

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

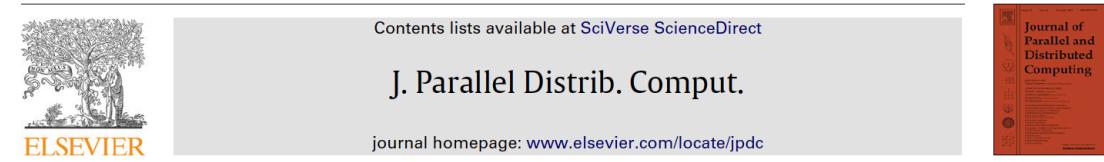
**THE 20TH IEEE INTERNATIONAL WORKSHOP ON
PARALLEL AND DISTRIBUTED SCIENTIFIC AND ENGINEERING COMPUTING
(PDSEC '19)**

**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

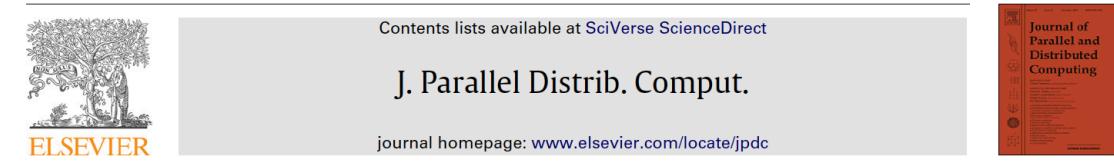
The HPC Algorithm Landscape

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

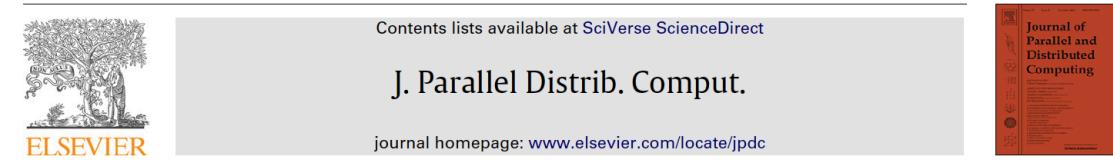
The HPC Algorithm Landscape

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

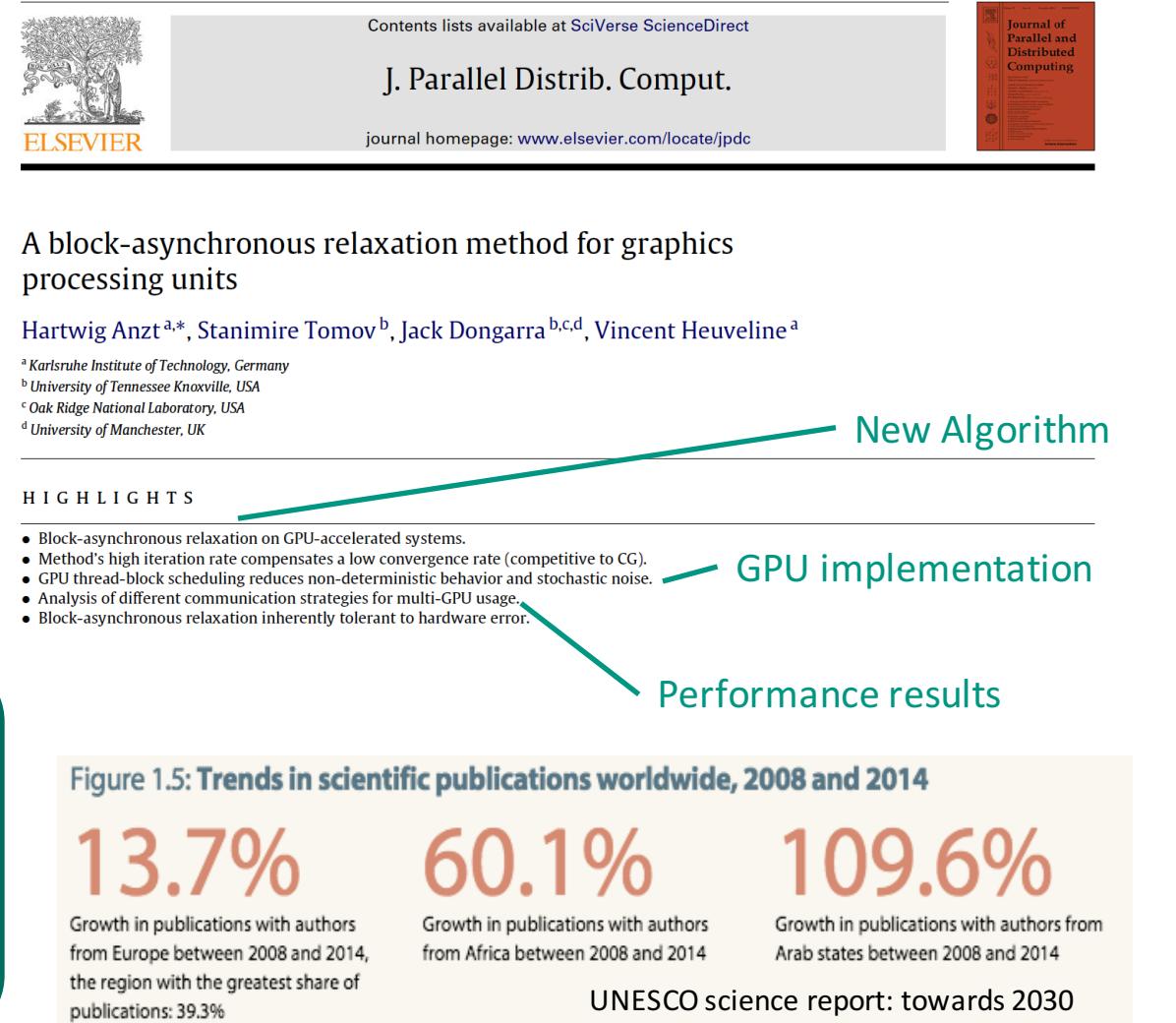
The HPC Algorithm Landscape

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;



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**;



