Automatic distribution of OpenMP kernels using CATO

Jannek Squar Tim Jammer Michael Blesel Michael Kuhn Thomas Ludwig

2021-03-25

Universität Hamburg
squar@informatik.uni-hamburg.de

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Table of Contents

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. Focus on shared memory parallelisation techniques (e.g. OpenMP)
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

Motivation

- Better explanatory power through model run on extended problem size
  - Problem area
  - Resolution
  - Grid dimension
- OpenMP:
  - Easy to use ...
  - ... but limited to shared memory
- MPI:
  - More difficult to use ...
  - ... but allows distributed computing

Goals:

- Automatic distribution of input problem
- Minimal data redundancy

Alternative Approaches

- Distributed Shared Memory
  - PGAS [Pad11] (e.g. UPC, Coarray Fortran, *TreadMarks* [ACD+96])
  - *XcalableMP* [Xca], *OpenMPD* [LSB07]
  - *SSI* [BCJ01], *Cluster OpenMP* [Hoe06]
- Existing MPI approaches [BE05, SCC15, MMPS08, HFE11, AT06]

Not suitable to achieve our goal, because:

- Require changes to code or OS/FS
- Focus on optimisation of performance rather than of problem size
- Some not actively maintained anymore

Table of Contents

- Message Passing Interface
- Communication between processes
- User must pay attention to:
  - memory consistency
  - synchronisation
- Established in HPC

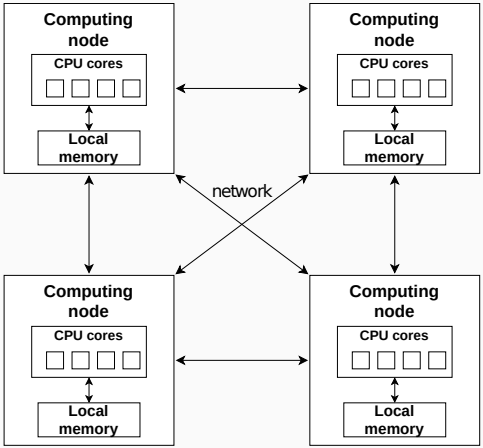


Figure 1: Distributed computing [Ble21]

- Modular open source compiler infrastructure
- Abstraction layer: *Intermediate Representation* (IR)

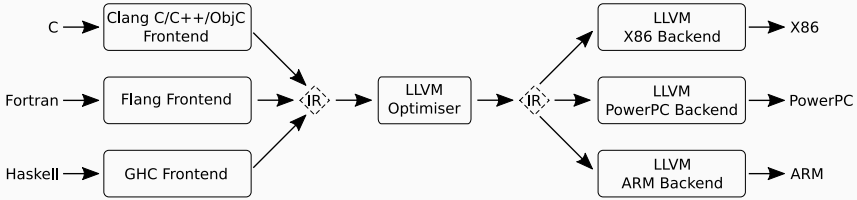


Figure 2: Modular compiler infrastructure (based on [Lat])

Frontend $\rightarrow$ IR

Optimisation $\rightarrow$ Analysis/transformation of IR

Backend $\rightarrow$ Binary

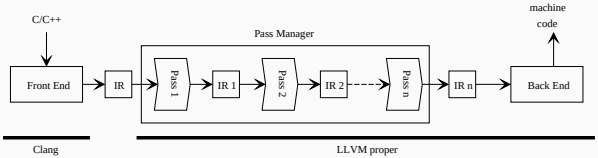


Figure 3: LLVM workflow: Frontend $\rightarrow$ Optimizer $\rightarrow$ backend (based on [Sam])

Intermediate Representation Language (I)

```
1  int *counter = (int*)malloc(1*sizeof(int));
2  *counter = 0;
3  #pragma omp parallel
4  {
5      printf("Hello from thread %d\n", omp_get_thread_num());
6      #pragma omp critical
7      (*counter)++;
8  }
```

Listing 1: Example C application (extract)

Intermediate Representation Language (II)

```
    %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
}
```

CATO (**c**ompiler **a**ssisted source **t**ransformation of **O**penMP kernels)

