CATO

Compiler assisted source-to-source transformation of OpenMP kernels to utilise distributed memory

Jannek Squar Michael Blesel Tim Jammer Michael Kuhn Thomas Ludwig

Scientific Computing
Universität Hamburg
https://wr.informatik.uni-hamburg.de

2018-09-25

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Trend - Computer Science

1. Increasing core count of manycore architecture
   - → ↗ $\frac{\texttt{peak performance}}{\texttt{node}}$ ↗
   - → Parallelisation techniques on shared memory needed
     - **First choice: OpenMP**
2. Increasing gap between memory and CPU performance in manycore architecture
   - → ↘ $\frac{\texttt{memory}}{\texttt{CPU core}}$ ↘ favours CPU-bound applications
   - → Limited maximal problem size per node
   - → Parallelisation techniques on distributed memory needed
     - **First choice: MPI**

Trend - Natural Science

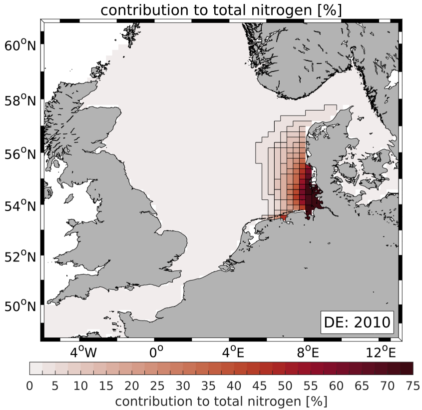
1. Scientific software often created by domain experts
   → Focus on shared memory parallelisation techniques
2. Limited problem size to fit single node memory
   → Limited horizontal scalability
3. Trivial distribution approach: Fragment problem space
   - Execute new process for each sub-problem
   → Additional overhead
   → Sub-problems must be independent

Introduction

- **Real world examples**
- Using distributed memory
- Propose a new approach

Example TBNT (I)[13]

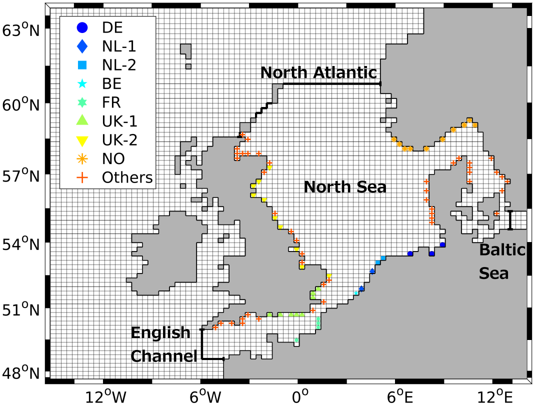
Setup North Sea:



- Trans-boundary nutrient transports (nutrients tracing)
- Postprocessing on maritime physical-biogeochemical ecosystem model

Example TBNT (II)

Setup North Sea:



- 14769 wet points (relevant grid cells)
- Runtime: $\sim \frac{\textit{30}\ \texttt{minutes}}{\textit{1}\ \texttt{simulated year}}$
- I/O: $\sim \frac{\textit{1}\ \texttt{TB}}{\textit{1}\ \texttt{simulated year}}$

Example TBNT (III)

Setup Northern Gulf of Mexico:

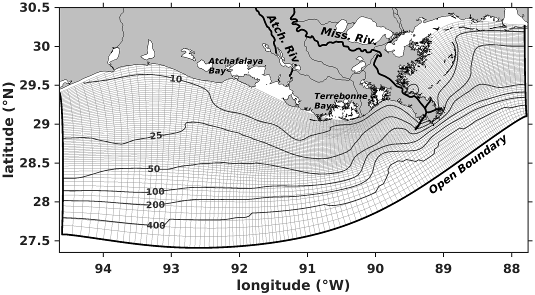
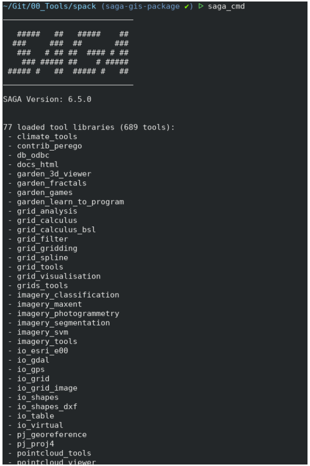


Figure: by F. Große (pers. comm.)

