

Nov 12th 2020 / LLVM-HPC 2020

Towards Kernel Fusion for the Optimisation of Scientific Applications

Andrew Lamzed-Short, Gihan Mudalige, University of Warwick

Tim Law, AWE

Andrew Mallinson, Intel

Stephen Jarvis, University of Birmingham

© British Crown Owned Copyright 2020/AWE. Published with permission of the Controller of Her Britannic Majesty's Stationery Office. This document is of United Kingdom origin and contains proprietary information which is the property of the Secretary of State for Defence. It is furnished in confidence and may not be copied, used or disclosed in whole or in part without prior written consent of Defence Intellectual Property Rights DGDCDIPR-PL — Ministry of Defence, Abbey Wood, Bristol, BS34 8JH, England.



Agenda

- Motivation
- Implementation
- Experimental Setup and Results
- Conclusion and Future Work



Motivation

- Modular functions used in large codebases, teams
- Function calls incur a performance cost at runtime
 - Loading data in and out of main memory and cache incurs additional performance penalties



Motivation

Idea

- Intermediate stage during compilation to fuse these kernel functions together
- Maintains code modularity; recovers runtime performance



Input Program Constraints

Kernel restrictions

- Called exactly once per main loop iteration
- Returns void, inputs and outputs defined in function parameters
- Function body consists only of a loop no preceding or succeeding statements





Loop restrictions

- Exactly one level deep no nested loops
- Stride of 1; increasing from a constant value
- When two loop iteration spaces between kernels differ, they differ by at most 1 iteration*



Example



Consistent Input

O1 optimisation level required

Other passes required for consistency

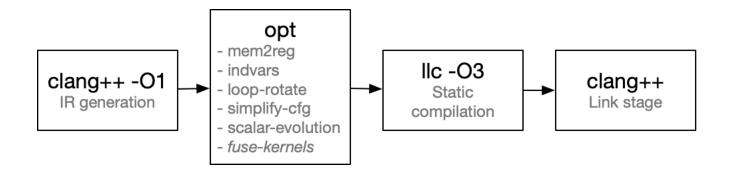
- indvar
- simplifycfg
- mem2reg

- scalar-evolution
- loop-rotate

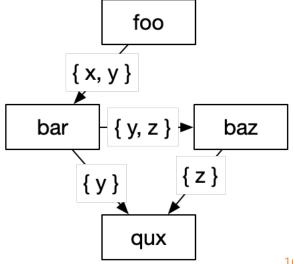
WARWICK



Compilation Pipeline



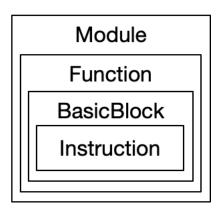
- Directed acyclic graph (DAG): kernels as vertices, dependencies as edges
- Traversed using topological sort to resolve inter-kernel dependencies





Implemented as a ModulePass

- Require whole module intermediate representation (IR)
 - Read, fuse, and remove all kernel functions
 - Write a new fused kernel function, replace calls

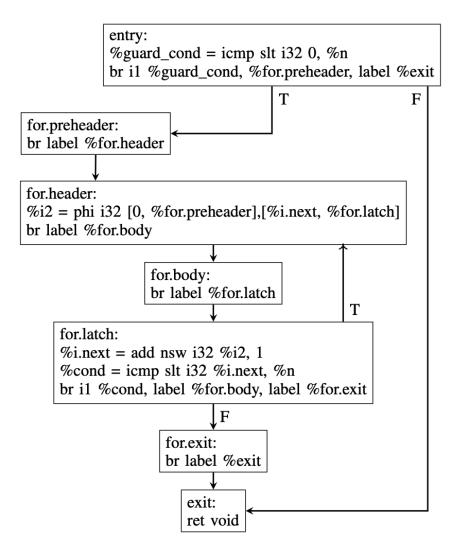


WAR\//



- Additional function declarations included and inlined into IR during compilation
- Source code decoration introduced: [[clang::fuse_kernel]]

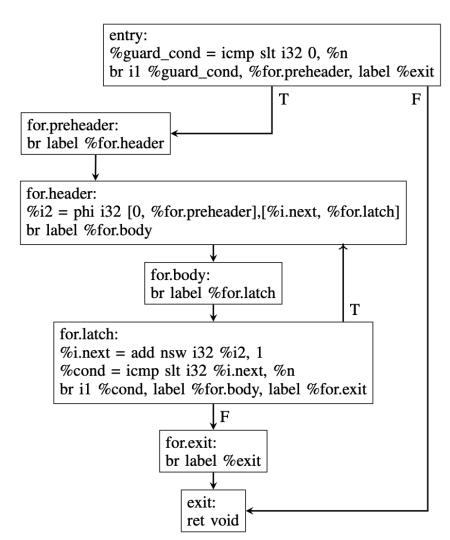
WARWICK



Canonical Loop Structure in LLVM IR

- BasicBlock structure of a Function with a single Loop within
- Single Static Assignment form





