ICS'22 Home

mstrout@cs.arizona.edu Profile · Help · Sign out

Rebuttal response deadline in 69 hours

#173 Data Layout Optimization in RAJA

Main Edit Your submissions #514 > (All) Search

Email notification
Select to receive email on updates to reviews and comments.

▼ PC conflicts

Bronis de Supinski
Catherine Olschanowsky
Christian Engelmann
Ignacio Laguna
Martin Schulz
Michelle Strout
Pavlos Petoumenos
Stephanie Brink
Suren Byna

Submitted

Submission (671kB) 4 Feb 2022 4:44pm AoE · 391eac3f

▶ Abstract

The layout of data in memory is a key

▶ Authors (blind)

B. Neth, T. Scogland, B. de Supinski, M. Strout [details]

[more]

	OveMer	RevExp
Review #173A	2	3
Review #173B	2	3
Review #173C	2	2

You are an **author** of this submission.

Edit submission Add Rebuttal response

Reviews in plain text

Review #173A

Overall merit

2. Weak reject

Reviewer expertise

3. Knowledgeable

Paper summary

This paper introduces an API in RAJA to change data layouts between computations. In addition. a performance model that takes into account the data order and the loop bounds is introduced to determine the best layouts.

Weaknesses

The approach is evaluated on very simple benchmarks from the Polybench benchmark suite.

Strengths

The paper proposes an API and performance model aided by benchmarks to determine if changing the data layout will be beneficial for the performance of the application.

Comments for author

1) It is difficult to assess the benefit of this approach for real applications, as the evaluation is done on very simple benchmarks. It is unclear if the approach will work for real applications with more complex access patterns.
2) If I understand correctly, the benchmarks are executed (at least symbolically) to determine the best layout. I wonder if the execution of the benchmark is taken into account in the overall running times of the application or if this time is not considered at all in the times reported.
3) I was also wondering if changing the layout is also the best option. Imagine a double nested loop that is accessing all the 2D arrays in column major order. The proposed approach will change the layout of all the arrays, but in practice the best optimization would be to simply exchange the loops and avoid changing the data layout in memory.
3) The benefit of changing the data layout depends on the computation that is being performed, the access pattern, the hardware prefetch, whether the data fits in the cache, ... There is no discussion about this in the paper, but it is actually very important for the overall performance application, because if the data is already in cache, changing the layout will require data copies and will make the code more inefficient.

Review #173B

Overall merit

2. Weak reject

Reviewer expertise

3. Knowledgeable

Paper summary

The authors introduce support in RAJA allowing programmers to trigger runtime (intra-execution) layout transformations with minimal programmer effort. They complement their API with (i) a performance model that dynamically estimates performance of different layouts and (ii) runtime microbenchmarking to provide input to the model. Therefore, programmers can even skip

specifying layout transformations, letting the system evaluate and identify appropriate ones on its own.

Weaknesses

- Quite a few limitations
- Weak experimental evaluation
- Essentially work in progress

Strengths

- Improves performance (yet not impressively in the selected benchmarks)
- Requires limited programmer intervention, assuming however RAJA (and its variants are already used).

Comments for author

The idea of facilitating data layout optimizations at execution time is certainly interesting and worth exploring. The example in Fig. 1, albeit an extreme case (in the "bad" case the layout is the opposite of what the typical programmer would opt for) motivates the approach.

I had the question about the limitations of the approach throughout the paper, and the authors, to their credit, confirmed my concerns in conclusions. Those limitations (at least in the current status of the work) significantly reduce the applicability of the approach. Also, I was not totally convinced that the runtime microbenchmarks (and the respective cached results) can provide a representative estimation of the performance of different layouts at significantly different data footprints.

The experimental evaluation is also lacking more complex benchmarks. One question on the evaluation is the optimization options selected for the compilation. Moreover, it was not clear to me why the limited performance difference among versions that perform layout transformation is an indication of the low overhead of your approach (your approach could potentially be identifying better layouts, yet at higher overhead outweighing the benefits). The performance improvement in the benchmarks that are amenable to layout optimizations is rather limited, however, this is expected due to the small size and the simplicity of the benchmarks.

In summary, this is a promising approach, however, in its current state, it is a work-in-progress.

Review #173C

Overall merit

2. Weak reject

Reviewer expertise

2. Some familiarity

Paper summary

This paper an extension to RAJA that enables data-layout transformation optimisations to be performed not just at data initialisation but also at point between loop-nests. The paper also offers a performance model, based on formulating an ILP problem for each array for each kernel, that accounts for the layout transposition overhead.

The paper concludes with experimental results designed to demonstrate the notation and performance modelling, and some analysis of the number of lines of code needed.

Weaknesses

- Limited in applicability due to restriction to perfect loop nests only, and compile-time-constant loop bounds (line 624).
- As a consequence of these restrictions, the experimental work is limited to rather few benchmark programs - just four, but only two of these are actually plausible candidates as the others have no re-use (line 653).
- The performance differences achieved (Fig 4) are actually rather small - which is disappointing as in principle data layout transformation could result in quite big improvements in at least some examples.
- The performance modelling work is nice - but it's nice because it dodges the hard problem, of non-constant loop bounds.
- Very limited discussion of related work.

Strengths

- The paper is generally well written, the explanation of the cost model and constraint system is clear. The contributions claimed are important and justified by the work.
- The proposed annotation scheme seems fairly elegant and should indeed help reduce the source code complexity consequences of dynamic layout transposition.

Comments for author

The lack of a related works section makes it difficult to understand how novel or new the ideas presented and implemented are; changing data-layout mid computation appears to be a key differentiator of this work but it is not well contextualised by previous works that aim to do the same thing in perhaps different contexts/frameworks or made clear that this is a novel contribution that the authors claim.

Fig 1 lacks any details of hardware or problem size.

Listing 3 is confusing to read - it is unclear what 'conversion_variable' is? is it conv_{i,o,t} as in the proes? how is estimated_cost() calculated? and what is the implicit conversion to something numeric to multiply it by coefficient? Also 'for access to view in accesses' should it be 'for access to view in view_accesses'.

line 373 - 'Conversion-Format matching' should be 'Format-Conversion matching'?

line 495 - missing full stop after knl2

Fig 2 - the blue and red underlines could be explained more clearly even something as simple as: 'The red underlines and blue underlines each highlight an example of....'

I found the explanation of the access order a bit difficult because you switch from the B in knl2 example to B in knl1 to make your performance impact point and these are just very similar names

Are vectorization optimizations ignored entirely by your system? These exist in RAJA as loop schedules as you state but it is unclear if these are used in the performance benchmarking or if your cost model accounts for vectorization in deciding of effective layouts.

There is no evaluation of the effect on memory bottle-necks of different data-layouts. It would be useful to see how efficiently the memory bandwidth available is used with these data-layout changes so understand how bound by memory vs compute the benchmarks are.

Minor:

Line 618 please provide compiler flags.

Figure 3 on page 7 is missing its figure number, title and caption. It's also rather ugly how the (a) and (b) subcaptions almost collide.

Re: Figure 4: were these experiments repeated? How many times? (perhaps you might add a sentence around line 618). This is important as the performance differences we're looking at are rather small.

Summary:

With improved benchmarking and significant additions for better contextualization and addressing the interactions between their work and vectorization optimizations this could be a good paper. Overcomming the limitation to constant bounds seems critical for this work to be of practical value.