- Main component: LLVM transform pass
  - Local installation + Robustness
- Derive information about suitable data distribution from OpenMP kernels
  → Shared memory might be a candidate for distribution
- Automated code replacement
  - Probably less scalable...
  - ... but focus on eased optimisation of memory usage and horizontal scaling
  - Eased usage of advanced features of MPI-3

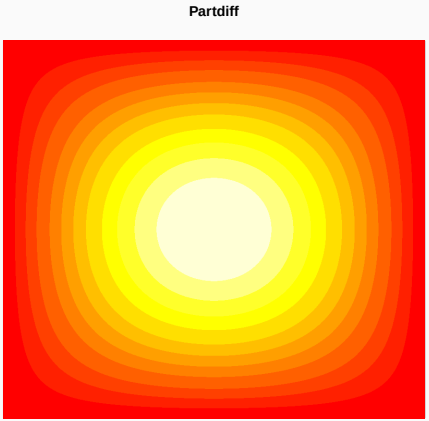
  → Can be used by untrained personnel

Table of Contents

User Workflow

1. Replace compiler call with CATO wrapper script
2. (*optional*) Provide expert knowledge by adding CATO pragmas
3. LLVM frontend translates original code into IR
4. CATO analyses and transforms IR
5. LLVM backend translates IR into machine code
6. Execute binary via `mpiexec` or `srun`

Benchmark Setup



- Partial differential equation solver
  - Jacobi method
  - (Gauß-Seidel method)
- Testmachine (2 nodes):
  - 2 $\times$ Intel Xeon X5650 @ 2.67GHz
  - 6 cores per CPU with hyperthreading
  - 12GB RAM
  - 1Gb Ethernet
  - Ubuntu@18.04, Linux@4.15, MPICH@3.3.1

Figure 4: Example output of partdiff

Scaling Behaviour

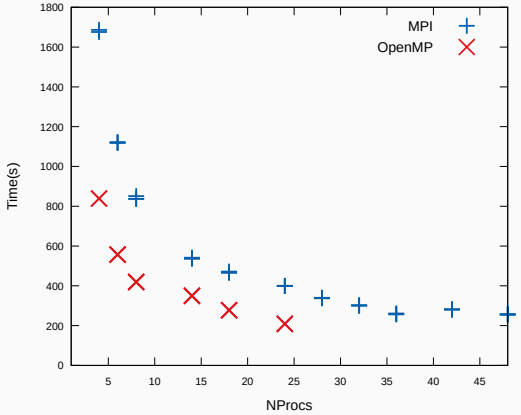


Figure 5: Strong scaling

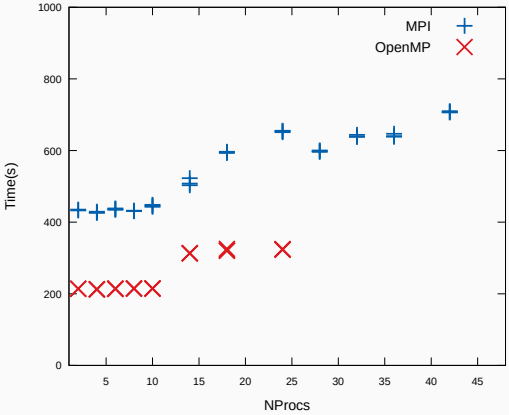


Figure 6: Weak scaling

Table of Contents

Previous and current related topics (BSc/MSc):

2017: Initial tool implementation [Ble17]

2018: Improve implementation and consideration of patterns [Jam18]

2019: Static code analysis for MPI verification [HB19]

TBD: Performance comparison of two-sided and one-sided MPI communication

TBD: Static and dynamic code analysis to derive communication pattern

Ongoing Improvements

Memory Consumption and I/O backends

- Avoid sequential data initialisation by master process
  - Avoid peak memory consumption before distribution
  - (Potential speedup by parallelised I/O phase)
- Limitation: Application must allocate memory itself
  - I/O library handles memory: CATO not feasible!
  - I/O receives pointer by application: CATO replaces I/O calls

$\Rightarrow$ Implementation of NetCDF4 backend is ongoing

Prospective Improvements

- Optimise memory chasing via user tree
- Reinsert OpenMP threads
- Automatise communication pattern recognition
  - Heuristics
  - Reuse well-established MPI versions of communication pattern
  - Make use of more advanced MPI constructs
- Move into public repository `https://github.com/jsquar/cato`

Summary

- Functional prototype of CATO
- Automatic replacement of OpenMP kernels through equivalent MPI operations
- Focus on increasing input size through distribution
  - Chasing and handling shared memory
  - Improve horizontal scaling

$\Rightarrow$ Reach problem size beyond the possibility of OpenMP

Table of Contents

References

[ABC+06] 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*, Tech. report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[ACD+96] Cristiana Amza, Alan L Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel, *Treadmarks: Shared memory computing on networks of workstations*, Computer **29** (1996), no. 2, 18–28.

