

Are we doing the right thing?

- A Critical Analysis of the Academic HPC Community

20th PDSEC Workshop

May 24th 2019 | Rio de Janeiro



Hartwig Anzt



Goran Flegar

Hartwig Anzt, Goran Flegar

**THE 20TH IEEE INTERNATIONAL WORKSHOP ON
PARALLEL AND DISTRIBUTED SCIENTIFIC AND ENGINEERING COMPUTING
(PDSEC '19)**

**MAY 24, 2019
HILTON RIO DE JANEIRO COPACABANA
RIO DE JANEIRO, BRAZIL**

The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)



A block-asynchronous relaxation method for graphics processing units

Hartwig Anzt^{a,*}, Stanimire Tomov^b, Jack Dongarra^{b,c,d}, Vincent Heuveline^a

^a Karlsruhe Institute of Technology, Germany

^b University of Tennessee Knoxville, USA

^c Oak Ridge National Laboratory, USA

^d University of Manchester, UK

HIGHLIGHTS

- Block-asynchronous relaxation on GPU-accelerated systems.
- Method's high iteration rate compensates a low convergence rate (competitive to CG).
- GPU thread-block scheduling reduces non-deterministic behavior and stochastic noise.
- Analysis of different communication strategies for multi-GPU usage.
- Block-asynchronous relaxation inherently tolerant to hardware error.

New Algorithm

GPU implementation

Performance results

The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

- + **New ideas** presented;
- + **Performance evaluations** presented;
- Performance evaluations are typically “**selective**”;
- Users need to **re-implement code**;
- Difficult if **few details** are provided;



A block-asynchronous relaxation method for graphics processing units

Hartwig Anzt^{a,*}, Stanimire Tomov^b, Jack Dongarra^{b,c,d}, Vincent Heuveline^a

^a Karlsruhe Institute of Technology, Germany

^b University of Tennessee Knoxville, USA

^c Oak Ridge National Laboratory, USA

^d University of Manchester, UK

HIGHLIGHTS

- Block-asynchronous relaxation on GPU-accelerated systems.
- Method's high iteration rate compensates a low convergence rate (competitive to CG).
- GPU thread-block scheduling reduces non-deterministic behavior and stochastic noise.
- Analysis of different communication strategies for multi-GPU usage.
- Block-asynchronous relaxation inherently tolerant to hardware error.

New Algorithm

GPU implementation

Performance results

The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

- + **New ideas** presented;
- + **Performance evaluations** presented;
- Performance evaluations are typically “**selective**”;
- Users need to **re-implement code**;
- Difficult if **few details** are provided;



A block-asynchronous relaxation method for graphics processing units

Hartwig Anzt^{a,*}, Stanimire Tomov^b, Jack Dongarra^{b,c,d}, Vincent Heuveline^a

^a Karlsruhe Institute of Technology, Germany

^b University of Tennessee Knoxville, USA

^c Oak Ridge National Laboratory, USA

^d University of Manchester, UK

HIGHLIGHTS

- Block-asynchronous relaxation on GPU-accelerated systems.
- Method's high iteration rate compensates a low convergence rate (competitive to CG).
- GPU thread-block scheduling reduces non-deterministic behavior and stochastic noise.
- Analysis of different communication strategies for multi-GPU usage.
- Block-asynchronous relaxation inherently tolerant to hardware error.

New Algorithm

GPU implementation

Performance results

Figure 1.5: Trends in scientific publications worldwide, 2008 and 2014

13.7%

Growth in publications with authors from Europe between 2008 and 2014, the region with the greatest share of publications: 39.3%

60.1%

Growth in publications with authors from Africa between 2008 and 2014

109.6%

Growth in publications with authors from Arab states between 2008 and 2014

UNESCO science report: towards 2030

The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

- + **New ideas** presented;
- + **Performance evaluations** presented;
- Performance evaluations are typically “**selective**”;
- Users need to **re-implement code**;
- Difficult if **few details** are provided;



In a **perfect world**, new algorithms, implementations & performance results are

- fully reproducible;
- publicly accessible;
- ready to be used by the community / domain scientists;
- integrated into community packages;



A block-asynchronous relaxation method for graphics processing units

Hartwig Anzt^{a,*}, Stanimire Tomov^b, Jack Dongarra^{b,c,d}, Vincent Heuveline^a

^a Karlsruhe Institute of Technology, Germany

^b University of Tennessee Knoxville, USA

^c Oak Ridge National Laboratory, USA

^d University of Manchester, UK

HIGHLIGHTS

- Block-asynchronous relaxation on GPU-accelerated systems.
- Method's high iteration rate compensates a low convergence rate (competitive to CG).
- GPU thread-block scheduling reduces non-deterministic behavior and stochastic noise.
- Analysis of different communication strategies for multi-GPU usage.
- Block-asynchronous relaxation inherently tolerant to hardware error.

