Turning Control Flow Graphs into Callgraphs

Transformation of partitioned codes for execution in heterogeneous architectures

PABLO BARRIO
TOBIAS KENTER
CARLOS CARRERAS
CHRISTIAN PLESSL
ROBERTO SIERRA
Outline

1. Heterogeneous High Performance Computing

2. Compilation toolchain

3. Code refactoring for execution in heterogeneous platforms
Outline

1. Heterogeneous High Performance Computing

2. Compilation toolchain

3. Code refactoring for execution in heterogeneous platforms
### High Performance Computing & Embedded Systems

<table>
<thead>
<tr>
<th></th>
<th>Embedded</th>
<th>HPC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type of processors</strong></td>
<td>Heterogeneous</td>
<td>Homogeneous</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Small</td>
<td>Massive</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>Shared</td>
<td>Distributed</td>
</tr>
</tbody>
</table>

but getting closer every day...
Objectives

• A code partitioner for heterogeneous architectures.

• Easy to add models for new devices and architectures.

• Partitioning based on software and hardware characteristics.

• Communications generated for distributed memory systems.

• Automatic parallelization, both functional and data parallel.
The solution under research

C, C++, Fortran...

FRONT END

LLVM IR

optimization passes

BACK END

LLVM IR

asm, VHDL...

Profiling

Estimation

Partitioning & Mapping

Code refactoring

Turning CFGs into callgraphs
Outline

1. Heterogeneous High Performance Computing

2. Compilation toolchain

3. Code refactoring for execution in heterogeneous platforms
LLVM-based compilation toolchain

Module 1
  .
  .
  .

Module N

Front ends

opt & link

Linked module

Profiling?
  yes
  lli
  no

Exe 1
  .
  .

Exe M

Back end 1

Module 1

Code refactoring

Partitioning & mapping

Estimation

Profile info

RSD file

Data in

Module 1

Module M

Module M

Module M

Module M

Turning CFGs into callgraphs
[PartitioningPass] PARTITIONING OVERVIEW:
Initial exec time was 1.81e-07 s, new is 1.06e-07
-- Speedup = 1.71e+00

[PartitionWriterPass] Generating partitioned code
PartitionWriterPass::runOnModule() -- Original functions:
  odd with BBs:
    entry --> CPU
  main with BBs:
    entry --> CPU
    3 --> CPU
    4 --> CPU
    beforeHeader --> CPU
    5 --> CPU
    6 --> CPU
    7 --> CPU
    8 --> CPU_SIMD
    9 --> CPU_SIMD
    11 --> CPU_SIMD
    12 --> CPU_SIMD
    13 --> CPU
    14 --> CPU
    afterHeader --> CPU

...
Outline

1. Heterogeneous High Performance Computing

2. Compilation toolchain

3. Code refactoring for execution in heterogeneous platforms
Function-based control flow

BB_A:
  ...
  jmp BB_B

BB_B:
  ...
  jne A, C

BB_C:
  ...

BB_A:
  ...
  call B()
  ret

A()

BB_B:
  ...
  jne callA, callC

BB_C:
  ...

BB_C:
  ...

callA:
  ...
  call A()
  ret

callC:
  ...
  call C()
  ret

callA:
  ...
  call A()
  ret
Refactoring methodology

duplicate constants

distribute globals

for every original function \( f \)

\[
\text{initiatorList} \leftarrow \text{find initiators}(f)
\]

create new functions(\( f, \text{initiatorList} \))

fix branches(\( \text{initiatorList} \))

fix phi nodes(\( \text{initiatorList} \))
Refactoring methodology

duplicate constants

distribute globals

for every original function f

\[ \text{initiatorList} \leftarrow \text{find initiators}(f) \]

create new functions(\(f, \text{initiatorList}\))

fix branches(\text{initiatorList})

fix phi nodes(\text{initiatorList})
Initiator list ← find initiators(f)

Partitioning result

Initiators

Resulting functions

Turning CFGs into callgraphs
Refactoring methodology

duplicate constants

distribute globals

for every original function f

    initiatorList ← find initiators(f)

    create new functions(f, initiatorList)

    fix branches(initiatorList)

    fix phi nodes(initiatorList)
create new functions (f, initiatorList)

MODULE 1

i32 f(i8* %num)

BB_A:
%3 = add i32 %2, 6
jmp BB_B

BB_B:
%4 = mul i32 %3, %3
jne BB_A, BB_C

BB_C:
ret call i32 @puts(%num)

MODULE 2

declare i32 @puts(i8*)

DEV 1

DEV 2

Declare used functions in the destination module

Turning CFGs into callgraphs

16
Turning CFGs into callgraphs
create new functions \((f, \text{initiatorList})\)

- **MODULE 1**
  - `i32 f(i8* %num)`
  - **BB_A:**
    - `%3 = add i32 %2, 6`
    - `jmp BB_B`