[AT06] Dieter An Mey and Irene Tedjo, *Adaptive Integration - Form OpenMP to MPI*.

[BCJ01] Rajkumar Buyya, Toni Cortes, and Hai Jin, *Single system image*, Int. J. High Perform. Comput. Appl. **15** (2001), no. 2, 124–135.

[BE05] Ayon Basumallik and Rudolf Eigenmann, *Towards automatic translation of openmp to MPI*, Proceedings of the 19th Annual International Conference on Supercomputing, ICS 2005, Cambridge, Massachusetts, USA, June 20-22, 2005 (Arvind and Larry Rudolph, eds.), ACM, 2005, pp. 189–198.

[Ble17] Michael Blesel, *Compiler assisted translation of OpenMP to MPI using LLVM*, Online `https://wr.informatik.uni-hamburg.de/_media/research:theses:michael_blesel_compiler_assisted_translation_of_openmp_to_mpi_using_llvm.pdf`, 10 2017.

[Ble21] Michael Blesel, *Message passing safety and correctness checks at compile time using Rust*, Master's thesis, Universität Hamburg, 02 2021.

[Col04] Phillip Colella, *Defining software requirements for scientific computing*.

[HB19] Marcel Heing-Becker, *Verification of one-sided MPI communication code using static analysis in LLVM*, Master's thesis, Universität Hamburg, 02 2019.

[HDT+15] Torsten Hoefler, James Dinan, Rajeev Thakur, Brian Barrett, Pavan Balaji, William Gropp, and Keith Underwood, *Remote Memory Access Programming in MPI-3*, ACM Transactions on Parallel Computing **2** (2015), no. 2, 1–26.

References

[HFE11] Khaled Hamidouche, Joel Falcou, and Daniel Etiemble, *A framework for an automatic hybrid mpi+openmp code generation*, 2011 Spring Simulation Multi-conference, SpringSim '11, Boston, MA, USA, April 03-07, 2011. Volume 6: Proceedings of the 19th High Performance Computing Symposia (HPC) (Layne T. Watson, Gary W. Howell, William I. Thacker, and Steven Seidel, eds.), SCS/ACM, 2011, pp. 48–55.

[Hoe06] Jay P. Hoeflinger, *Extending openmp to clusters*, White Paper, Intel Corporation (2006).

[Jam18] Tim Jammer, *Characterization and translation of OpenMP use cases to MPI using LLVM*, Master's thesis, Universität Hamburg, 12 2018.

[Lat] Chris Lattner, *The architecture of open source applications*, `http://www.aosabook.org/en/llvm.html`.

[LLV18] LLVM Project, *Writing an LLVM Pass*, 2018.

[LSB07] Jinpil Lee, Mitsuhisa Sato, and Taisuke Boku, *Design and implementation of openmpd: An openmp-like programming language for distributed memory systems*, A Practical Programming Model for the Multi-Core Era, 3rd International Workshop on OpenMP, IWOMP 2007, Beijing, China, June 3-7, 2007, Proceedings (Barbara M. Chapman, Weimin Zheng, Guang R. Gao, Mitsuhisa Sato, Eduard Ayguadé, and Dongsheng Wang, eds.), Lecture Notes in Computer Science, vol. 4935, Springer, 2007, pp. 143–147.

[MMPS08] Daniel Millot, Alain Muller, Christian Parrot, and Frédérique Silber-Chaussumier, *STEP: A distributed openmp for coarse-grain parallelism tool*, OpenMP in a New Era of Parallelism, 4th International Workshop, IWOMP 2008, West Lafayette, IN, USA, May 12-14, 2008, Proceedings (Rudolf Eigenmann and Bronis R. de Supinski, eds.), Lecture Notes in Computer Science, vol. 5004, Springer, 2008, pp. 83–99.