Fusion Algorithm

- Create a skeleton function, new induction variable
- Append for.body's, replace references
- Merge header blocks into set





Override runOnModule(Module& m) function

- Filter list of kernel functions
- Create DAG
- Create fused kernel, fusing each function and its loop
- Remove kernels, emit fused kernel to IR



Experimental Setup

- Intel[®] Xeon[®] Gold 6252 Processor (formerly codenamed "Cascade Lake")
 - 24 physical cores, 2 threads per core; 2.10GHz clock speed
 - 6 channel memory bandwidth, max. 131.13GB/s
 - L2 cache: 24,576 KiB; L3 cache: 36,608KiB
 - Vectorisation: Intel[®] Advanced Vector Extensions 512 (Intel[®] AVX-512)



Experimental Setup

- L3 cache can support an array of length $\frac{36608000}{8} = 4756000$
- Stream benchmark recommends 4x this to properly measure cache bandwidth, but will test wide range of problem sizes upto and including
- Proportional amount of iterations for equal amount of work



Experimental Setup

Problem Size	1000	2000	4000	8000	16000	32000	64000	128000	256000
Iterations	13107200	6553600	3276800	1638400	819200	409600	204800	102400	51200
Problem Size	512000	1024000	2048000	4096000	8192000	16384000	32768000	65536000	
Iterations	25600	12800	6400	3200	1600	800	400	200	



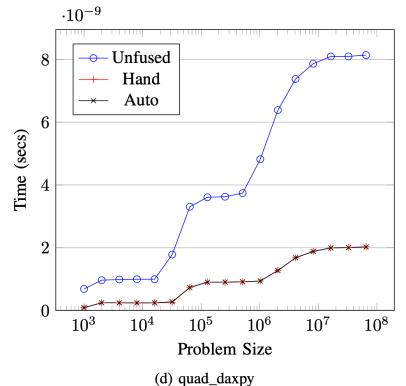
Results

- 1.5 to 4x speedup over unfused program
 - Factor determined by program's runtime characteristics
- Performance parity with hand-fused version of same program



Memory-Bound Problems

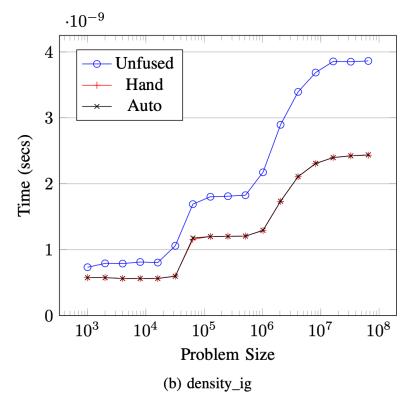
- Quintessential use-case for fusion techniques
- 4x runtime speedup
- Predominantly memory-bound, low arithmetic complexity
 - 4 lots of daxpy
- See where problem falls out of L2 and L3 caches



WARWICK THE UNIVERSITY OF WARWICK

Mixed Boundedness Problems

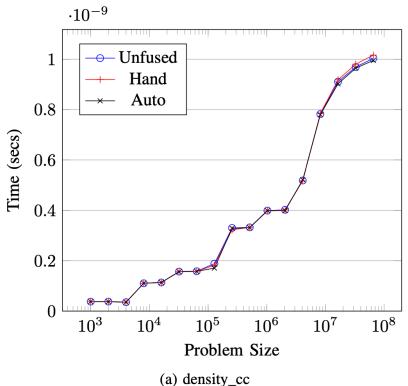
- Performance still to be gained with problems with increased compute-boundedness and arithmetic complexity
 - Factor is smaller
- Same performance cliffs as before





Compute-Bound Problems

 Identical performance for problems whose performance is dictated primarily by the complexity of their computations





Conclusion

- We are developing an LLVM transformation pass to optimise the runtime performance of kernelled scientific applications
- Present an upto 4x speedup by improving data locality and cache residency



Future Work

- Build up the complexity to support real-world applications and benchmarks
- Upstream integration into the LLVM repository and/or SYCL-based compilers (such as Intel's oneAPI DPC++ compiler)