- **MODULE 2**
  - `declare i32 @puts(i8*)`
  - `i32 f2(i8* %arg1, i32 %arg2)`
  - **BB_B:**
    - `%4 = mul i32 %3, %3`
    - `jne BB_A, BB_C`
  - **BB_C:**
    - `ret call i32 @puts(%num)`
create new functions (f, initiatorList)

MODULE 1

```assembly
i32 f(i8* %num)

BB_A:
  %3 = add i32 %2, 6
  jmp BB_B
```

Fix argument uses

MODULE 2

```assembly
declare i32 @puts(i8*)

i32 f2(i8* %arg1, i32 %arg2)

BB_B:
  %4 = mul i32 %arg2, %arg2
  jne BB_A, BB_C

BB_C:
  ret call i32 @puts(%arg1)
```
Refactoring methodology

duplicate constants

distribute globals

for every original function f

initiatorList ← find initiators(f)

create new functions(f, initiatorList)

fix branches(initiatorList)

fix phi nodes(initiatorList)
fix branches (initiatorList)

MODULE 1

i32 f(i8* %num)

BB_A:
%3 = add i32 %2, 6
%r = call i32 f2(%num, %3)
ret %r

MODULE 2

i32 f2(i8* %arg1, i32 %arg2)

BB_B:
%4 = mul i32 %arg2, %arg2
jne fcaller, BB_C

BB_C:
ret call i32 @puts(%arg1)

fcaller:
%r = call i32 f(%num, %3)
ret %r

Replace old branches by function calls
Refactoring methodology

duplicate constants

distribute globals

for every original function \( f \)

\[
\text{initiatorList } \leftarrow \text{find initiators}(f)
\]

create new functions\( (f, \text{initiatorList}) \)

fix branches\( (\text{initiatorList}) \)

fix phi nodes\( (\text{initiatorList}) \)
Loops generate recursive calls

BB_A:
...  
jmp BB_B

BB_B:
...  
jne A, C

BB_C:
...

BB_A:
...  
call B()  
ret

BB_B:
...  
jne callA, callC

BB_C:
...

callA:
...  
call A()  
ret

callC:
...  
call C()  
ret

STACK A  
vars ret vars ret vars ret vars ret  
• • •  
vars ret vars ret vars ret vars

STACK B  
ret vars ret vars ret vars ret  
• • •  
ret vars ret vars ret vars ret
Fixing loop recursion: a loop pass

header:
%3 = add i32 %2, 6
br label %latch

latch:
%4 = mul i32 %3, %3
%cond = icmp ne %4, 0
br i1 %cond, label %header, label %exit

exit:
ret call i32 @puts(%num)

preheader:
%cmpRes = alloca i1
store i1 true, i1* %cmpRes
br label %header

header:
%cond = load i1* %cmpRes
br i1 %cond, label %postheader, label %"exit"

postheader:
%3 = add i32 %2, 6
jmp label %latch

latch:
%4 = mul i32 %3, %3
%cond = icmp ne %4, 0
store i1 %cond, i1* cmpRes
br label %header

exit:
ret call i32 @puts(%num)
Fixing loop recursion: final code refactoring

header:
%cond = load i1* %cmpRes
br i1 %cond, label %postheader, label "%exit"

postheader:
%3 = add i32 %2, 6
jmp label %latch

latch:
%4 = mul i32 %3, %3
%cond = icmp ne %4, 0
store i1 %cond, i1* cmpRes
br label %header

exit:
ret call i32 @puts(%num)

f() preheader:
%cmpRes = alloca i1
store i1 true, i1* %cmpRes
br label %header

preheader:
%cmpRes = alloca i1
store i1 true, i1* %cmpRes
br label %header

header:
%cond = load i1* %cmpRes
br i1 %cond, label %postheader, label "%cal"

postheader:
%3 = add i32 %2, 6
call latch()
br label %header

cal:
call exit()

latch() latch:
%4 = mul i32 %3, %3
%cond = icmp ne %4, 0
store i1 %cond, i1* cmpRes
ret

exit() exit:
ret call i32 @puts(%num)
Output from the tool

Time profiling hello.ir
[HPCmap] Parsing module hello.ir...
[ReadArchPass] Parsing architecture ../architectures/CPU_SIMD.arch...
[EstimationPass] Estimating from profiling information...

[PartitioningPass] PARTITIONING OVERVIEW:
[PartitioningPass] Initial exec time was 1.81e-07 s, new is 1.06e-07 -- Speedup = 1.71e+00
[LoopRecursionBreakPass] Analyzing loop 5 <-> 12
[PartitionWriterPass] Generating partitioned code

PartitionWriterPass::runOnModule() -- Original module's functions:
          odd with BBs:
                          entry --> CPU
          main with BBs:
                          entry --> CPU
                          3 --> CPU

...  