New Algorithm

GPU implementation

Performance results

Figure 1.5: Trends in scientific publications worldwide, 2008 and 2014

13.7%

Growth in publications with authors from Europe between 2008 and 2014, the region with the greatest share of publications: 39,3%

60.1%

Growth in publications with authors from Africa between 2008 and 2014

109.6%

Growth in publications with authors from Arab states between 2008 and 2014

UNESCO science report: towards 2030

The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

- + **New ideas** presented;
- + **Performance evaluations** presented;
- Performance evaluations are typically “**selective**”;
- Users need to **re-implement code**;
- Difficult if **few details** are provided;



In a **perfect world**, new algorithms, implementations & performance results are

- **fully reproducible**;
- **publicly accessible**;
- **ready to be used** by the community / domain scientists;
- **integrated into community packages**;



Established community software packages

- are the **powertrain** behind many **scientific simulation codes**;
- often **fall short** in providing production-ready implementations of **novel algorithms**;
- often accept merge requests that **lack comprehensive documentation** and rigorous **performance assessment**;

The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

- + **New ideas** presented;
- + **Performance evaluations** presented;
- Performance evaluations are typically “**selective**”;
- Users need to **re-implement code**;
- Difficult if **few details** are provided;



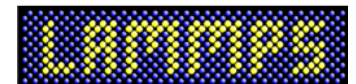
In a **perfect world**, new algorithms, implementations & performance results are

- **fully reproducible**;
- **publicly accessible**;
- **ready to be used** by the community / domain scientists;
- **integrated into community packages**;



Established community software packages

- are the **powertrain** behind many **scientific simulation codes**;
- often **fall short** in providing production-ready implementations of **novel algorithms**;
- often accept merge requests that **lack comprehensive documentation** and rigorous **performance assessment**;



The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

- + **New ideas** presented;
- + **Performance evaluations** presented;
- Performance evaluations are typically “**selective**”;
- Users need to **re-implement code**;
- Difficult if **few details** are provided;



In a **perfect world**, new algorithms, implementations & performance results are

- **fully reproducible**;
- **publicly accessible**;
- **ready to be used** by the community / domain scientists;
- **integrated into community packages**;



Established community software packages

- are the **powertrain** behind many **scientific simulation codes**;
- often **fall short** in providing production-ready implementations of **novel algorithms**;
- often accept merge requests that **lack comprehensive documentation** and rigorous **performance assessment**;

Why are we not changing the system?

- $\text{effort}(\text{Prototype Code}) \ll \text{effort}(\text{Production Code})$;
- The academic system **does not reward software development**;
- Promotion and appointability based on **scientific papers**;
- Sluggish acceptance of the “**scientific software engineer**”;
- Authors want to keep algorithms confidential;

The HPC Algorithm Landscape

The Typical Publication in HPC Conferences / Journals

- An article describing a **new algorithm / implementation** outperforming existing solutions.
- **Performance benchmarks** on high-end HPC resources (not even archived)
- **Internal prototype code** (not publicly accessible)

How does the community benefit from reading this?

- + **New ideas** presented;
- + **Performance evaluations** presented;
- Performance evaluations are typically “**selective**”;
- Users need to **re-implement code**;
- Difficult if **few details** are provided;



In a **perfect world**, new algorithms, implementations & performance results are

- **fully reproducible**;
- **publicly accessible**;
- **ready to be used** by the community / domain scientists;
- **integrated into community packages**;



Established community software packages

- are the **powertrain** behind many **scientific simulation codes**;
- often **fall short** in providing production-ready implementations of **novel algorithms**;
- often accept merge requests that **lack comprehensive documentation** and rigorous **performance assessment**;

Why are we not changing the system?