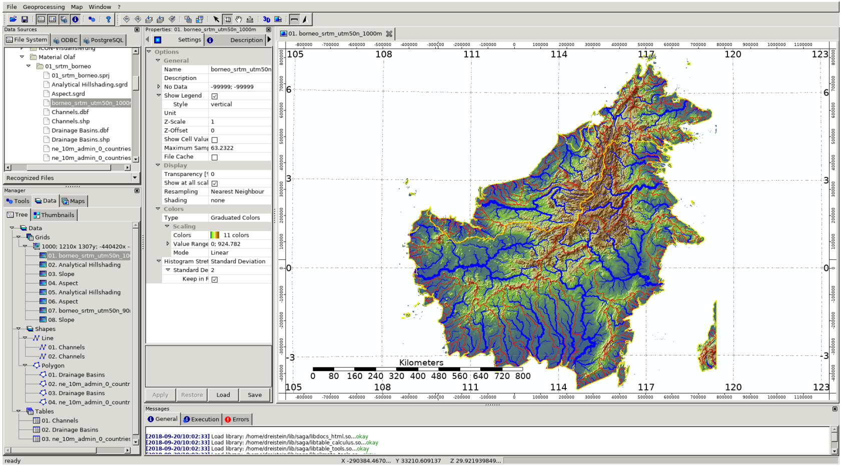
- 136660 wet points (relevant grid cells)
- Runtime: $\sim \frac{\textit{2 days}}{\textit{1 simulated year}}$

Project-Example SAGA-GIS (I) [10]

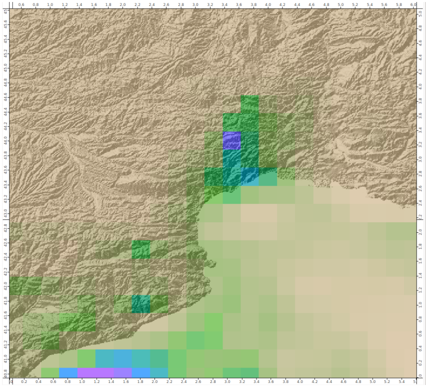


- System for Automated Geoscientific Analyses
- "Comprehensive, growing set of geoscientific methods"
- Collection of dynamically loaded tool
- CMD: Batch mode
- GUI: interactive mode

Project-Example 💡 SAGA-GIS (II)

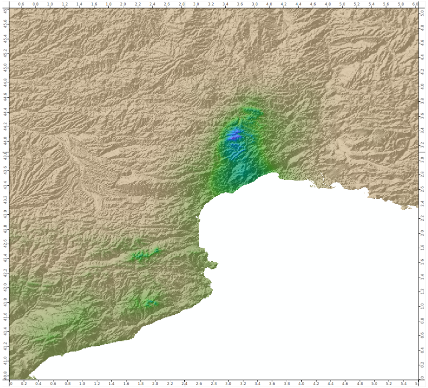


Project-Example SAGA-GIS (III) - Statistical downscaling with GFS



Downscaling

$27km \rightarrow 1km$



Project-Example SAGA-GIS (IV)

```
for(int y=0; y<Get_NY() && Set_Progress(y); y++)
{
    #pragma omp parallel for
    for(int x=0; x<Get_NX(); x++)
    {
        int Belt    = 0;     // no data
        if( !pL->is_NoData(x, y) )
        {
            [...]
        }
        pBelt->Set_Value(x, y, Belt);
    }
}
```

Listing 1: "Thermic Belt Classification"

Total occurrences in all SAGA-GIS tools:

322× **#pragma** omp parallel **for**
↪ [private(...)] [reduction(...)]

1× **#pragma** omp critical

Introduction

- Real world examples
- **Using distributed memory**
- Propose a new approach

Existing Approaches

Approaches to utilise distributed memory:

- Partitioned Global Address Space
- Single System Image
- Message Passing Interface
- ...

