Optimization Through Recomputation in the Polyhedral Model

By Mike Jongen, Luc Waeijen, Roel Jordans, Lech Jóźwiak, Henk Corporaal.
Contents

• Introduction
• Related work
• Optimizing Through Recompute
• Polyhedral modelling
• Experimental Results
• Conclusion and future work
Introduction
Introduction

• (Mobile) systems use more artificial neural networks
  – Artificial vision
  – Image processing
  – Speech recognition

• Large amount of data accesses

• Can be improved by code transformations
Current possibilities and extensions

• Tiling

• Fusion

• Distribution

• Recomputation/overlapped tiling
  – Allows for better parallelism
  – Reduces memory traffic
This paper

• An example CNN application which includes recompute

• Extension of Polly

• Demonstration of the effectiveness of recomputation
Related Work
Automated polyhedral optimization frameworks

- Greatly reduce the effort of translating the original network description into an optimized form
- Automatically verifying the validity
- Different options: Polly, R-Stream-TF, and PPCG
- None of these frameworks provides a method of including recomputation in the optimization space
Why do we use Polly

- Uses the Polyhedral model for optimizations
- Direct integration with the LLVM compiler framework
- Adjustable
  - Add extra functionality
  - User defined schedules
  - Automate the process
Optimizing Through Recompute
System Architecture

- Processor
- Local Memory
- Global Memory
Educational Example
Inter Tile Reuse

Stored Part of the intermediate image
Inter Tile Reuse

Stored Part of the intermediate image
Inter Tile Reuse
Inter Tile Reuse
Other Dimensions
Methods to handle overlap

- Store the overlap globally
- Store the overlap locally
- Recompute the overlap
Global Method

- Pixels are stored externally
- Small buffer size
- Expensive memory accesses
Local Method

- Pixels are stored locally
- Larger buffers required
- Cheaper accesses
Recomputation Method

- Recomputes the pixels
- No extra memory required
- No extra accesses required
- More computations are required
Recomputation Tradeoffs
Recomputation Tradeoffs

Storing the overlap
Recomputation Tradeoffs

Storing the overlap
Recomputation Tradeoffs

Storing the overlap
Recomputation Tradeoffs

Storing the overlap
Recomputation Tradeoffs

Storing the overlap
Recomputation Tradeoffs

Recomputing the overlap
Recomputation Tradeoffs

Recomputing the overlap
Recomputation Tradeoffs

Recomputing the overlap
Recomputation Tradeoffs

Recomputing the overlap
Recomputation Tradeoffs

Recomputing the overlap
Recomputation Tradeoffs

Storing the overlap

Recomputing the overlap
Polyhedral Modeling
The Polyhedral Model and Recomputation

- Execution order is defined by the schedule

- Schedule is singular valued
  - One execution time per statement
  - One statement per execution time

- Recomputation:
  - Statements are executed multiple times
  - Non-singular valued schedules are required
Including Recomputation

- Support for non-singular valued schedules
- Transforming non-singular valued schedules to singular valued schedules
Example

Stmt[0] -> [0, 0]
Stmt[1] -> [0, 1], [1, 0]
Stmt[2] -> [1, 1]
Example

Old Schedule

Stmt[0] [0, 0]
Stmt[1] [0, 1]
Stmt[2] [1, 0]

Data: OriginalSchedule
Result: NewSchedule
set Lexmin to the lexicographical minimum of OriginalSchedule;
if OriginalSchedule is equal to Lexmin then
  set newSchedule to OriginalSchedule;
else
  add a dimension to newSchedule;
  set i to 0;
  RestOfSchedule = OriginalSchedule - Lexmin;
  while RestOfSchedule is not empty do
    add Lexmin to newSchedule with the new dimension set to i;
    i++;
    set Lexmin to the lexicographical minimum of RestOfSchedule;
    RestofSchedule = RestofSchedule - Lexmin;
  end
  add Lexmin to newSchedule with the new dimension set to i;
end
**Example**

**Old Schedule**

- **Stmt[0]** → [0, 0]
- **Stmt[1]** → [0, 1]
- **Stmt[2]** → [1, 0]

**Lexicographical Minimum**

- **Stmt[0]** → [0, 0]
- **Stmt[1]** → [0, 1]
- **Stmt[2]** → [1, 1]