- $\text{effort}(\text{Prototype Code}) \ll \text{effort}(\text{Production Code})$;
- The academic system **does not reward software development**;
- Promotion and appointability based on **scientific papers**;
- Sluggish acceptance of the “**scientific software engineer**”;
- Authors want to keep algorithms confidential;

Extremely inefficient and unsatisfying!

Let's try to change the System!

- Scientific papers and the Hirsch-Index (H-Index) will remain the Gold Standard for appointability.
- **We can move the publication system towards crediting software contributions!**
 - Reproducibility initiatives (ACM Journals)
 - Artifact evaluation (Euromicro, Supercomputing,...)



Let's try to change the System!

- Scientific papers and the Hirsch-Index (H-Index) will remain the Gold Standard for appointability.
- **We can move the publication system towards crediting software contributions!**
 - Reproducibility initiatives (ACM Journals)
 - Artifact evaluation (Euromicro, Supercomputing,...)
- Let's go even further:

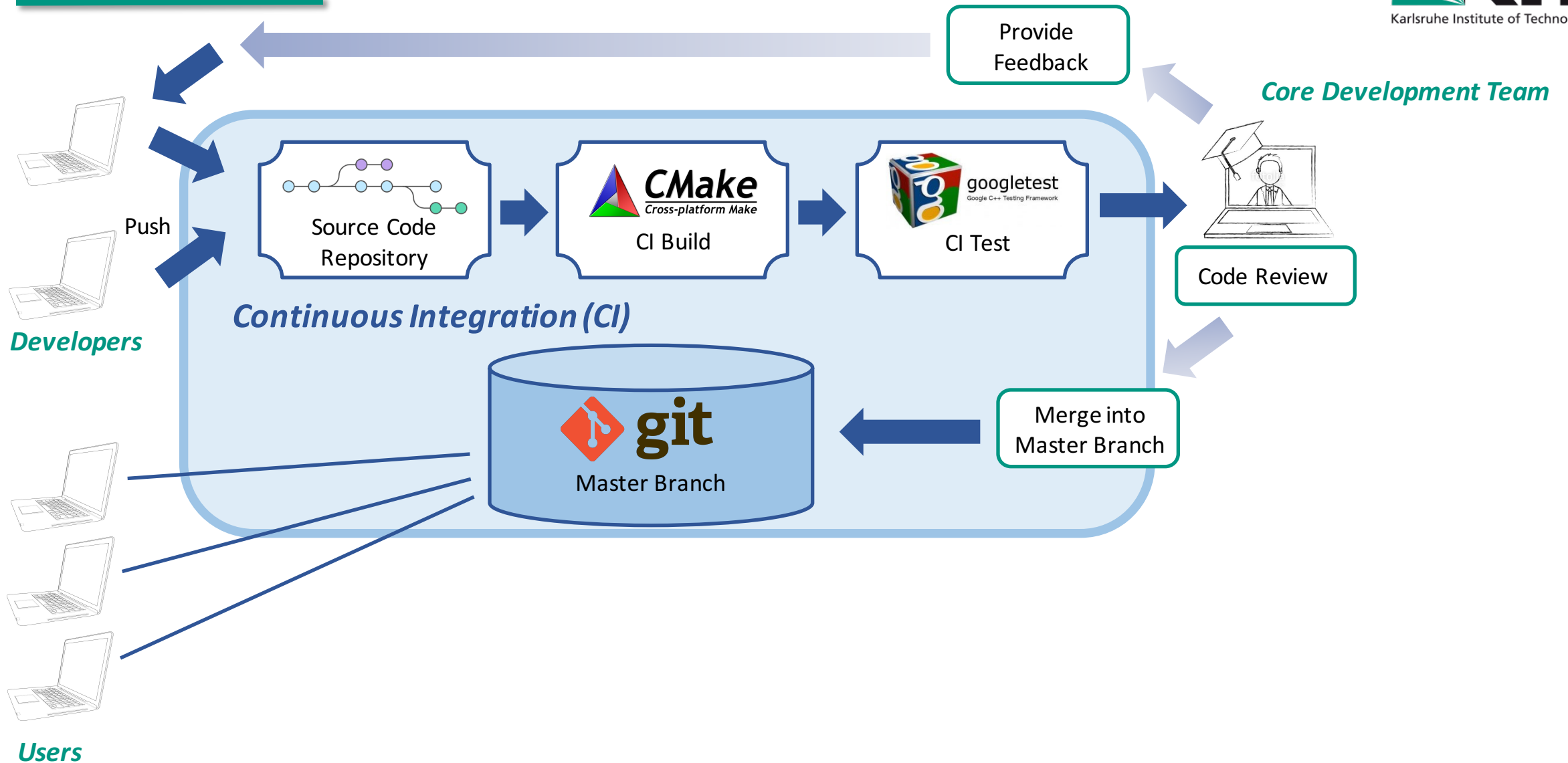


Well-documented software patches as full conference contribution

- Well-documented software patches contributing new algorithms/ implementations;
- Comprehensive code documentation and performance assessment;
- Feedback from code reviewers;
- Software patch description + performance evaluation included in conference proceedings;

- ✓ **Full reproducibility** and **traceability** is ensured;
- ✓ Not only reviewers but the complete **community can track the software patch**;
- ✓ The versioning systems helps to **identify the main contributors** of a software contribution, **ensuring full recognition**;
- ✓ The versioning systems also **links to the right person in case of technical questions**;
- ✓ **Novel algorithms** and hardware-optimized implementations are **quickly integrated into community packages**;
- ✓ The **code quality is increased** as the community can comment on the patches;
- ✓ Software patches as conference contributions naturally imply an **extremely high level of code documentation**;
- ✓ Presenting patches at a conference **makes** the whole **community aware of a new feature**;
- ✓ **Domain scientists** can directly **interact with software developers**;

A Healthy Software Development Cycle



Software Patches

- Software patches usually submitted as *merge-/push-request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.

ginkgo-project / ginkgo

Unwatch 9 Star 19 Fork 8

Code Issues 44 Pull requests 7 Projects 0 Wiki Insights