Partitioned Global Address Space - PGAS

?	+	−
Virtual shared memory Implementations: – Unified Parallel C – Coarray Fortran – GASPI	May use MPI-3 RMA Data locality Thread-based programming Asynchronous communication	Comparison with MPI-3 RMA[12, 14] Re-implementation necessary Additional learning effort Less control over performance

Single System Image - SSI

?	+	−
Centralised system view	No code changes	Poor performance[20]
System property	No user interaction	Poor scalability[20]
Implementations: JUMP[1], TreadMarks[3]	Easy to use	Neglected development

Message Passing Interface - MPI

?	+	−
Message passing	Active development	Additional learning effort
Processes	RDMA	Buffer management
Explicit communication		Re-implementation necessary

Existing approaches to transform OpenMP into MPI (I)

- Basumallik and Eigenmann (2005)
- An Mey and Tedjo (2006)
- Millot et al. (2008)
- Saa-Garriga et al. (2015)
- ...

Commonalities:

- Transformation into readable MPI code
- No one-sided MPI communication

Introduction

- Real world examples
- Using distributed memory
- **Propose a new approach**

Another Possible Solution

- Automatic solution to relieve domain experts
  - ⊕ Easy to apply for application
  - ⊖ Probably less scalability than handwritten code
  - ⇒ Focus rather on improved horizontal scaling than absolute runtime
- Compiler-based approach
  - Local installation possible
  - Robustness
    - Additional layer of abstraction
    - Based on existing framework
    - Independent of language selection
- Use latest features of MPI-3

Our solution: CATO

Based on:

- One-sided MPI-3 communication
- LLVM compiler infrastructure

CATO specification:

- Requirements
- CATO Workflow
- Memory Handling

One-sided MPI operations

- Collective offering of memory through windows

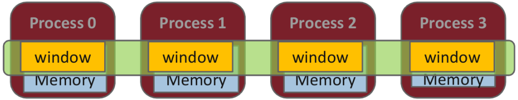


Figure: RMA Window[6]

- One-sided communication: *origin* $\rightarrow$ *target*
- Synchronisation through epochs:
  - Active target synchronisation
  - Generalized active target synchronisation
  - Passive target synchronisation
- Hardware-support through RDMA

Example 💡 Generalized active target synchronisation

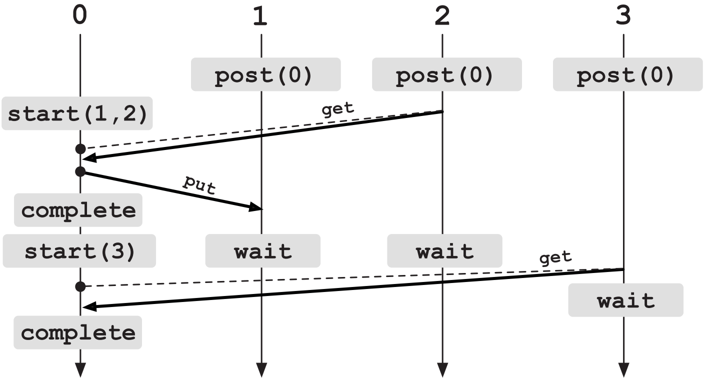


Figure: PSCW-Synchronisation[7]

LLVM

- Modular compiler infrastructure
- Active community
- Code adaptation through passes (shared libraries)

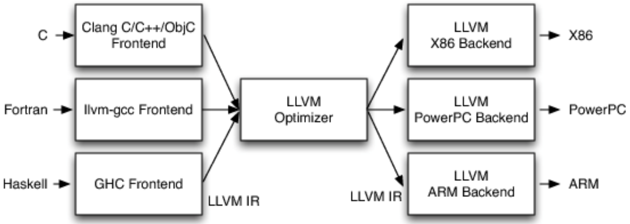


Figure: Modular compiler infrastructure [18]