**Data:** OriginalSchedule

**Result:** NewSchedule

- set Lexmin to the lexicographical minimum of OriginalSchedule;
- if OriginalSchedule is equal to Lexmin then
  - set newSchedule to OriginalSchedule;
- else
  - add a dimension to newSchedule;
  - set i to 0;
  - RestofSchedule = OriginalSchedule - Lexmin;
  - while RestofSchedule is not empty do
    - add Lexmin to newSchedule with the new dimension set to i;
    - i++;
    - set Lexmin to the lexicographical minimum of RestOfSchedule;
    - RestofSchedule = RestofSchedule - Lexmin;
  - end
- add Lexmin to newSchedule with the new dimension set to i;
end
Example

Data: OriginalSchedule

Result: NewSchedule

set Lexmin to the lexicographical minimum of OriginalSchedule;

\[
\text{if } \text{OriginalSchedule is equal to Lexmin then}
\]

| set newSchedule to OriginalSchedule; |

\[
\text{else}
\]

\[
\text{add a dimension to newSchedule;}
\]

\[
\text{set i to 0;}
\]

\[
\text{RestofSchedule = OriginalSchedule - Lexmin;}
\]

\[
\text{while RestofSchedule is not empty do}
\]

\[
\text{add Lexmin to newSchedule with the new dimension set to i;}
\]

\[
i++;
\]

\[
\text{set Lexmin to the lexicographical minimum of RestOfSchedule;}
\]

\[
\text{RestofSchedule = RestofSchedule - Lexmin;}
\]

\[
\text{end}
\]

\[
\text{add Lexmin to newSchedule with the new dimension set to i;}
\]

\[
\text{end}
\]
Example

Old Schedule

Data: OriginalSchedule
Result: NewSchedule
set Lexmin to the lexicographical minimum of OriginalSchedule;
if OriginalSchedule is equal to Lexmin then
set newSchedule to OriginalSchedule;
else
add a dimension to newSchedule;
set i to 0;
RestofSchedule = OriginalSchedule - Lexmin;
while RestofSchedule is not empty do
add Lexmin to newSchedule with the new dimension set to i;
i++;
set Lexmin to the lexicographical minimum of RestofSchedule;
RestofSchedule = RestofSchedule - Lexmin;
end
add Lexmin to newSchedule with the new dimension set to i;
end

Lexicographical Minimum
Example

Rest of Schedule

Stmt[1] → [1, 0]

Lexicographical Minimum

Stmt[0] → [0, 0]
Stmt[1] → [0, 1]
Stmt[2] → [1, 1]

Data: OriginalSchedule
Result: NewSchedule
set Lexmin to the lexicographical minimum of OriginalSchedule;
if OriginalSchedule is equal to Lexmin then
   set newSchedule to OriginalSchedule;
else
   add a dimension to newSchedule;
   set i to 0;
   RestofSchedule = OriginalSchedule - Lexmin;
while RestofSchedule is not empty do
   add Lexmin to newSchedule with the new dimension set to i;
   i++;
   set Lexmin to the lexicographical minimum of RestOfSchedule;
   RestofSchedule = RestofSchedule - Lexmin;
end
add Lexmin to newSchedule with the new dimension set to i;
end
Example

**Rest of Schedule**

- Stmt[1] \[1, 0\]

**New Schedule**

- Stmt[0, 0] \[0, 0\]
- Stmt[1, 0] \[0, 1\]
- Stmt[2, 0] \[1, 1\]

**Data:** OriginalSchedule
**Result:** NewSchedule
set Lexmin to the lexicographical minimum of OriginalSchedule;

if OriginalSchedule is equal to Lexmin then
    set newSchedule to OriginalSchedule;
else
    add a dimension to newSchedule;
    set i to 0;
    RestofSchedule = OriginalSchedule - Lexmin;
    while RestofSchedule is not empty do
        add Lexmin to newSchedule with the new dimension set to i;
        i++;
        set Lexmin to the lexicographical minimum of RestOfSchedule;
        RestofSchedule = RestofSchedule - Lexmin;
    end
    add Lexmin to newSchedule with the new dimension set to i;
end
Example

Lexicographical Minimum

Data: OriginalSchedule
Result: NewSchedule
set Lexmin to the lexicographical minimum of OriginalSchedule;

if OriginalSchedule is equal to Lexmin then
  set newSchedule to OriginalSchedule;
else
  add a dimension to newSchedule;
  set i to 0;
  RestofSchedule = OriginalSchedule - Lexmin;
  while RestofSchedule is not empty do
    add Lexmin to newSchedule with the new dimension set to i;
    i++;
    set Lexmin to the lexicographical minimum of RestofSchedule;
    RestofSchedule = RestofSchedule - Lexmin;
  end
  add Lexmin to newSchedule with the new dimension set to i;
end