Block-interleaved block storage in block-Jacobi #159

Merged gflregar merged 3 commits into develop from interleaved_block_jacobi on Nov 26, 2018

Conversation 10 Commits 3 Checks 0 Files changed 9 +481 -169

gflregar commented on Oct 31, 2018 • edited Member

This PR further improves the performance of the block-Jacobi preconditioner for smaller block sizes by redesigning the way blocks are stored in memory. In addition to column-major storage introduced in #158, this PR interleaves the blocks to maximize coalescence when a single warp handles multiple problems. The idea is shown in the following figure, where the maximum block size allows to interleave 2 blocks to fill the cache line:

Option 1:

Option 2:

Legend:

- Jacobi block
- leading dimension
- padding

There's trade-off in both approaches depicted in the figure. Option 1 always results in aligned data access, but consumes more memory in total. Option 2 consumes less memory, but data accesses are not always aligned.

I'm currently running benchmarks for both options on PizDaint, but the results on an initial implementation of this I got before suggest that option 2 is faster.

Reviewers

- pratikvn
- hartwiganz
- tcjean

Assignees

- gflregar

Labels

- CUDA
- Core
- Enhancement
- Reference

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because your review was requested.

4 participants

Software Patches

- Software patches usually submitted as *merge-/push- request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.

The screenshot displays a GitHub pull request for the 'ginkgo-project' repository, specifically for the file 'core/preconditioner/block_jacobi.hpp'. The pull request is titled 'pratikvn reviewed on Oct 31, 2018' and includes a 'View changes' button. The code changes are highlighted in green, showing three lines of code that define a stride between two columns of a block. The pull request is reviewed by 'pratikvn' on Oct 31, 2018, and 'gflregar' on Nov 1, 2018. The review text discusses the cache line size and its impact on performance, suggesting a compile-time approximation. The pull request also includes a list of comments from 'tcojean' on Nov 1, 2018, discussing the cache line size and providing a link to a GitHub repository for cache line size information. The right sidebar shows a list of users who have reviewed the pull request, including 'pratikvn', 'artwiganzt', 'tcojean', 'gflregar', and 's'. The bottom of the sidebar shows a list of participants, including 'pratikvn', 'artwiganzt', 'tcojean', and 'gflregar'.

