Automatic Code Generation of Distributed Parallel Tasks

CSE 2016

Nelson Lossing Corinne Ancourt François Irigoin
firstname.lastname@mines-paristech.fr

MINES ParisTech,
PSL Research University

Paris, August 24th, 2016

Scientific Program

Signal Processing

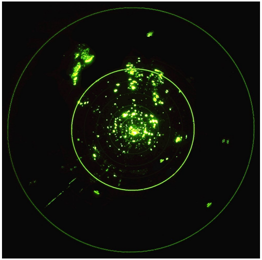
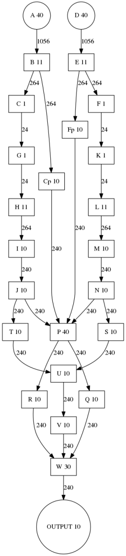


Image Processing



Context

Tools

- Automatic task parallelization (OpenModelica[1])
- Automatic distributed parallelization (Pluto+[2])
- *Black box*

But no automatic distributed parallelization task tool

[1] Mahder Gebremedhin and Peter Fritzson. "Automatic Task Based Analysis and Parallelization in the Context of Equation Based Languages". In: *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. EOOLT '14. Berlin, Germany: ACM, 2014, pp. 49–52.

[2] Uday Bondhugula. "Compiling Affine Loop Nests for Distributed-memory Parallel Architectures". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: ACM, 2013.

Source-to-Source Transformations

- Fortran code
- C code

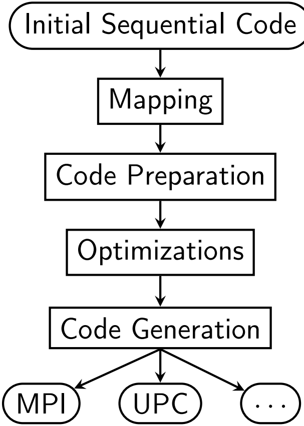
- *Scientific Program*
- *Image Processing*

- Static analyses
- Instrumentation/ Dynamic analyses
- Transformations
- Source code generation
- Code modelling
- Prettyprint

- Fortran code
- C code

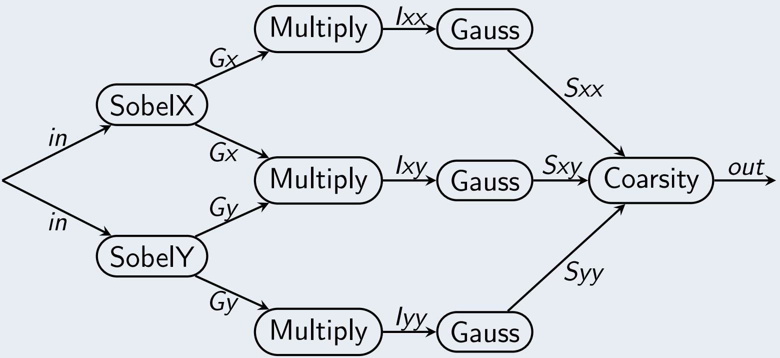
- *Distributed Parallel Code*

Compilation Process



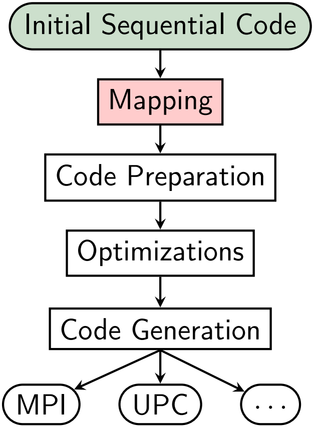
Case Example

Harris&Stephens algorithm[3]



[3] Chris Harris and Mike Stephens. "A combined corner and edge detector". In: *In Proc. of Fourth Alvey Vision Conference*. 1988, pp. 147–151