Using an LLVM pass

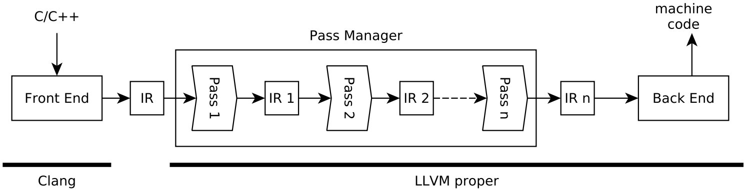


Figure: Pass integration, based on [22]

```
#Compile pass
$ clang++ -fno-rtti -fPIC -shared -o pass.so pass.cpp
#Link pass into application
$ mpicxx -cxx=clang++ -fopenmp -Xclang -load -Xclang pass.so -o app.x app.cpp
```

Listing 2: Linking an LLVM module pass

Example 💡 Intermediate Representation Language (I)

```c
int *counter = (int*)malloc(1*sizeof(int));
*counter = 0;
#pragma omp parallel
{
    printf("Hello from thread %d\n", omp_get_thread_num());
    #pragma omp critical
    (*counter)++;
}
```

Listing 3: Example C application (extract)

Example 💡 Intermediate Representation Language (II)

```
define i32 @main() #0 {
    %1 = alloca i32*, align 8
    %3 = call noalias i8* @malloc(i64 4) #3
    %4 = bitcast i8* %3 to i32*
    store i32* %4, i32** %1, align 8
    [...]
    call void ( [...] ) @__kmpc_fork_call( [...] @.omp_outlined. to void ( [...] )), i32** %1)
    [...]
    ret i32 0
}

define internal void @.omp_outlined.([...]) #0 {
    [...]
    %8 = call i32 @omp_get_thread_num()
    %9 = call i32 (i8*, ...) @printf([...] i32 %8)
    [...]
    call void @__kmpc_critical(%ident_t* @0, i32 %11, [8 x i32]* @.gomp_critical_user_.var)
    %12 = load i32*, i32** %7, align 8
    %13 = load i32, i32* %12, align 4
    %14 = add nsw i32 %13, 1
    store i32 %14, i32* %12, align 4
    call void @__kmpc_end_critical(%ident_t* @0, i32 %11, [8 x i32]* @.gomp_critical_user_.var)
    ret void
}
```

Our solution: CATO

Based on:

- One-sided MPI-3 communication
- LLVM compiler infrastructure

CATO specification:

- Requirements
- CATO Workflow
- Memory Handling

Ambitions

CATO: **C**ompiler **A**ssisted s2s **T**ransformation of **O**penMP kernels

- Easy usage use cases:
  - Minimal effort : user adjusts compilation calls
  - Normal effort : user adjusts problem size
  - Maximal effort : user annotates code, execute profiler
- Use one-sided MPI communication
- Set up on well-known, popular compiler-suite: LLVM
- No need to generate readable MPI code

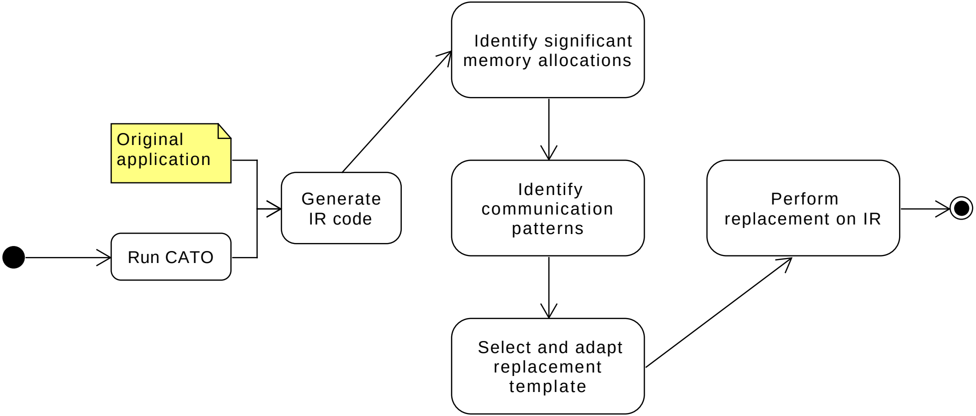
Target Audience

- Who?
  - Application without MPI parallelisation
  - Teams without budget/time to do MPI parallelisation themselves
- What?
  - Better explanatory power through model run on extended problem size
    - ↗ Problem area ↗
    - ↗ Resolution ↗
    - ↗ Grid dimension ↗
- How?
  - Execute application on distributed cluster
  - Fit problem in combined node memory