Software Patches

- Software patches usually submitted as *merge-/push-request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.
- The submitter can attach a performance analysis to the software patch.

ginkgo-project / ginkgo

pratikvn reviewed on Oct 31, 2018

View changes

gflgar commented on Nov 18, 2018

Member

Unlike the V100 version, where both interleaved options were a bit slower, due to some strange spikes in performance for the non-interleaved version, on the V100, interleaved storage (version 2) wins:

V100 performance of 'simple_apply' step of 'apply' stage

Block size	column-major	interleaved	padded interleaved
1	10	10	10
2	20	20	20
3	30	30	30
4	40	40	40
5	50	50	50
6	60	60	60
7	70	70	70
8	80	80	80
9	90	90	90
10	100	100	100
11	110	110	110
12	120	120	120
13	130	130	130
14	140	140	140
15	150	150	150
16	160	160	160
17	150	150	150
18	140	140	140
19	130	130	130
20	120	120	120
21	110	110	110
22	100	100	100
23	90	90	90
24	80	80	80
25	70	70	70
26	60	60	60
27	50	50	50
28	40	40	40
29	30	30	30
30	20	20	20
31	10	10	10
32	10	10	10

I'll generate more details plots and send them around tomorrow.

tcojean on Nov 1, 2018 • edited • Member

I don't know of tools to get that for all architectures (both GPU and CPU), there surely is some, but for the CPU you can at least find the information here on Linux:

```
/sys/devices/system/cpu/cpu0/cache/index0/coherency_line_size
```

Mine says 64 bytes for example. We could either get this information statically through CMake (but you have to compile on the final system) or use some executable/functions to get the information dynamically.

There is also this tool (just a simple function really) for the CPU which has Linux, MacOS and Windows compatibility.