Compilation Process



Mapping

Can be done

- Automatically with a task scheduler[4]
- Manually

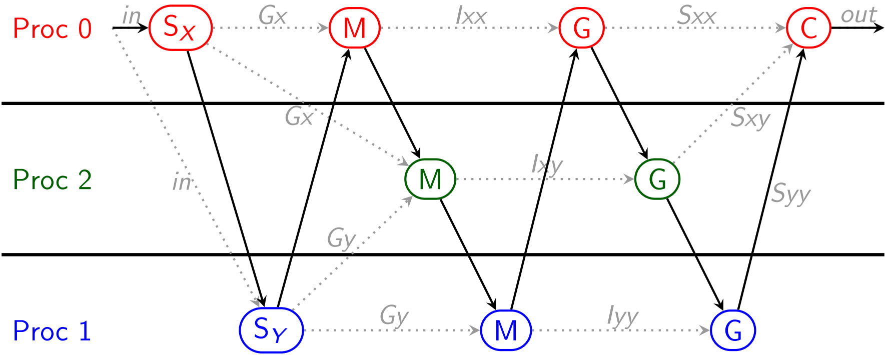
Pragma directive

- New pragma `distributed`
- On sequence of instructions, loop, test, etc.
- Not inside loop or condition
- `on_cluster` to define the process to use
- No data dependence information needed [5]

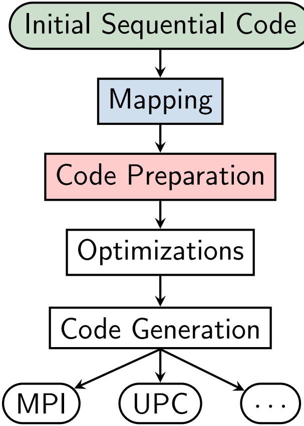
[4] Dounia Khaldi, Pierre Jouvelot, and Corinne Ancourt. "Parallelizing with BDSC, a Resource-constrained Scheduling Algorithm for Shared and Distributed Memory Systems". In: *Parallel Comput.* 41.C (Jan. 2015), pp. 66–89

[5] Martin Tillenius et al. "Resource-Aware Task Scheduling". In: *ACM Trans. Embed. Comput. Syst.* 14.1 (Jan. 2015), 5:1–5:25

After Mapping



Compilation Process



Code Preparation

Task on process

1. Add declaration for each variable on each process
2. Add copy/communication for written variables
3. Substitute "original variables" by "local variables"
4. Remove "original variables" declarations

Code Preparation

Task on process

1. **Add declaration for each variable on each process**
2. Add copy/communication for written variables
3. Substitute "original variables" by "local variables"
4. Remove "original variables" declarations

```
int x;        ⇒   int x;
                  int x_0;
                  int x_1;
                  int x_2;
                  ...
```

Code Preparation

Task on process

1. Add declaration for each variable on each process
2. **Add copy/communication for written variables**
3. Substitute "original variables" by "local variables"
4. Remove "original variables" declarations

Copy/Communication

- inside the task
  - More precise
  - Issue for code generation on dynamic cases
- between the tasks
  - No dynamic cases
  - Less precise

Code Preparation

Task on process

1. Add declaration for each variable on each process
2. Add copy/communication for written variables
3. **Substitute "original variables" by "local variables"**
4. Remove "original variables" declarations

```
#pragma distributed on_cluster 0
{                                          {
   ...                                        ...
   x=0;                                       x_0=0;
   ...                                  ⇒     ...
   x_0=x;                                     x_0=x_0;
   x_1=x;                                     x_1=x_0;
   x_2=x;                                     x_2=x_0;
   ...                                        ...
}                                          }
```