[Pad11] David A. Padua (ed.), *Encyclopedia of parallel computing*, Springer, 2011.

[Sam] Adrian Sampson, *Llvm for grad students*, `http://www.cs.cornell.edu/~asampson/blog/llvm.html`.

[SCC15] Albert Saà-Garriga, David Castells-Rufas, and Jordi Carrabina, *OMP2MPI: automatic MPI code generation from openmp programs*, CoRR **abs/1502.02921** (2015).

[Xca] XcalableMP Specification Working Group, *Xcalablemp language specification*.

Table of Contents

Considered Communication Patterns

- Classification[Jam18] of OpenMP kernels based on 7+6 *dwarves*
  - Important patterns for science and engineering
  - Similar communication and data movement patterns
  - Similar computation patterns

→ Provide Equivalence Class templates

  - Preserve semantics of OpenMP kernel, equivalent behaviour
  - One-sided MPI-3 communication operations
  - Written in C++
  - Shared object to load during tool-runtime

7+6 dwarves

Original dwarves: [Col04]

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

Additional dwarves: [ABC$^+$06]

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

Workflow - CATO (I)

1. MPI initialisation
2. Memory allocation
3. Memory initialisation
4. Microtask execution
5. Memory interaction
6. MPI finalisation

```
1  int main()
2  {
3      int* a = (int*)malloc(sizeof(int)*4);
4      int* b = (int*)malloc(sizeof(int)*4);
5
6      a[0] = 0;
7      a[1] = 1;
8      a[2] = 2;
9      a[3] = 3;
10
11     #pragma omp parallel for
12     {
13         for(int i = 0; i < 4; i++)
14         {
15             b[i] = a[i];
16         }
17     }
18
19     printf("[%d,%d,%d,%d]\n", b[0], b[1], b[2], b[3]);
20
21     free(a);
22     free(b);
23 }
```

Listing 2: Original code

Step 1: MPI Initialisation

```
14 define dso_local i32 @main() #0 {
15 entry:
16   %a = alloca i32*, align 8
17   %b = alloca i32*, align 8
18   %call = call noalias i8* @malloc(i64 16) #4
19   %0 = bitcast i8* %call to i32*
20   store i32* %0, i32** %a, align 8
```

Listing 3: Original IR

```
14 define dso_local i32 @main() #0 {
15 entry:
16   %0 = alloca i32
17   %1 = alloca i32
18   %2 = alloca i32
19   %3 = alloca i32
20   %4 = alloca i32
21   %5 = alloca i32
22   %6 = alloca i32
23   %7 = alloca i32
24   %8 = alloca i32
25   call void @_Z15cato_initializeb(i1 ↩
         ↪ false)
26   br label %entry.split
27
28 entry.split:
29   %a = alloca i32*, align 8
30   %b = alloca i32*, align 8
```

Listing 4: Transformed IR

Step 2: Memory Allocation

```
14 define dso_local i32 @main() #0 {
15 entry:
16   %a = alloca i32*, align 8
17   %b = alloca i32*, align 8
18   %call = call noalias i8* @malloc(i64 16) #4
19   %0 = bitcast i8* %call to i32*
20   store i32* %0, i32** %a, align 8
21   %call1 = call noalias i8* @malloc(i64 16) #4
22   %1 = bitcast i8* %call1 to i32*
23   store i32* %1, i32** %b, align 8
24   %2 = load i32*, i32** %a, align 8
```

Listing 5: Original IR

```
28 entry.split:
29   %a = alloca i32*, align 8
30   %b = alloca i32*, align 8
31   %call = call i8* ↩
         ↪ @_Z22allocate_shared_memorylii(i64 ↩
         ↪ 16, i32 1275069445, i32 1)
32   %9 = bitcast i8* %call to i32*
33   store i32* %9, i32** %a, align 8
34   %call1 = call i8* ↩
         ↪ @_Z22allocate_shared_memorylii(i64 ↩
         ↪ 16, i32 1275069445, i32 1)
35   %10 = bitcast i8* %call1 to i32*
36   store i32* %10, i32** %b, align 8
37   %11 = load i32*, i32** %a, align 8
```