<https://github.com/NickStrupat/CacheLineSize>

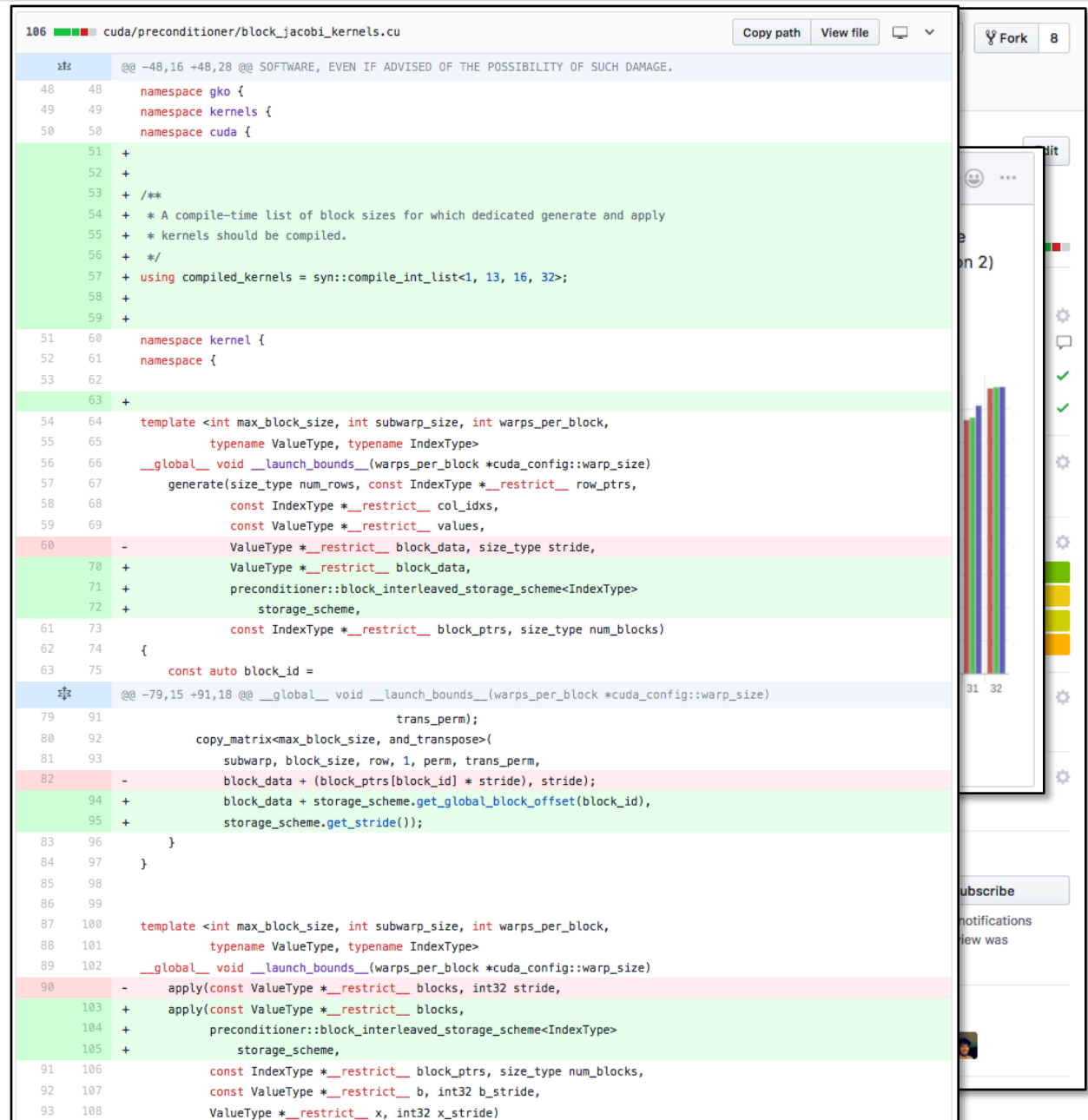
Software Patches

- Software patches usually submitted as *merge-/push-request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.
- The submitter can attach a performance analysis to the software patch.
- Software patches can either add new functionality...

```
170 core/preconditioner/block_jacobi.hpp
@@ -78,6 +78,106 @@ struct index_type<Op<ValueType, IndexType>> {
78 78     }; // namespace detail
79 79
80 80
81 + // TODO: replace this with a custom accessor
82 + /**
83 +  * Defines the parameters of the interleaved block storage scheme used by
84 +  * block-Jacobi blocks.
85 +  *
86 +  * @tparam IndexType type used for storing indices of the matrix
87 +  */
88 + template <typename IndexType>
89 + struct block_interleaved_storage_scheme {
90 +     /**
91 +      * The offset between consecutive blocks within the group.
92 +      */
93 +     IndexType block_offset;
94 +     /**
95 +      * The offset between two block groups.
96 +      */
97 +     IndexType group_offset;
98 +     /**
99 +      * Then base 2 power of the group.
100 +      *
101 +      * I.e. the group contains '1 << group_power' elements.
102 +      */
103 +     uint32 group_power;
104 +
105 +     /**
106 +      * Returns the number of elements in the group.
107 +      *
108 +      * @return the number of elements in the group
109 +      */
110 +     GKO_ATTRIBUTES IndexType get_group_size() const noexcept
111 +     {
112 +         return one<IndexType>() << group_power;
113 +     }
114 +
115 +     /**
116 +      * Computes the storage space required for the requested number of blocks.
117 +      *
118 +      * @param num_blocks the total number of blocks that needs to be stored
119 +      *
120 +      * @return the total memory (as the number of elements) that need to be
121 +      *         allocated for the scheme
122 +      */
123 +     GKO_ATTRIBUTES IndexType compute_storage_space(IndexType num_blocks) const
124 +     noexcept
125 +     {
```