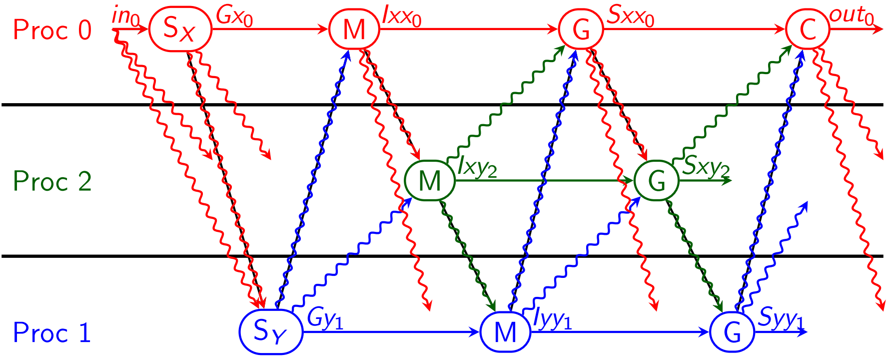
Code Preparation

Task on process

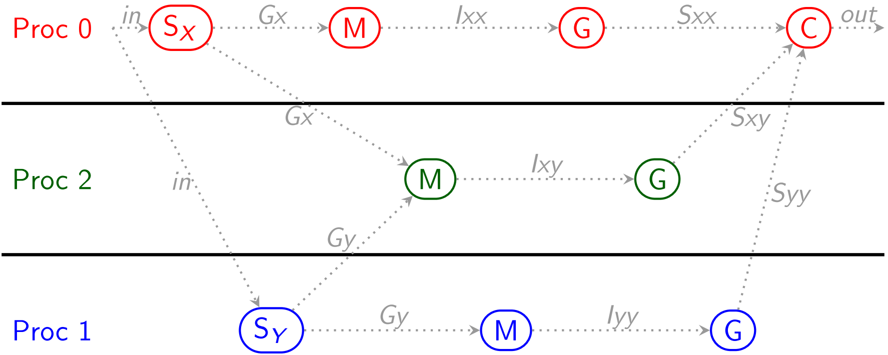
1. Add declaration for each variable on each process
2. Add copy/communication for written variables
3. Substitute "original variables" by "local variables"
4. **Remove "original variables" declarations**

```
int x;
int x_0;                int x_0;
int x_1;          ⇒     int x_1;
int x_2;                int x_2;
...                     ...
```

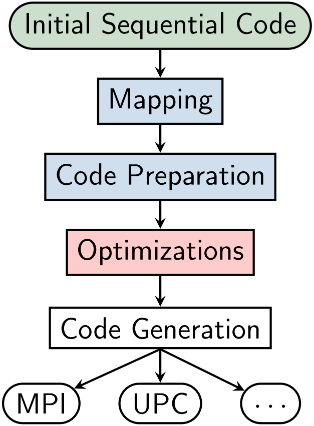
After Code Preparation



Task Graph


Proc 0
in
S_X
Gx
M
Ixx
G
Sxx
C
out
Gx
Sxy
Proc 2
in
M
Ixy
G
Syy
Gy
Proc 1
S_Y
Gy
M
Iyy
G

Compilation Process



Optimizations

Reduce Copy/Communication

- Dead-code Elimination
- Dead-iteration Elimination

Reduce Memory Footprint

- Array Resizing

Reduce Copy/Communication

- Dead-code Elimination
- Dead-iteration Elimination

	case 1	case 2	case 3
task on P	write x_P ~~x_Q = x_P~~ x_R = x_P	write x_P x_Q = x_P ~~x_R = x_P~~	write a_P[0..n] ~~a_Q[0..n]=a_P[0..n]~~ a_Q[0..n/2]=a_P[0..n/2] ~~a_R[0..n]=a_P[0..n]~~ a_R[n/2..n]=a_P[n/2..n]
task on Q Q≠P	...	read x_Q write x_Q x_P = x_Q x_R = x_Q	read a_Q[0..n/2]
task on R R≠{P,Q}	read x_R	read x_R	read a_R[n/2..n]