Listing 6: Transformed IR

Step 3: Memory Initialisation


```
25  %arrayidx = getelementptr inbounds i32, i32* %2, i64 0
26  store i32 0, i32* %arrayidx, align 4
27  %3 = load i32*, i32** %a, align 8
28  %arrayidx2 = getelementptr inbounds i32, i32* %3, i64 1
29  store i32 1, i32* %arrayidx2, align 4
30  %4 = load i32*, i32** %a, align 8
31  %arrayidx3 = getelementptr inbounds i32, i32* %4, i64 2
32  store i32 2, i32* %arrayidx3, align 4
33  %5 = load i32*, i32** %a, align 8
34  %arrayidx4 = getelementptr inbounds i32, i32* %5, i64 3
35  store i32 3, i32* %arrayidx4, align 4
```

Listing 7: Original IR

```
38  %arrayidx = getelementptr inbounds i32, i32* %11, i64 0
39  %12 = bitcast i32* %11 to i8*
40  store i32 0, i32* %0
41  %13 = bitcast i32* %0 to i8*
42  call void (i8*, i8*, i32, ...) ↩
        ↪ @_Z30shared_memory_sequential_storePvS_iz(i8* %12, i8* %13, ↩
        ↪ i32 1, i64 0)
43  %14 = load i32*, i32** %a, align 8
44  %arrayidx2 = getelementptr inbounds i32, i32* %14, i64 1
45  %15 = bitcast i32* %14 to i8*
46  store i32 1, i32* %1
47  %16 = bitcast i32* %1 to i8*
48  call void (i8*, i8*, i32, ...) ↩
        ↪ @_Z30shared_memory_sequential_storePvS_iz(i8* %15, i8* %16, ↩
        ↪ i32 1, i64 1)
49  %17 = load i32*, i32** %a, align 8
50  %arrayidx3 = getelementptr inbounds i32, i32* %17, i64 2
51  %18 = bitcast i32* %17 to i8*
52  store i32 2, i32* %2
53  %19 = bitcast i32* %2 to i8*
54  call void (i8*, i8*, i32, ...) ↩
        ↪ @_Z30shared_memory_sequential_storePvS_iz(i8* %18, i8* %19, ↩
        ↪ i32 1, i64 2)
55  %20 = load i32*, i32** %a, align 8
56  %arrayidx4 = getelementptr inbounds i32, i32* %20, i64 3
57  %21 = bitcast i32* %20 to i8*
58  store i32 3, i32* %3
59  %22 = bitcast i32* %3 to i8*
60  call void (i8*, i8*, i32, ...) ↩
        ↪ @_Z30shared_memory_sequential_storePvS_iz(i8* %21, i8* %22, ↩
        ↪ i32 1, i64 3)
```

Listing 8: Transformed IR

Step 4: Microtask Execution (I)

```
36  call void (%struct.ident_t*, i32, void (i32*, i32*, ...)*, ...) ↩
        ↪ @__kmpc_fork_call(%struct.ident_t* @1, i32 2, void (i32*, i32*, ...)* ↩
        ↪ bitcast (void (i32*, i32*, i32**, i32**)* @.omp_outlined. to void ↩
        ↪ (i32*, i32*, ...)*), i32** %b, i32** %a)
```

Listing 9: Original IR

```
61  call void @.omp_outlined.(i32* null, i32* null, i32** %b, i32** %a)
62  call void @_Z11mpi_barrierv()
```

Listing 10: Transformed IR

Step 4: Microtask Execution (II)

```
82  store i32 0, i32* %.omp.lb, align 4
83  store i32 3, i32* %.omp.ub, align 4
84  store i32 1, i32* %.omp.stride, align 4
85  store i32 0, i32* %.omp.is_last, align 4
86  %2 = load i32*, i32** %.global_tid..addr, align 8
87  %3 = load i32, i32* %2, align 4
88  call void @__kmpc_for_static_init_4(%struct.ident_t* @0, i32 %3, i32 34, i32* ↩
        ↪ %.omp.is_last, i32* %.omp.lb, i32* %.omp.ub, i32* %.omp.stride, i32 1, ↩
        ↪ i32 1)
```