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;



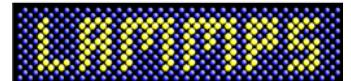
In a **perfect world**, new algorithms, implementations & performance results are

- **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**;



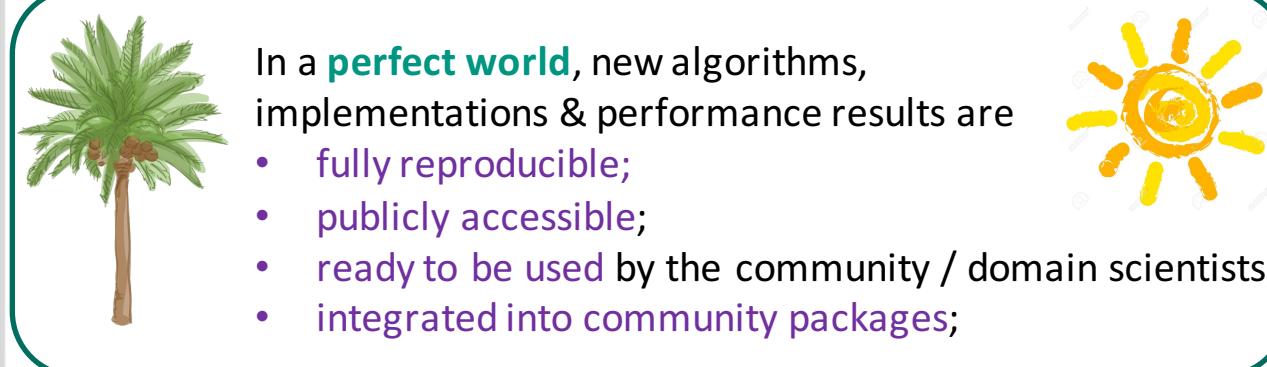
The HPC Algorithm Landscape

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;



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?

- $\text{effort(Prototype Code)} \gg \text{effort(Production Code)}$;
- 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?

- $\text{effort(Prototype Code)} \gg \text{effort(Production Code)}$;
- 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:



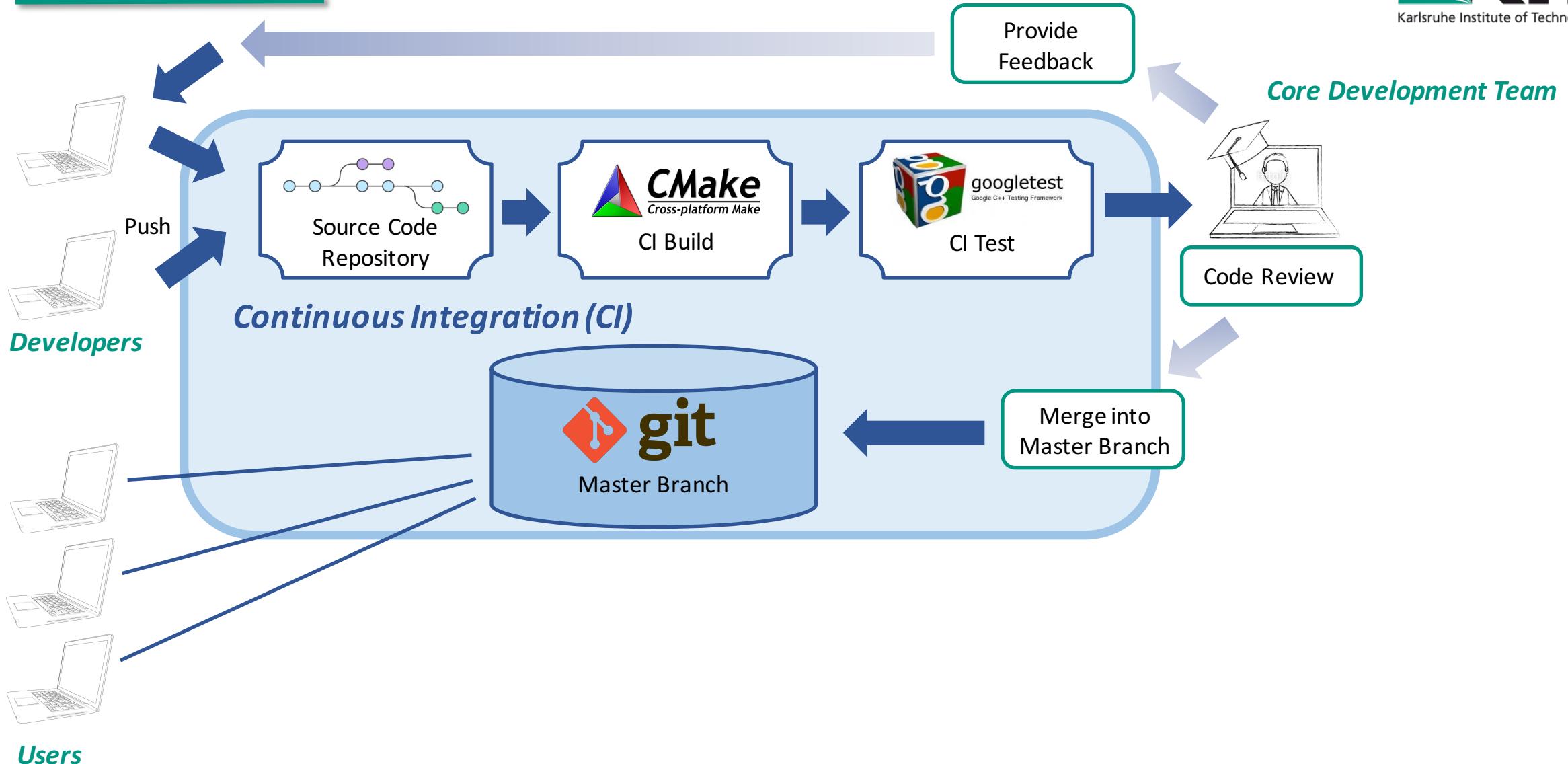
Well-documented software patches as full conference contribution

- 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;

Software Patches as Conference Contribution

- ✓ **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

Code Issues 44 Pull requests 7 Projects 0 Wiki Insights

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

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

gflegar commented on Oct 31, 2018 • edited

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:

Legend:
- Jacobi block
- leading dimension
- padding

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, hartwiganzt, tcojean

Assignees: gflegar

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 shows a GitHub pull request page for the 'ginkgo-project / ginkgo' repository. A user named 'pratikvn' has reviewed a patch on Oct 31, 2018. The patch is titled 'core/preconditioner/block_jacobi.hpp'. The code changes are as follows:

```
293 +    /**
294 +     * Stride between two columns of a block (as number of elements).
295 +     *
296 +     * Should be a multiple of cache line size for best performance.
```

Comments from 'pratikvn' on Oct 31, 2018:

The cache line size varies across machines and different hardware right? Would you not like to have this as a parameter then? Or have I misunderstood something ?

Comments from 'gflagar' on Nov 1, 2018:

You're completely right. This one is an approximation that is actually a (small) multiple of the cache line size, depending on the combination of hardware and size of the elements. (E.g. float on NVIDIA GPUs is exactly 1 cache line, double is 2, on CPUs it's more cache lines).

However having it as a parameter brings a lot of problems that are not yet solved in Ginkgo:

1. The user is required to know what is the cache line size of the system, or we somehow have to figure that out by ourselves.
2. Whatever value is passed cannot be smaller than the maximum block size, otherwise the storage scheme breaks.
3. Since the cache line size is different on the CPU than on the GPU, copying the object between them would require non-trivial storage transformations (i.e. a simple `memcpy` would not be enough).

For that reason, I've just used a compile-time approximation that works correctly (but not optimally) on all systems, until those problems are solved.