CATO workflow (minimal effort use case)

1. User replaces compiler call with CATO wrapper-script
2. LLVM frontend translates application code into IR
3. CATO adjusts IR
4. LLVM backend translates IR into machine code
5. User executes generated binary via `mpiexec`

Planned Design - UML Activity Diagram



Transform Communication Pattern

Fixed sequence of replacement steps:

1. Insert MPI initialisation and finalisation calls in `main` method
2. Identify OpenMP kernels
   - Location
   - Classification
3. Replace `__kmpc_fork_call`
4. Load and apply MPI RMA Equivalence Class (EC) templates to handle shared memory
   - Distribute shared memory
   - Replace local load and store operations
   - Ensure memory consistency

Considered Communication Patterns

- Classification[17] of OpenMP kernels based on 7+6 *dwarves*[5, 9]
  - Important patterns for science and engineering
  - Similar communication and data movement patterns
  - Similar computation patterns

→ Provide Equivalence Class templates
  - Preserve semantic of OpenMP kernel, equivalent behaviour
  - One-sided MPI-3 communication operations
  - Written in C++
  - Shared object to load during runtime

7+6 dwarves

Original dwarves:

- Dense Linear Algebra
- Sparse Linear Algebra
- Spectral Methods
- N-Body Methods
- Structured Grids
- Unstructured Grids
- Monte Carlo

Additional dwarves:

- Combinational Logic
- Graph Traversal
- Dynamic Programming
- Backtrack & Branch and Bound
- Graphical Models
- Finite State Machine

Memory Allocation

- Actual behaviour depends on identified communication pattern
- Single value variable
  - Master based
  - Duplicated
- Struct
  - Stored in global dynamic window
  - Pointer replaced by new struct with meta information
- Array
  - Master based
  - Duplicated
  - Distributed
    - Load whole array line ("cache") if possible
    - Presume continuous memory

Status Quo & Perspective

- Example: Partdiff
- OpenMP pragmas
- Future Work

Example Partdiff

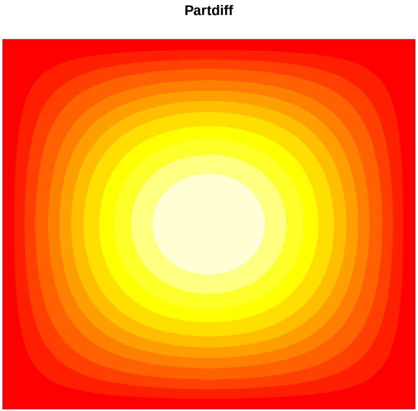


Figure: Example output of partdiff

- Partial differential equation solver
  - Gauß-Seidel method
  - Jacobi method
- Used for teaching
- Testmachine:
  - 2 × Intel Xeon X5650 @ 2.67GHz
  - 6 cores per CPU
  - Hyperthreading activated

Preliminary Results ⚠ Scaling [17]

