Automatic Code Generation of Distributed Parallel Tasks

CSE 2016

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

MINES ParisTech

MINES ParisTech, PSL Research University

Paris, August 24th, 2016
Motivation

Scientific Program

Signal Processing

Image Processing
Tools

- Automatic task parallelization (OpenModelica\textsuperscript{1})
- Automatic distributed parallelization (Pluto+\textsuperscript{2})
- *Black box*

But no automatic distributed parallelization task tool


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

Initial Sequential Code → Mapping → Code Preparation → Optimizations → Code Generation

MPI  UPC  ...
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

1. Initial Sequential Code
2. Mapping
3. Code Preparation
4. Optimizations
5. Code Generation
6. MPI
7. UPC
8. ...
Mapping

Can be done
- Automatically with a task scheduler
- 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

---


After Mapping

Proc 0

Proc 2

Proc 1

\(S_X\) \(\rightarrow\) \(G\) \(\rightarrow\) \(C\)

\(S_Y\) \(\rightarrow\) \(M\) \(\rightarrow\) \(G\)

\(G_x\) \(\rightarrow\) \(I_{xx}\) \(\rightarrow\) \(S_{xx}\)

\(G_y\) \(\rightarrow\) \(I_{xy}\) \(\rightarrow\) \(S_{xy}\)

\(I_{yy}\) \(\rightarrow\) \(S_{yy}\)
Compilation Process

Initial Sequential Code

Mapping

Code Preparation

Optimizations

Code Generation

MPI

UPC

···
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
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

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

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

```c
⇒
int x_0;
int x_1;
int x_2;
...
```
After Code Preparation
Compilation Process

Initial Sequential Code

Mapping

Code Preparation

Optimizations

Code Generation

MPI

UPC

⋯
Optimizations

- Reduce Copy/Communication
  - Dead-code Elimination
  - Dead-iteration Elimination

- Reduce Memory Footprint
  - Array Resizing
Reduce Copy/Communication

- Dead-code Elimination
- Dead-iteration Elimination

<table>
<thead>
<tr>
<th>task on P</th>
<th>case 1</th>
<th>case 2</th>
<th>case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>write x_P</td>
<td>write x_P</td>
<td>write a_P[0..n]</td>
<td></td>
</tr>
<tr>
<td>x_Q = x_P</td>
<td>x_Q = x_P</td>
<td>a_Q[0..n]=a_P[0..n]</td>
<td></td>
</tr>
<tr>
<td>x_R = x_P</td>
<td>x_R = x_P</td>
<td>a_Q[0..n/2]=a_P[0..n/2]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>task on Q</th>
<th>case 1</th>
<th>case 2</th>
<th>case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q≠P</td>
<td>read x_Q</td>
<td>read a_Q[0..n/2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write x_Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_P = x_Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_R = x_Q</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>task on R</th>
<th>case 1</th>
<th>case 2</th>
<th>case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R≠{P,Q}</td>
<td>read x_R</td>
<td>read a_R[n/2..n]</td>
<td></td>
</tr>
</tbody>
</table>
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;

⇒
int a[10];
int i;
for (i=5; i<15; i++)
  a[i-5] = i*i;
```
Before Optimizations
After Optimizations
Compilation Process

- Initial Sequential Code
- Mapping
- Code Preparation
- Optimizations
- Code Generation

Options:
- MPI
- UPC
- ...
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
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

```c
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

```c
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);

x_1 = x_0;
⇒
```

20 / 24
After Parallel Code Generation

Proc 0

Proc 2

Proc 1

\( S_X \) \( G_{x_0} \) \( M \) \( I_{xx_0} \) \( G \) \( S_{xx_0} \) \( C \) \( out_0 \)

\( S_Y \) \( G_{y_1} \) \( M \) \( I_{yy_1} \) \( G \) \( S_{yy_1} \)

\( in_0 \)
Experimental Results

Benchmark: BLAS in Polybench
size: $\sim 3000 \times 4000$
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

MINES ParisTech,
PSL Research University

Paris, August 24th, 2016