Reduce Memory Footprint

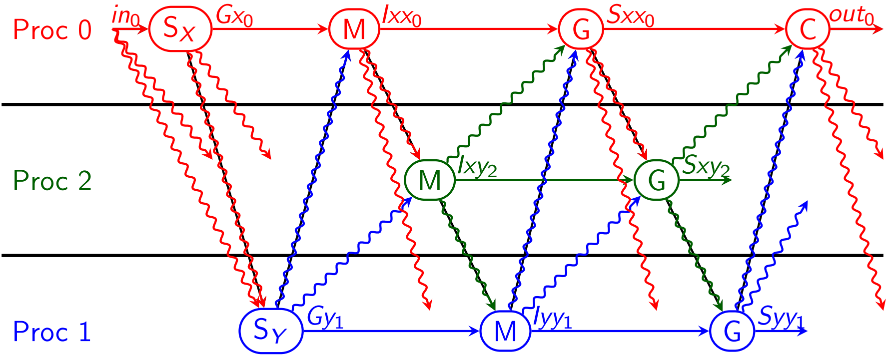
- Array Resizing
  - Compute new array size
  - Resize array declarations
  - Shift array cells access

```c
int a[20];
int i;
for (i=5; i<15; i++)
  a[i] = i*i;
```

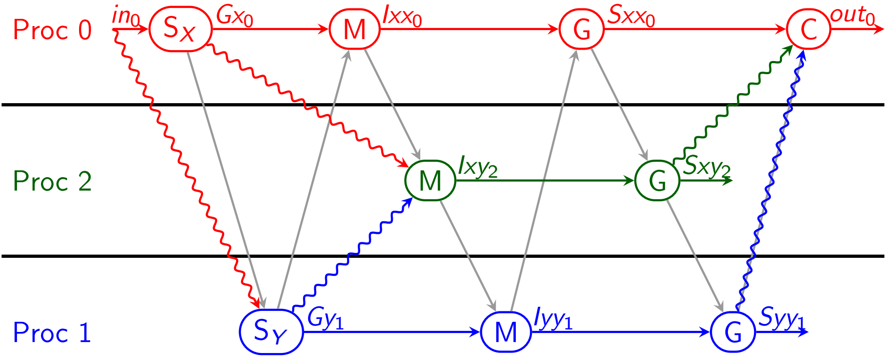
$\Rightarrow$

```c
int a[10];
int i;
for (i=5; i<15; i++)
  a[i-5] = i*i;
```

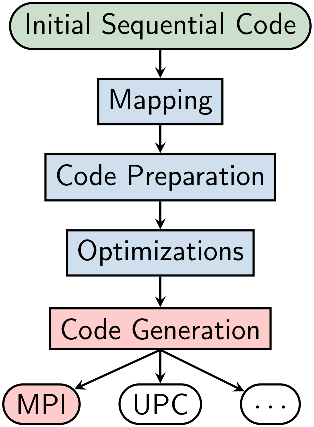
Before Optimizations



After Optimizations



Compilation Process



MPI Code Generation

- Configure MPI Environment
- Replace pragma block by test on process rank
- Replace copy by
  - Send message for rhs on rhs process to lhs process
  - Receive message for lhs on lhs process from rhs process

Parallel Code Generation

MPI Code Generation

- **Configure MPI Environment**
- Replace pragma block by test on process rank
- Replace copy by
  - Send message for rhs on rhs process to lhs process
  - Receive message for lhs on lhs process from rhs process

```
MPI_Status status;
int size, rank;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (size<REQUIRE_PROC_NUMBER) {
  printf("Not enough processes launched!");
  MPI_Finalize();
  return 0;
}
...
MPI_Finalize();
return 0;
```

Parallel Code Generation

MPI Code Generation

- Configure MPI Environment
- **Replace pragma block by test on process rank**
- Replace copy by
  - Send message for rhs on rhs process to lhs process
  - Receive message for lhs on lhs process from rhs process

```
#pragma distributed on_cluster 0
{
  ...
}
```