New Schedule

Stmt[0, 0] → [0, 0]
Stmt[1, 0] → [0, 1]
Stmt[2, 0] → [1, 1]
Example

Lexicographical Minimum

Stmt[0, 0] → [0, 0]
Stmt[1, 0] → [0, 1]
Stmt[2, 0] → [1, 1]

New Schedule

Data: OriginalSchedule
Result: NewSchedule
set Lexmin to the lexicographical minimum of OriginalSchedule;
if OriginalSchedule is equal to Lexmin then
  set newSchedule to OriginalSchedule;
else
  add a dimension to newSchedule;
  set i to 0;
  RestofSchedule = OriginalSchedule - Lexmin;
  while RestofSchedule is not empty do
    add Lexmin to newSchedule with the new dimension set to i;
    i++;
    set Lexmin to the lexicographical minimum of RestOfSchedule;
  end
  RestofSchedule = RestofSchedule - Lexmin;
end
add Lexmin to newSchedule with the new dimension set to i;
end
Example

Lexicographical Minimum

Stmt[1] → [1, 0]

New Schedule

Stmt[0, 0] → [0, 0]
Stmt[1, 0] → [0, 1]
Stmt[2, 0] → [1, 1]

Data: OriginalSchedule
Result: NewSchedule
set Lexmin to the lexicographical minimum of OriginalSchedule;
if OriginalSchedule is equal to Lexmin then
  set newSchedule to OriginalSchedule;
else
  add a dimension to newSchedule;
  set i to 0;
  RestofSchedule = OriginalSchedule - Lexmin;
  while RestofSchedule is not empty do
    add Lexmin to newSchedule with the new dimension set to i;
    i++;
    set Lexmin to the lexicographical minimum of RestOfSchedule;
    RestofSchedule = RestofSchedule - Lexmin;
  end
  add Lexmin to newSchedule with the new dimension set to i;
end


**Example**

**Lexicographical Minimum**

- **Stmt[1]**: [1, 0]

**New Schedule**

- **Stmt[0, 0]**: [0, 0]
- **Stmt[1, 0]**: [0, 1]
- **Stmt[1, 1]**: [1, 0]
- **Stmt[2, 0]**: [1, 1]

**Data:** OriginalSchedule

**Result:** NewSchedule

set Lexmin to the lexicographical minimum of OriginalSchedule;

if OriginalSchedule is equal to Lexmin then

set newSchedule to OriginalSchedule;

else

add a dimension to newSchedule;
set i to 0;
RestofSchedule = OriginalSchedule - Lexmin;
while RestofSchedule is not empty do

add Lexmin to newSchedule with the new dimension set to i;
i++;
set Lexmin to the lexicographical minimum of RestOfSchedule;
RestofSchedule = RestofSchedule - Lexmin;
end

add Lexmin to newSchedule with the new dimension set to i;
end
Including Recomputation: location

Transformations

SCoP Detection
LLVM IR

Polyhedral SCoP

Dependence Analysis

Code Generation
SIMD
OpenMP

LLVM IR

Export
Import

JSCoP | scoplib

Manual Optimization / External Optimizers (PoCC/PLuTo)
Jscop Implementation

Conv[i0,i1,i2,i3] → [i0,i1,i2,i3]
Conv[i0,i1,i2,i3] → [t0,i1,t1,i2,i3] :
0 <= t0 < no_tiles and
0 <= t1 < tilesize and
i0 = tilesize * t0 + t1
Jscop Implementation

Conv[i0,i1,i2,i3] →[t0,i1,t1,i2,i3] :
0 <= t0 < no_tiles and
0 <= t1 < tilesize + overlap and
i0 = tilesize * t0 + t1
Dependencies

Before

Statement 1

Statement 2

After

Statement 1

Copy of Statement 1

Statement 2

OR
Experimental Results
Results for different tile sizes
Results for different tile sizes
Results for different tile sizes and several kernel sizes
Conclusion and Future Work
Conclusion

- An example CNN application which includes recompute
- Extension of Polly
- Demonstration of the effectiveness of recomputation
Future Work

- Legality Checks
- Model of the effects
- More applications
And Finally...

- Questions?
- Remarks?
- Suggestions?