Comments from 'tcojean' on Nov 1, 2018 (edited):

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>

The sidebar on the right shows the review history with '+481 -169' changes, a list of reviewers, and a sidebar for notifications and 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 submitter can attach a performance analysis to the software patch.

ginkgo-project / ginkgo

pratikvn reviewed on Oct 31, 2018

View changes

Block Member + 88 ...

gflegar commented on Nov 18, 2018

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

GFlops

Block size

Block size	column-major (GFlops)	interleaved (GFlops)	padded interleaved (GFlops)
1	~10	~10	~10
2	~30	~35	~30
3	~50	~60	~45
4	~60	~70	~60
5	~70	~80	~70
6	~80	~90	~80
7	~90	~100	~90
8	~110	~120	~110
9	~120	~130	~120
10	~130	~125	~115
11	~125	~135	~120
12	~115	~140	~130
13	~130	~135	~135
14	~140	~145	~140
15	~150	~155	~150
16	~160	~165	~155
17	~135	~135	~130
18	~140	~140	~135
19	~145	~145	~140
20	~140	~140	~140
21	~150	~150	~145
22	~145	~145	~145
23	~155	~155	~150
24	~160	~160	~155
25	~145	~150	~145
26	~150	~150	~145
27	~155	~155	~150
28	~160	~160	~155
29	~165	~165	~160
30	~160	~160	~160
31	~170	~170	~165
32	~175	~175	~170

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>

Unsubscribe

receiving notifications
use your review was posted.

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 submitter can attach a performance analysis to the software patch.
- Software patches can either add new functionality...

The screenshot shows a GitHub pull request interface. The top bar indicates the pull request number (178) and the file being viewed (core/preconditioner/block_jacobi.hpp). The main area displays a code diff between two branches. The left column shows the current state of the code, and the right column shows the changes introduced by the pull request. The changes include comments explaining the purpose of the interleaved block storage scheme, defining parameters like group_offset and group_power, and implementing methods like get_group_size and compute_storage_space. The GitHub interface includes standard features like 'Show comments', 'Copy path', and 'View file' at the top, and a sidebar on the right with various repository and pull request details.

```
178 core/preconditioner/block_jacobi.hpp
@@ -78,6 +78,106 @@ struct index_type<Op<ValueType, IndexType>> {
78     };
79     // namespace detail
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      * @param 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.

The screenshot shows a GitHub pull request interface. The top bar indicates the pull request number (106) and the file path (cuda/preconditioner/block_jacobi_kernels.cu). The main area displays a code diff between two branches. The diff highlights changes in a CUDA kernel source file. The left side shows the original code, and the right side shows the modified code. The changes are color-coded: green for additions and red for deletions. The code itself is written in C++ with CUDA-specific syntax like __global__ and __restrict__. A vertical bar chart on the right side of the diff shows the commit history for the repository, with colors corresponding to different authors or teams. The bottom right corner of the screenshot shows a small user profile and a 'Subscribe' button.

```
106 cuda/preconditioner/block_jacobi_kernels.cu
@@ -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         +
51 60             namespace kernel {
52 61                 namespace {
53 62                     +
54 64                         template <int max_block_size, int subwarp_size, int warps_per_block,
55 65                             typename ValueType, typename IndexType>
56 66                         __global__ void __launch_bounds__(warps_per_block *cuda_config::warp_size)
57 67                             generate(size_type num_rows, const IndexType *__restrict__ row_ptrs,
58 68                                 const IndexType *__restrict__ col_idxs,
59 69                                 const ValueType *__restrict__ values,
60 70                                     ValueType *__restrict__ block_data, size_type stride,
61 71                                     ValueType *__restrict__ block_data,
62 72                                     preconditioner::block_interleaved_storage_scheme<IndexType>
63 73                                         storage_scheme,
64 74                                         const IndexType *__restrict__ block_ptrs, size_type num_blocks)
65 75             {
66 76                 const auto block_id =
67 77                     @@ -79,15 +91,18 @@ __global__ void __launch_bounds__(warps_per_block *cuda_config::warp_size)
68 91                         trans_perm);
69 92                         copy_matrix<max_block_size, and_transpose>(
70 93                             subwarp, block_size, row, 1, perm, trans_perm,
71 94                             block_data + (block_ptrs[block_id] * stride), stride);
72 95                             block_data + storage_scheme.get_global_block_offset(block_id),
73 96                             storage_scheme.get_stride());
74 97             }
75 98             +
76 99             +
77 100             template <int max_block_size, int subwarp_size, int warps_per_block,
78 101                 typename ValueType, typename IndexType>
79 102             __global__ void __launch_bounds__(warps_per_block *cuda_config::warp_size)
80 103                 - apply(const ValueType *__restrict__ blocks, int32 stride,
81 104                 + apply(const ValueType *__restrict__ blocks,
82 105                 + preconditioner::block_interleaved_storage_scheme<IndexType>
83 106                     storage_scheme,
84 107                     const IndexType *__restrict__ block_ptrs, size_type num_blocks,
85 108                     const ValueType *__restrict__ b, int32 b_stride,
86 109                     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**.