⇒

```
if (rank==0)
{
  ...
}
```

Parallel Code Generation

MPI Code Generation

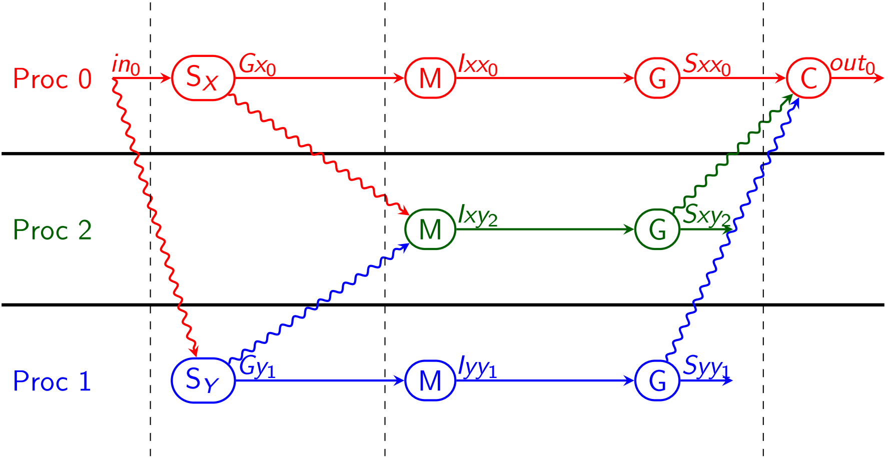
- Configure MPI Environment
- Replace pragma block by test on process rank
- **Replace copy by**
  - **Send message for rhs on rhs process to lhs process**
  - **Receive message for lhs on lhs process from rhs process**

```
x_1 = x_0;
```

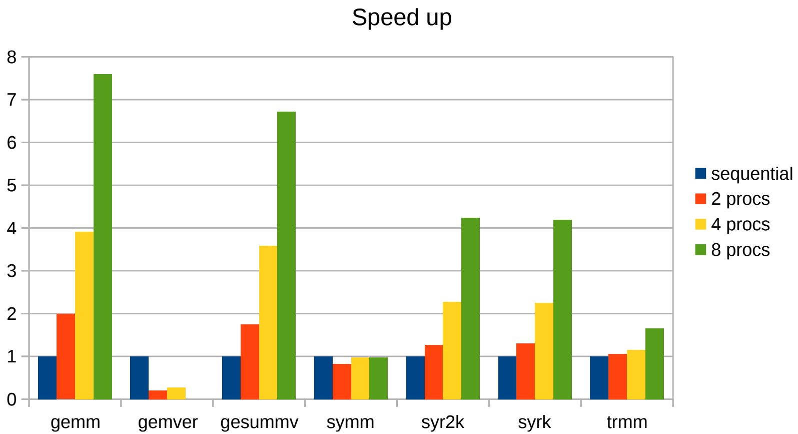
$\Rightarrow$

```
if (rank==0)
  MPI_Send(&x_0, 1, MPI_DOUBLE, 1, 0,
           MPI_COMM_WORLD);
if (rank==1)
  MPI_Recv(&x_1, 1, MPI_DOUBLE, 0, 0,
           MPI_COMM_WORLD);
```

After Parallel Code Generation



Experimental Results



Benchmark: BLAS in Polybench
size: ∼3000x4000
type: double

Limitations

General Limitations

- Number of processes known at the beginning
- No dynamic parallelism
- Communication overestimation in case of dynamic communications

Experimental Limitations

- Strongly mapping dependent

Conclusion

Achievement

- Automatic source-to-source transformations
- Succession of simple transformations
- Basic communication functions
- Provable transformations
- Good efficiency

Future Work

- Improvement of the initial mapping by loop rescheduling
- Asynchronous communications instead of synchronous communications

Automatic Code Generation of Distributed Parallel Tasks

CSE 2016

Nelson Lossing Corinne Ancourt François Irigoin
`firstname.lastname@mines-paristech.fr`

MINES ParisTech,
PSL Research University

Paris, August 24th, 2016