Software Patches

- Software patches usually submitted as *merge-/push-request* in the *software versioning system* (e.g. Git).
- The patches are accompanied by detailed documentation explaining code functionality and feature usage.
- The community can comment and review the code.
- The submitter can attach a performance analysis to the software patch.
- Software patches can either add new functionality...
... or change / enhance existing code.



```
106 cuda/preconditioner/block_jacobi_kernels.cu
Copy path View file
Fork 8

@@ -48,16 +48,28 @@ SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
48 48 namespace gko {
49 49 namespace kernels {
50 50 namespace cuda {
51 51 +
52 52 +
53 53 + /**
54 54 + * A compile-time list of block sizes for which dedicated generate and apply
55 55 + * kernels should be compiled.
56 56 + */
57 57 + using compiled_kernels = syn::compile_int_list<1, 13, 16, 32>;
58 58 +
59 59 +
60 60 namespace kernel {
61 61 namespace {
62 62
63 63 +
64 64 template <int max_block_size, int subwarp_size, int warps_per_block,
65 65 typename ValueType, typename IndexType>
66 66 __global__ void __launch_bounds__(warps_per_block * cuda_config::warp_size)
67 67 generate(size_type num_rows, const IndexType *__restrict__ row_ptrs,
68 68 const IndexType *__restrict__ col_idxs,
69 69 const ValueType *__restrict__ values,
70 70 ValueType *__restrict__ block_data, size_type stride,
71 71 ValueType *__restrict__ block_data,
72 72 preconditioner::block_interleaved_storage_scheme<IndexType>
73 73 storage_scheme,
74 74 const IndexType *__restrict__ block_ptrs, size_type num_blocks)
75 75 {
76 76 const auto block_id =
77 77 @@ -79,15 +91,18 @@ __global__ void __launch_bounds__(warps_per_block * cuda_config::warp_size)
78 78 trans_perm);
79 79 copy_matrix<max_block_size, and_transpose>{
80 80 subwarp, block_size, row, 1, perm, trans_perm,
81 81 block_data + (block_ptrs[block_id] * stride), stride);
82 82 - block_data + storage_scheme.get_global_block_offset(block_id),
83 83 + block_data + storage_scheme.get_global_block_offset(block_id),
84 84 + storage_scheme.get_stride());
85 85 }
86 86 }
87 87
88 88 template <int max_block_size, int subwarp_size, int warps_per_block,
89 89 typename ValueType, typename IndexType>
90 90 __global__ void __launch_bounds__(warps_per_block * cuda_config::warp_size)
91 91 - apply(const ValueType *__restrict__ blocks, int32 stride,
92 92 + apply(const ValueType *__restrict__ blocks,
93 93 + preconditioner::block_interleaved_storage_scheme<IndexType>
94 94 + storage_scheme,
95 95 const IndexType *__restrict__ block_ptrs, size_type num_blocks,
96 96 const ValueType *__restrict__ b, int32 b_stride,
97 97 ValueType *__restrict__ x, int32 x_stride)
```

Software Patches as Conference Contribution

Envisioned Workflow:

1. The algorithm/implementation **developer submits a software patch to a community package** with
 - detailed description of the functionality and code documentation;
 - comprehensive performance assessment;
 - **mark the patch for a conference contribution**;
2. The **core development team and the community**
 - **comments** on the algorithm, the implementation, and the performance;
 - **reviews** and ultimately merges the patch;
3. The **developer submits the patch as a conference contribution**
 - **linking** to all documentation, performance results, and comments;
 - **acknowledging significant comments** from community;

Software Patches as Conference Contribution

Envisioned Workflow:

1. The algorithm/implementation **developer submits a software patch to a community package** with
 - detailed description of the functionality and code documentation;
 - comprehensive performance assessment;
 - **mark the patch for a conference contribution**;
2. The **core development team and the community**
 - **comments** on the algorithm, the implementation, and the performance;
 - **reviews** and ultimately merges the patch;
3. The **developer submits the patch as a conference contribution**
 - **linking** to all documentation, performance results, and comments;
 - **acknowledging significant comments** from community;
4. The **conference committee / external reviewers** do a “**light**” review of functionality, documentation, performance.
5. If accepted, the conference contribution is presented along with a **user tutorial or application examples**;
6. The submission is as a **regular paper** included in the conference proceedings
 - potentially featuring a **shorter general introduction**;
 - **including the algorithm description and performance assessment**;
potentially including **code segments, digital artifacts**, or a **link** to the merge request;
 - **listing all (significant) code reviewers / commenters**;

Summary and Limitations

Summary:

- We argue for accepting **well-documented and performance-analyzed software patches** as **regular conference contribution** included in the conference proceedings.
- **Versioning Systems** are an excellent tool to **track scientific software development** preserving **intellectual property**.

Limitations:

- **Not applicable to Position Papers** (This paper being an example 😊).
- Not applicable to **purely theoretical papers** or papers presenting an **idea**, only.
- Algorithms / implementations can **fall under export control**.



<https://bssw.io/>

In a larger picture, **accepting software patches as conference contribution** is another step **in the direction of entrenching scientific software development** as an **academic field**, and moving the academic evaluation system from traditional metrics (like the H-Index) towards **community-advancing software contributions**.