PartitionWriterPass::find_initiators() -- Inspecting function main()
          Trivial initiators:
                          5
                          8
          Entry block initiator: entry
          Nontrivial initiators:
                          14

...  

PartitionWriterPass::create_new_Fs() -- Splitting up function main
          Function main1_CPU inserted in module CPU.part
          Moving BB 14 from function main to function main1_CPU

...  

PartitionWriterPass::branches_to_fcalls() -- Fixing branches:
                      to BB entry, moved to function main
                      to BB 14, moved to function main1_CPU

PartitionWriterPass::fix_initiator_phis() -- Initiators:
          main2_CPU::5
                          2 phis updated

[PartitionWriterPass] Module CPU.part generated
[PartitionWriterPass] Module CPU_SIMD.part generated
Partitioned hello.ir
Preliminary results

Turning CFGs into callgraphs

<table>
<thead>
<tr>
<th></th>
<th>ctrl</th>
<th>arrayops</th>
<th>adi</th>
<th>floyd-warshall</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>partitioned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vectorized</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>part + full vect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>part + semi vect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Execution time (s)
Conclusions

• Compilation toolchain for heterogeneous architectures

• Code refactoring based on splitting functions into smaller ones.

• Removed recursion generated by loops being transformed into functions.

• The function call approach does not introduce a significant overhead so far.
Work in progress…

IN THE REFACTORIZING PASS

Execute in a real architecture (one executable per device)
Distributed memory
Automatic communications

IN THE COMPLETE TOOLCHAIN

Identification of parallelism
Data partitioning
Improve estimation, partitioning heuristics, profiling…
Time profiling hello.ir
[HPCmap] Parsing module hello.ir...
[ReadArchPass] Parsing architecture ../architectures/CPU_SIMD.arch...
[EstimationPass] Estimating from profiling information...
[PartitioningPass] Partitioning...
[PartitioningPass] PARTITIONING OVERVIEW:
[PartitioningPass] Initial exec time was 1.81e-07 s, new is 1.06e-07 -- Speedup = 1.71e+00
[LoopRecursionBreakPass] DONE
[LoopRecursionBreakPass] Analyzing Loop 6 <-> 6
[LoopRecursionBreakPass] DONE
[PartitionWriterPass] Generating partitioned code
PartitionWriterPass::runOnModule() -- Original module's functions:
  odd with BBs:
    entry --> CPU
  main with BBs:
    entry --> CPU
    3 --> CPU
    beforeHeader --> CPU
    8 --> CPU_SIMD
    9 --> CPU_SIMD
    13 --> CPU
    afterHeader --> CPU
  puts with BBs:
PartitionWriterPass::find_initiators() -- Inspecting function main()
  Trivial initiators:
    5
    11
  Entry block initiators:
  Nontrivial initiators:
    14
  Results:
    entry has initiator entry
    beforeHeader has initiator entry
    5 has initiator 5
    11 has initiator 11
    12 has initiator 11
PartitionWriterPass] Module CPU.part generated
PartitionWriterPass] Module CPU_SIMD.part generated
Partitioned hello.ir