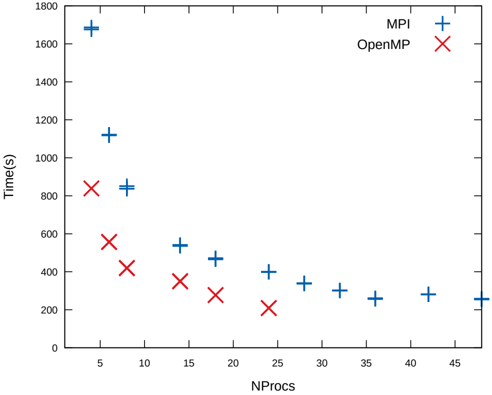


Figure: Strong scaling

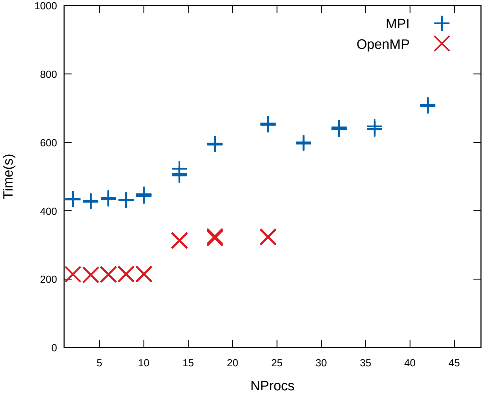


Figure: Weak scaling

Excerpt of considered OpenMP Pragmas [1]

Scope	parallel, single, master, task, sections, for
Clauses	num_threads, private, firstprivate, threadprivate, shared, reduction
Synchronisation	barrier, critical, atomic
Scheduling	static, dynamic, guided, auto, runtime
Functions	get_thread_num, get_num_threads
Misc	target, simd

[1] colour key: not planned, planned, work in progress

Static EC template checker

- Essential primary target: `OpenMP kernel` $\equiv$ `MPI replacement`
- Non-trivial usage of MPI-3 RMA
- Enhance confidence in memory consistency[15]:
  - Based on the idea of MPI checker[11]
  - Analyse usage of RMA synchronisation
  - Trace corresponding operations through state machine

Estimate optimal distribution of OpenMP kernel based on ...

... application level

- Evaluate optional user annotations
- Static code analysis of memory usage
- Runtime profiling of memory usage

... hardware level

- Micro-benchmarks
- Adapted roofline model[23]

⇒ Tune pass through determined environment[2] configuration

[2] hardware+application

Further development steps

- Reinsert OpenMP kernels after distribution[16]
- Give up focus on single OpenMP kernel
- Survey of used OpenMP pragmas
  - Focus development of CATO on most used OpenMP constructs
  - Focus on non-HPC codes in public repositories
  - Consider scheduling clauses
- Introduce Polly[2] into workflow
- Develop profiling components
- Comparison with existing approaches
- ....

Summary

- Aim for an easy usage
- Communication pattern $\rightarrow \frac{\texttt{communication}}{\texttt{memory redundancy}}$ balance
- Functional prototyp of CATO
- Ongoing development of additional features and general improvements
- ↗ Increased usable memory ↗
- ↗ Increased maximal problem size ↗
- ↘ Probably loss of performance (increased absolute runtime) ↘

5 References

References I

[1] The jump software dsm system. http://www.snrg.cs.hku.hk/srg/html/jump.htm. URL http://www.snrg.cs.hku.hk/srg/html/jump.htm. Accessed: 2018-09-06.

[2] Polly - LLVM Framework for High-Level Loop and Data-Locality Optimizations. https://polly.llvm.org/. Accessed: 06.09.2018.

[3] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, Honghui Lu, R. Rajamony, Weimin Yu, and W. Zwaenepoel. Treadmarks: shared memory computing on networks of workstations. *Computer*, 29(2):18–28, Feb 1996. ISSN 0018-9162. doi: 10.1109/2.485843.

[4] Dieter An Mey and Irene Tedjo. Adaptive Integration-Form OpenMP to MPI. 2006.

[5] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[6] Pavan Balaji, Rajeev Thakur, Ken Raffenetti, and Yanfei Guo. Parallel Programming with MPI.

References II

[7] Brian Barrett, Torsten Hoefler, James Dinan, Rajeev Thakur, Pavan Balaji, Bill Gropp, and Keith Douglas Underwood. Remote memory access programming in mpi-3. Technical report, Sandia National Laboratories, 2013.

[8] Ayon Basumallik and Rudolf Eigenmann. Towards automatic translation of OpenMP to MPI. In *Proceedings of the 19th annual international conference on Supercomputing - ICS '05*, ICS '05, page 189, New York, New York, USA, 2005. ACM Press. ISBN 1595931678. doi: 10.1145/1088149.1088174. URL http://doi.acm.org/10.1145/1088149.1088174http://portal.acm.org/citation.cfm?doid=1088149.1088174.

