our Approach

- **As we iterate over the CFG, dependencies** (*i.e. statements belonging to a slice*) **are saved**
  - Extensive code motion to move dependencies apart along with the init & completion calls
  - Preserve execution order
- **Other conditions for the insertion point remain the same**
  - Preserve MPI call order
  - Stay within the enclosing control flow scope

Resolving Data Dependencies

- Find statements needed by the communication arguments
  - Iterative computation of the backward slice
- Find statements using the communication arguments
  - Iterative computation of the forward slice

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Finalize();
```

deps = {

Resolving Data Dependencies

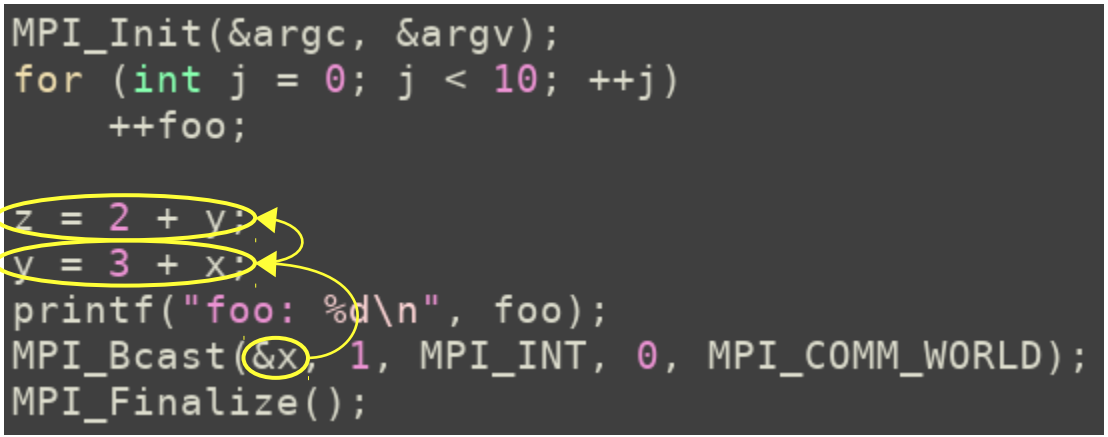
- Find statements needed by the communication arguments
  - Iterative computation of the backward slice
- Find statements using the communication arguments
  - Iterative computation of the forward slice

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Finalize();
```

deps = {
y = 3 + x ;

Resolving Data Dependencies

- Find statements needed by the communication arguments
  - Iterative computation of the backward slice
- Find statements using the communication arguments
  - Iterative computation of the forward slice




```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
y = 3 + x ;
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
y = 3 + x ;
z = 2 + y ;
}
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
y = 3 + x ;
z = 2 + y ;
}
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
y = 3 + x ;
z = 2 + y ;
}
```

Transforming Blocking Calls

- Inserting the initialization call

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
y = 3 + x ;
z = 2 + y ;
}
```

Transforming Blocking Calls

- Inserting the initialization call

Insertion location
for MPI_Ibcast

```
MPI_Init(&argc, &argv);
for (int j = 0; j < 10; ++j)
    ++foo;

z = 2 + y;
y = 3 + x;
printf("foo: %d\n", foo);
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

```
deps = {
y = 3 + x ;
z = 2 + y ;
}
```

```
to_move = {
MPI_Ibcast ;
MPI_Request req ;
y = 3 + x ;
z = 2 + y ;
}
```

Transforming Blocking Calls

- Resulting code after insertion of the initialization call

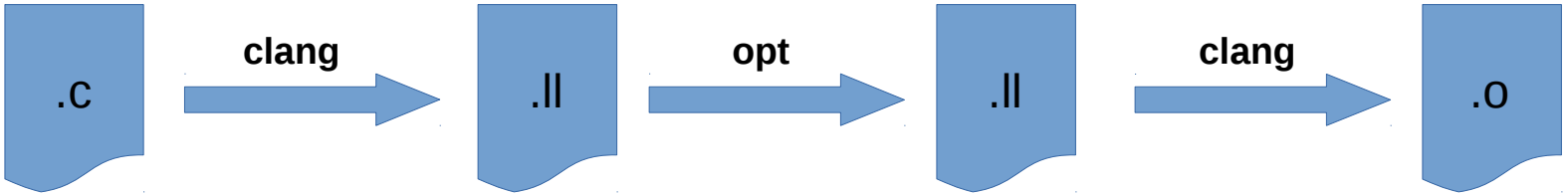
```
MPI_Init(&argc, &argv);
z = 2 + y;
y = 3 + x;
MPI_Request req;
MPI_Ibcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD, &req);
for (int j = 0; j < 10; ++j)
    ++foo;

printf("foo: %d\n", foo);
MPI_Wait(&req, MPI_STATUS_IGNORE);
MPI_Finalize();
```

Resulting overlap interval

Experimental Results

Implementation

- Intraprocedural optimization pass for LLVM 7.0.1
- Built-in analyses: loop information, (post-)dominator trees, def-use and use-def chains
- Applied on selected LLVM IR files using **opt**



Testing Environment & Evaluation

- Intel Sandy Bridge CPUs and an InfiniBand network
  - 16 cores per node
- Evaluation: duration of overlap intervals
  - Instrument timers at the borders of each overlap interval
    - Apply code motion, without performing the nonblocking transformation
- Comparison
  - Basic: Replicate state of art optimization passes by inserting at the nearest dependency
  - Extended: Apply code motion on dependencies as well

Experimental results – 1

- miniMD (v**1.2**)
  - Run parameters: EAM force, x=y=z=128
  - MPI: OpenMPI **2.0**, N=8, np=120 (15 MPI processes/Node)
- 6 calls with a significant overlapping interval (duration > 1µs)

MPI Call	File	Line	Interval duration basic (μs)	Interval duration extended (μs)
`MPI_Allreduce`	`thermo.cpp`	133	0.05	65.84
`MPI_Bcast`	`force_eam.cpp`	524	41.59	54.34
`MPI_Bcast`	`force_eam.cpp`	525	32.53	42.51
`MPI_Bcast`	`force_eam.cpp`	526	25.66	35.37
`MPI_Bcast`	`force_eam.cpp`	527	16.71	18.31
`MPI_Bcast`	`force_eam.cpp`	528	9.40	10.09
Max. MPI Call Overlap			125.94	226.46

Experimental results – 2

- miniFE (v**2.0**)
  - Run parameters: x=y=z=1024
  - MPI: OpenMPI **2.0**, N=8, np=120 (15 MPI processes/Node)
- 3 calls with a significant overlapping interval (duration > 1µs)

MPI call	File	Line	Interval duration basic (μs)	Interval duration extended (μs)
`MPI_Allreduce`	`SparseMatrix_functions.hpp`	313	0.11	4193
`MPI_Bcast`	`utils.cpp`	92	0.51	166
`MPI_Allreduce`	`make_local_matrix.cpp`	216	0.22	1.41
Max. MPI Call Overlap			0.84	4360.41

Conclusion

Conclusion

- Nonblocking communications
  - Overlapping: hide communication overheads
- Automatic creation of overlapping windows
  - Blocking → Nonblocking
  - **Added code motion of dependency slices**
  - Implemented as an LLVM pass
- Opens new opportunities for overlapping intervals
  - MiniFE : 0.11µs → 4193µs
- **Future work**:
  - Better support for existing nonblocking calls (Wait/Test matching)
  - Code motion beyond the control flow scope

Thank you for your attention