Turning CFGs into callgraphs
Time profiling hello.ir
[HPCmap] Parsing module hello.ir...
[ReadArchPass] Parsing architecture ../architectures/CPU_SIMD.arch...
[EstimationPass] Estimating from profiling information...
[PartitioningPass] Partitioning...
[PartitioningPass] PARTITIONING OVERVIEW:
[PartitioningPass] Initial exec time was 1.81e-07 s, new is 1.06e-07 -- Speedup = 1.71e+00
[LoopRecursionBreakPass] DONE
[LoopRecursionBreakPass] Analyzing loop 6 <-> 6
[LoopRecursionBreakPass] DONE
[PartitionWriterPass] Generating partitioned code
PartitionWriterPass::runOnModule() -- Original module's functions:
  odd with BBs:
    entry --> CPU
  main with BBs:
    entry --> CPU
    3 --> CPU
    beforeHeader --> CPU
    8 --> CPU_SIMD
    9 --> CPU_SIMD
    13 --> CPU
    afterHeader --> CPU
  puts with BBs:
PartitionWriterPass::find_initiators() -- Inspecting function main() 
Trivial initiators:
  5
  11
Entry block initiators:
Entry block initiators:
Nontrivial initiators:
  14
Results:
  entry has initiator entry
  beforeHeader has initiator entry
  5 has initiator 5
  11 has initiator 11
  12 has initiator 11
[PartitionWriterPass] Module CPU.part generated
[PartitionWriterPass] Module CPU_SIMD.part generated
Partitioned hello.ir
Turning CFGs into callgraphs
Time profiling hello.ir
[HPCmap] Parsing module hello.ir...
[ReadArchPass] Parsing architecture ../architectures/CPU_SIMD.arch...
[EstimationPass] Estimating from profiling information...
[PartitioningPass] Partitioning...
[PartitioningPass] PARTITIONING OVERVIEW:
[PartitioningPass] Initial exec time was 1.81e-07 s, new is 1.06e-07 -- Speedup = 1.71e+00
[LoopRecursionBreakPass] DONE
[LoopRecursionBreakPass] Analyzing loop 6 <-> 6
[LoopRecursionBreakPass] DONE
[PartitionWriterPass] Generating partitioned code
PartitionWriterPass::runOnModule() -- Original module's functions:
  odd with BBs:
    entry --> CPU
  main with BBs:
    entry --> CPU
    3 --> CPU
    beforeHeader --> CPU
    8 --> CPU_SIMD
    9 --> CPU_SIMD
    13 --> CPU
    afterHeader --> CPU
  puts with BBs:
PartitionWriterPass::find_initiators() -- Inspecting function main()
  Trivial initiators:
    5
    11
  Entry block initiators:
  Nontrivial initiators:
    14
  Results:
    entry has initiator entry
    beforeHeader has initiator entry
    5 has initiator 5
    11 has initiator 11
    12 has initiator 11
[PartitionWriterPass] Module CPU.part generated
[PartitionWriterPass] Module CPU_SIMD.part generated
Partitioned hello.ir

Turning CFGs into callgraphs
Time profiling hello.ir
[HPCmap] Parsing module hello.ir...
[ReadArchPass] Parsing architecture ../architectures/CPU_SIMD.arch...
[EstimationPass] Estimating from profiling information...
[PartitioningPass] Partitioning...
[PartitioningPass] PARTITIONING OVERVIEW:
[PartitioningPass] Initial exec time was 1.81e-07 s, new is 1.06e-07 -- Speedup = 1.71e+00
[LoopRecursionBreakPass] DONE
[LoopRecursionBreakPass] Analyzing loop 6 <-> 6
[LoopRecursionBreakPass] DONE
[PartitionWriterPass] Generating partitioned code
PartitionWriterPass::runOnModule() -- Original module's functions:
  odd with BBs:
    entry --> CPU
  main with BBs:
    entry --> CPU
    3 --> CPU
    beforeHeader --> CPU
    8 --> CPU_SIMD
    9 --> CPU_SIMD
    13 --> CPU
    afterHeader --> CPU
  puts with BBs:
PartitionWriterPass::find_initiators() -- Inspecting function main()
Trivial initiators:
  5
  11
Entry block initiators:
Nontrivial initiators:
  14
Results:
  entry has initiator entry
  beforeHeader has initiator entry
  5 has initiator 5
  11 has initiator 11
  12 has initiator 11
[PartitionWriterPass] Module CPU.part generated
[PartitionWriterPass] Module CPU_SIMD.part generated
Partitioned hello.ir

Turning CFGs into callgraphs
Time profiling hello.ir
[HPMap] Parsing module hello.ir...
[ReadArchPass] Parsing architecture ../architectures/CPU_SIMD.arch...
[EstimationPass] Estimating from profiling information...
[PartitioningPass] Partitioning...
[LoopRecursionBreakPass] PARTITIONING OVERVIEW:
[PartitioningPass] Initial exec time was 1.81e-07 s, new is 1.06e-07 -- Speedup = 1.71e+00

[LoopRecursionBreakPass] DONE
[LoopRecursionBreakPass] Analyzing loop 6 <-> 6
[LoopRecursionBreakPass] DONE
[PartitionWriterPass] Generating partitioned code

PartitionWriterPass::runOnModule() -- Original module's functions:
  odd with BBs:
    entry --> CPU
  main with BBs:
    entry --> CPU
    3 --> CPU
    beforeHeader --> CPU
    8 --> CPU_SIMD
    9 --> CPU_SIMD
    13 --> CPU
    afterHeader --> CPU
  puts with BBs:
PartitionWriterPass::find_initiators() -- Inspecting function main()
  Trivial initiators:
    5
    11
  Entry block initiators:
  Nontrivial initiators:
    14
  Results:
    entry has initiator entry
    beforeHeader has initiator entry
    5 has initiator 5
    11 has initiator 11
    12 has initiator 11

[PartitionWriterPass] Module CPU.part generated
[PartitionWriterPass] Module CPU_SIMD.part generated
Partitioned hello.ir

Turning CFGs into callgraphs