[9] Phillip Colella. Defining software requirements for scientific computing. 2004.

[10] O. Conrad, B. Bechtel, M. Bock, H. Dietrich, E. Fischer, L. Gerlitz, J. Wehberg, V. Wichmann, and J. Böhner. System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. *Geoscientific Model Development*, 8(7):1991–2007, 2015. doi: 10.5194/gmd-8-1991-2015. URL https://www.geosci-model-dev.net/8/1991/2015/.

References III

[11] Alexander Droste, Michael Kuhn, and Thomas Ludwig. Mpi-checker: Static analysis for mpi. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 3:1–3:10, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-4005-2. doi: 10.1145/2833157.2833159. URL http://doi.acm.org/10.1145/2833157.2833159.

[12] Robert Gerstenberger, Maciej Besta, and Torsten Hoefler. Enabling highly-scalable remote memory access programming with MPI-3 one sided. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13*, pages 1–12, New York, New York, USA, 2013. ACM Press. ISBN 9781450323789. doi: 10.1145/2503210.2503286. URL http://dl.acm.org/citation.cfm?doid=2503210.2503286.

[13] Fabian Große, Markus Kreus, Hermann-Josef Lenhart, Johannes Pätsch, and Thomas Pohlmann. A novel modeling approach to quantify the influence of nitrogen inputs on the oxygen dynamics of the north sea. 4:383, 11 2017.

References IV

[14] Jeff R Hammond, Sayan Ghosh, and Barbara M Chapman. Implementing OpenSHMEM Using MPI-3 One-Sided Communication. In Stephen Poole, Oscar Hernandez, and Pavel Shamis, editors, *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools: First Workshop, OpenSHMEM 2014, Annapolis, MD, USA, March 4-6, 2014. Proceedings*, pages 44–58. Springer International Publishing, Cham, 2014. ISBN 978-3-319-05215-1. doi: 10.1007/978-3-319-05215-1_4. URL http://dx.doi.org/10.1007/978-3-319-05215-1{_}4.

[15] Marcel Heing. Verification of one-sided MPI communication code using static analysis in LLVM. unpublished M.Sc. thesis, 2019.

[16] Torsten Hoefler, James Dinan, Darius Buntinas, Pavan Balaji, Brian Barrett, Ron Brightwell, William Gropp, Vivek Kale, and Rajeev Thakur. MPI + MPI: a new hybrid approach to parallel programming with MPI plus shared memory. *Computing*, 95(12):1121–1136, dec 2013. ISSN 0010-485X. doi: 10.1007/s00607-013-0324-2. URL http://link.springer.com/10.1007/s00607-013-0324-2.

[17] Tim Jammer. Characterization and translation of openmp use cases to mpi using llvm. unpublished M.Sc. thesis, 2018.

References V

[18] Chris Lattner. The architecture of open source applications. http://www.aosabook.org/en/llvm.html. URL http://www.aosabook.org/en/llvm.html.

[19] Daniel Millot, Alain Muller, Christian Parrot, and Frédérique Silber-Chaussumier. STEP: A Distributed OpenMP for Coarse-Grain Parallelism Tool. pages 83–99. 2008. doi: 10.1007/978-3-540-79561-2_8. URL http://link.springer.com/10.1007/978-3-540-79561-2{_}8.

[20] David Padua, editor. *Encyclopedia of Parallel Computing*. Springer, 2011 edition, 2011. ISBN 9780387097657.

[21] Albert Saa-Garriga, David Castells-Rufas, and Jordi Carrabina. OMP2MPI: Automatic MPI code generation from OpenMP programs. feb 2015. URL https://arxiv.org/pdf/1502.02921.pdfhttp://arxiv.org/abs/1502.02921.

[22] Adrian Sampson. Llvm for grad students. http://www.cs.cornell.edu/~asampson/blog/llvm.html. URL http://www.cs.cornell.edu/~asampson/blog/llvm.html.

References VI

[23] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. *Communications of the Association for Computing Machinery*, 2 2009. doi: 10.1145/1498765.1498785.