Exposing parallelism in sequential code using a modern commutativity analysis

Chris Vasiladiotis

THE UNIVERSITY
of EDINBURGH

UK Systems Research Challenges Workshop 2020

December 2020

Outline

1 Introduction

2 Dynamic Commutativity Analysis

3 Future Directions

4 Conclusion

Introduction – Motivation

```
1 for (i = 0; i < N; ++i)
2 {
3   array[i]++;
4
5 }
```

If permuted execution produces the same result,

can we parallelize?

Introduction – Motivation

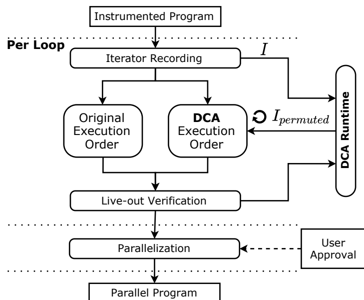
```c
1 for (i = 0; i < N; ++i)
2 {
3   array[i]++;
4
5 }
```

```c
1 while (ptr)
2 {
3   ptr->val++;
4   ptr = ptr->next;
5 }
```

If permuted execution produces the same result,

can we parallelize?

Dynamic Commutativity Analysis (DCA)




```
1 for (i = 0; i < N; ++i)
2 {
3   array[i]++;
4
5 }
```

To appear in:

International Symposium on Code Generation and Optimization (CGO), March 2021

Loop Parallelization using Dynamic Commutativity Analysis

Christos Vasiladiotis
University of Edinburgh
United Kingdom
c.vasiladiotis@sms.ed.ac.uk

Roberto Castañeda Lozano
University of Edinburgh
United Kingdom
rcastae@inf.ed.ac.uk

Murray Cole
University of Edinburgh
United Kingdom
mic@inf.ed.ac.uk

Björn Franke
University of Edinburgh
United Kingdom
bfranke@inf.ed.ac.uk

Abstract

Automatic parallelization has largely failed to keep its promise of extracting parallelism from sequential legacy code to max-

1 Introduction

Compilers for automatic parallelization have failed to deliver on their promise to seamlessly transition sequential legacy

Structure of evaluation

Evaluation dimensions:

- Detection performance
- Precision of detection
- Speedup performance

Comparison against baseline tools:

- 2 Dynamic techniques
- 3 Static techniques

Comparison with 2 sets of input code:

- NAS Parallel Benchmark (NPB) suite
- Collection of Pointer-Linked Data Structure (PLDS) loops

Evaluation – Detection vs Dynamic Methods

- Dependence Profiling [TWFO09]
- DiscoPoP [LAH+16]

Benchmark	Loops (#)	Dependence Profiling (#)	DiscoPoP (#)	DCA (this work) (#)
BT	182	168	176	168
CG	47	33	21	33
DC	105	—	—	41
EP	9	6	8	6
FT	42	36	34	36
IS	16	12	20	12
LU	186	160	164	160
MG	81	48	66	48
SP	250	233	231	233
UA	479	—	—	466
Total	**1397**	**696**	**720**	**1203**

Evaluation – Detection vs Static Methods

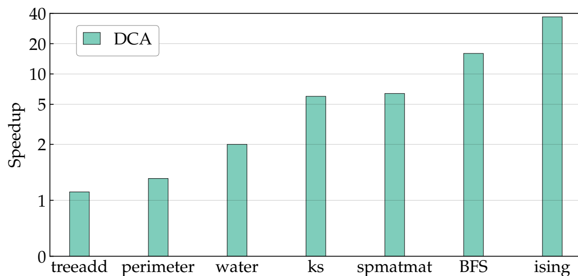
- IDIOMS [GO17]
- POLLY [GGL12]
- ICC [Inc20]

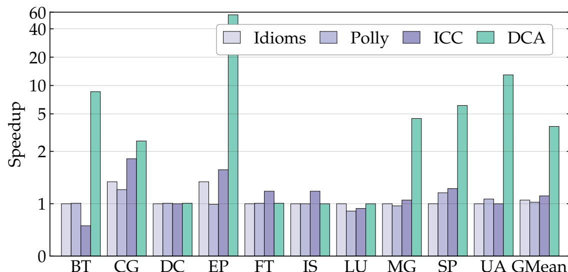
Benchmark	Loops	Combined Static		DCA (this work)	
	(#)	(#)	(%)	(#)	(%)
BT	182	80	44	168	92
CG	47	25	53	33	70
DC	105	39	37	41	39
EP	9	4	44	6	67
FT	42	8	19	36	86
IS	16	11	69	12	75
LU	186	90	48	160	86
MG	81	32	40	48	59
SP	250	113	45	233	93
UA	479	209	44	466	97
Total	**1397**	**611**	**44**	**1203**	**86**

