Nested Loops Fusion LLVM transformation pass for compile-time optimization of GPU programs

V.G. Bulavintsev
(v.g.bulavintsev@gmail.com)

Kazan Federal University

Kazan, PCT'2017
Modern GPUs are Single Instruction, Multiple Data (SIMD) devices.

- **SIMD architecture** features asymmetrical ratio of Control Units (CUs) to Processing Units (PUs).
- Group of computational threads executed by PUs controlled by a single CU in lockstep mode is called a «warp» (in NVIDIA“s terminology).
Divergent branches in SIMD kill performance

Values of $A, B$

Threads

PU stay idle waiting for instructions from the CU
Natural Loops in Control Flow Graphs

- Node \( A \) is said to dominate node \( B \) if every path from start node to \( B \) node goes through \( A \).

- If the edge“s head node dominates its tail node, the edge is a back edge.

- The natural loop of a back edge is the smallest set of nodes that includes the head and tail of the back edge, and has no predecessors outside the set, except for the predecessors of the header.

### Natural loops in example graph:

<table>
<thead>
<tr>
<th>7 → 4</th>
<th>{4, 5, 6, 7}</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 → 3</td>
<td>{3, 4, 5, 6, 7, 8}</td>
</tr>
<tr>
<td>9 → 1</td>
<td>{1, 2, 3, 4, 5, 6, 7, 8, 9}</td>
</tr>
</tbody>
</table>
Disjoint and Nested Natural Loops

- **Disjoint natural loops**
- **Nested natural loops**

Diagram:
- **Header**
- **Exit block**
- **Latch**
- **Backedge**

a) disjoint natural loops
b) nested natural loops
Reducible and irreducible CFGs

- Natural loop could be replaced by a single basic block.
- A graph is said to be **reducible**, if it can be reduced to a single basic block by iteratively replacing natural loops with basic blocks.
- Any irreducible graph could be transformed into reducible one by splitting some of it“s nodes.
- Natural loops of a reducible CFG form a tree hierarchy.
- Structured programming always produces reducible CFGs.
- Further we will speak only of reducible CFGs.
The main idea of the transformation is to «reuse» the outer loop’s backedge to «emulate» the inner loop’s backedge.

This transformation effectively creates an additional «convergence point» for SIMD threads.

Semantically the program remains the same.
(a) Original CFG.

(b) Split $A_{hw}$ and add new latch block $A_l$.

(c) Add conditional bypass for $A_w$ and $A_e$. Transformation complete.
Transformation effect on code execution

Original program

Transformed program

Threads:

<table>
<thead>
<tr>
<th>#0</th>
<th>#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

PUs stay idle for 7 execution blocks in total.

PUs stay idle for 1 execution block.
Speed-up condition

\[ \frac{n}{N} > \frac{T_a}{T_a + T_b} . \]

\( T_a \) – the time to execute the \( A_w \) basic block;

\( T_b \) – the time to execute \( B \)'s instructions;

\( N \) – the total number of threads/PUs in the SIMD group (\textit{warp size});

\( n \) – the number of idle threads in the SIMD group at the moment;

\( (N - n)T_a \) – “the cost” of going to the outer loop to get new work for idle threads;

\( nT_b \) – the potential payoff from getting new work for idle threads.
Low Level Virtual Machine (LLVM)

- LLVM was developed in 2000-2003 at University of Illinois to study the techniques of dynamic compilation.
- Initially the project consisted of the Low Level Virtual Machine that interpreted the special Intermediate Representation (IR) code.
- The project gained popularity and accumulated a large collection of various compilation-related tools: assemblers, compilers, parsers, etc.
- At the moment the project features modularized structure, rich API, and lots of libraries and utilities. Compilers for popular programming languages and assemblers for many hardware platforms are available, including Nvidia CUDA and PTX Assembly.
LLVM Transformation Pass

- LLVM compilation procedure allows users to apply optional transformation procedures called «passes», that work on IR.
- This procedures provide code optimization and analysis facilities.
- LLVM's rich API allows one to write his or her own pass.
- Passes can be chained into one another.
- Passes can be applied to IR out of compilation process with the use of opt utility.
Applying Loop Fusion transformation pass during the Clang CUDA compilation process

Source code .cu

Clang CUDA frontend

Host IR

Device IR

Here we apply our transformation with opt utility

Loop fusion transformation

Host code generator

Host compiler

Exec. binary

NVPTX codegen

PTX assembly
Test application design

- Test application consists of two nested loops.
- Exit from a loop is triggered by the specific output of its associated Pseudo Random Number Generator (PRNG).
- PRNGs have controllable probability of triggering the loop's exit conditions.
- Initially PRNGs are seed from the thread number.
- «Useful work» for the loops is counting the number of loops iterations.
Original CFG

Transformed CFG
Application execution time of original and transformed application.

Application execution time of original and transformed application relative to the execution time of the original application for k=0.

- **Original applications (all a/b ratios)**
- **Transformed application, a/b = 10/1**
- **Transformed application, a/b = 1/1**
- **Transformed application, a/b = 1/2**
- **Transformed application, a/b = 1/10**

Value of parameter k (number of threads in the SIMD group receiving "early exit" events)

Up to 18x times
Discussion

- Nested Loops Fusion transformation increases performance of synthetic test application on GPU up to 18x times, depending on the dataset.
- Transformation decreases performance of test application by 30% for the datasets that do not provoke «the SIMD effect».
- The procedure can be applied automatically during the compilation process.
- The procedure is a general one and can be applied to any program that could be represented in IR form.
- The procedure should be applied sparingly, as a part of profiling-driven optimization process.
- Future plans include pushing the transformation pass into LLVM mainline.
Thank you for your attention!