Evaluation – Detection vs Dynamic Methods

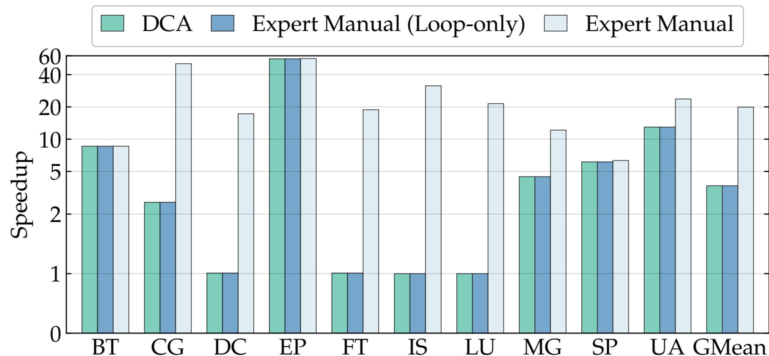
Benchmark Loop	Profitability			Detection Technique
	Sequential Coverage (%)	Potential Speedup (×)		Expert Manual
		Loop	Overall	
429.mcf	30	2.2	—	DSWP variant 1 [RVVA04, Ran07]
300.twolf	30	1.5	—	DSWP variant 2 [ROR+08]
ks	99	1.5	—	DSWP variant 1
otter	15	2.5	—	DSWP variant 2
em3d	100	∼ 2	—	DSWP variant 1
mst	100	1.5	—	DSWP variant 1
bh	100	2.75	—	DSWP variant 1
perimeter	100	2.25	—	DSWP variant 1
treeadd	100	—	∼ 7	Partitioning [FLG12]
hash	50	—	∼ 4	Partitioning
BFS	99	—	21	Galois [PNK+11]
ising	95	—	∼ 6	ASC [KWF+18]
spmatmat	89	—	∼ 4	APOLLO [CSB+17]
water-spatial	63	—	2	OpenMP

Evaluation – Speedup vs Baseline





Evaluation – Speedup vs Expert



Potential Future Directions

- Move towards mapping of higher level parallel constructs.
- What would it take to improve the answers to commutativity queries?

Direction 1

Move towards mapping of higher level parallel constructs.

- Maps, reductions and fusions of them.
- Stencils, scans, pipelines.
- Parallel constructs that orchestrate computation.
- Issues with their composition and exploitation.

Direction 2

What would it take to improve the answers to commutativity queries?

- Symbolic execution (e.g., assumptions in KLEE)
- Fuzz testing
- Synthesis

Conclusion

Main points of Dynamic Commutativity Analysis (DCA):

- a technique for testing the commutativity and hence potential parallelizability of arbitrarily complex loops
- effective in detecting parallelizable loops
- discovers loops with significant parallelization profitability and high precision

Thank you

Questions?

Evaluation – Coverage and precision

DCA execution time and precision detection.

Bmk	Loops	DCA (this work)				Combined Static
	(#)	Found (#)	False Positive (#)	False Negative (#)	Sequential Coverage (%)	Sequential Coverage (%)
BT	182	168	0	0	100	36
CG	47	33	0	0	91	7
DC	105	41	0	0	0	0
EP	9	6	0	0	100	37
FT	41	36	0	0	91	42
IS	16	12	0	0	60	56
LU	186	160	0	0	84	56
MG	81	48	0	0	87	56
SP	250	233	0	0	94	77
UA	479	466	0	0	86	57

References I

Juan Manuel Martinez Caamaño, Aravind Sukumaran-Rajam, Artiom Baloian, Manuel Selva, and Philippe Clauss, *APOLLO: Automatic speculative POLyhedral Loop Optimizer*, IMPACT 2017 - 7th International Workshop on Polyhedral Compilation Techniques, January 2017, p. 8.

Min Feng, Changhui Lin, and Rajiv Gupta, *PLDS: Partitioning linked data structures for parallelism*, ACM Trans. Archit. Code Optim. **8** (2012), no. 4, 38:1–38:21.

Tobias Grosser, Armin Groesslinger, and Christian Lengauer, *Polly — Performing Polyhedral Optimizations on a Low-Level Intermediate Representation*, Parallel Process. Lett. **22** (2012), no. 04, 1250010.

Philip Ginsbach and Michael F. P. O'Boyle, *Discovery and exploitation of general reductions: A constraint based approach*, Proceedings of the 2017 International Symposium on Code Generation and Optimization (Piscataway, NJ, USA), CGO '17, IEEE Press, 2017, pp. 269–280.

Intel Inc., *Intel® C++ Compiler*, https://software.intel.com/en-us/c-compilers, August 2020.

Peter Kraft, Amos Waterland, Daniel Y. Fu, Anitha Gollamudi, Shai Szulanski, and Margo Seltzer, *Automatic parallelization of sequential programs*, ArXiv180907684 Cs (2018).

References II

Zhen Li, Rohit Atre, Zia Huda, Ali Jannesari, and Felix Wolf, *Unveiling parallelization opportunities in sequential programs*, J. Syst. Softw. **117** (2016), no. C, 282–295.

Keshav Pingali, Donald Nguyen, Milind Kulkarni, Martin Burtscher, M. Amber Hassaan, Rashid Kaleem, Tsung-Hsien Lee, Andrew Lenharth, Roman Manevich, Mario Méndez-Lojo, Dimitrios Prountzos, and Xin Sui, *The tao of parallelism in algorithms*, SIGPLAN Not. **46** (2011), no. 6, 12–25.

Ram Rangan, *Pipelined multithreading transformations and support mechanisms*, PhD Thesis, Princeton University, USA, 2007.

Easwaran Raman, Guilherme Ottoni, Arun Raman, Matthew J. Bridges, and David I. August, *Parallel-stage decoupled software pipelining*, Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization (Boston, MA, USA), CGO '08, Association for Computing Machinery, April 2008, pp. 114–123.

Ram Rangan, Neil Vachharajani, Manish Vachharajani, and David I. August, *Decoupled Software Pipelining with the Synchronization Array*, Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (Washington, DC, USA), PACT '04, IEEE Computer Society, 2004, pp. 177–188.

References III

Georgios Tournavitis, Zheng Wang, Björn Franke, and Michael F.P. O'Boyle, *Towards a holistic approach to auto-parallelization: Integrating profile-driven parallelism detection and machine-learning based mapping*, ACM Press, 2009, p. 177.