Listing 11: Original IR

```
130  store i32 0, i32* %.omp.lb, align 4
131  store i32 3, i32* %.omp.ub, align 4
132  store i32 1, i32* %.omp.stride, align 4
133  store i32 0, i32* %.omp.is_last, align 4
134  %2 = load i32*, i32** %.global_tid..addr, align 8
135  %3 = load i32, i32* %2, align 4
136  call void @_Z26modify_parallel_for_boundsPiS_i(i32* %.omp.lb, i32* %.omp.ub, ↩
         ↪ i32 1)
```

Listing 12: Transformed IR

Step 5: Memory Interaction

```
37  %6 = load i32*, i32** %b, align 8
38  %arrayidx5 = getelementptr inbounds i32, i32* %6, i64 0
39  %7 = load i32, i32* %arrayidx5, align 4
40  %8 = load i32*, i32** %b, align 8
41  %arrayidx6 = getelementptr inbounds i32, i32* %8, i64 1
42  %9 = load i32, i32* %arrayidx6, align 4
43  %10 = load i32*, i32** %b, align 8
44  %arrayidx7 = getelementptr inbounds i32, i32* %10, i64 2
45  %11 = load i32, i32* %arrayidx7, align 4
46  %12 = load i32*, i32** %b, align 8
47  %arrayidx8 = getelementptr inbounds i32, i32* %12, i64 3
48  %13 = load i32, i32* %arrayidx8, align 4
49  %call9 = call i32 (i8*, ...) @printf(i8* getelementptr ↩
        ↪ inbounds ([15 x i8], [15 x i8]* @.str.1, i64 0, i64 ↩
        ↪ 0), i32 %7, i32 %9, i32 %11, i32 %13)
```

Listing 13: Original IR

```
81  call void (i8*, i8*, i32, ...) ↩
        ↪ @_Z29shared_memory_sequential_loadPvS_iz(i8* %34, i8* %35, i32 ↩
        ↪ 1, i64 2)
82  %36 = bitcast i8* %35 to i32*
83  %37 = load i32, i32* %36
84  %38 = load i32*, i32** %b, align 8
85  %arrayidx8 = getelementptr inbounds i32, i32* %38, i64 3
86  %39 = bitcast i32* %38 to i8*
87  %40 = bitcast i32* %7 to i8*
88  call void (i8*, i8*, i32, ...) ↩
        ↪ @_Z29shared_memory_sequential_loadPvS_iz(i8* %39, i8* %40, i32 ↩
        ↪ 1, i64 3)
89  %41 = bitcast i8* %40 to i32*
90  %42 = load i32, i32* %41
91  %call9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([15 ↩
        ↪ x i8], [15 x i8]* @.str.1, i64 0, i64 0), i32 %27, i32 %32, ↩
        ↪ i32 %37, i32 %42)
```

Listing 14: Transformed IR

Step 6: MPI Finalisation

```
50  %14 = load i32*, i32** %a, align 8
51  %15 = bitcast i32* %14 to i8*
52  call void @free(i8* %15) #4
53  %16 = load i32*, i32** %b, align 8
54  %17 = bitcast i32* %16 to i8*
55  call void @free(i8* %17) #4
56  ret i32 0
```

Listing 15: Original IR

```
92  %43 = load i32*, i32** %a, align 8
93  %44 = bitcast i32* %43 to i8*
94  call void @_Z18shared_memory_freePv(i8* %44)
95  %45 = load i32*, i32** %b, align 8
96  %46 = bitcast i32* %45 to i8*
97  call void @_Z18shared_memory_freePv(i8* %46)
98  store i32 0, i32* %8
99  br label %cato_finalize
100
101 cato_finalize:
102 call void @_Z13cato_finalizev()
103 %47 = load i32, i32* %8
104 ret i32 %47
```

Listing 16: Transformed IR

LLVM Function Pass[LLV18]


```
#include "llvm/Pass.h"
#include "llvm/IR/Function.h"
#include "llvm/Support/raw_ostream.h"

using namespace llvm;
namespace
{
    struct printFunction : public FunctionPass
    {
        static char ID;
        printFunction() : FunctionPass(ID) {}

        bool runOnFunction(Function &F) override
        {
            errs() << "Hello: ";
            errs().write_escaped(F.getName()) << '\n';
            return false;
        }
    };
}

char printFunction::ID = 0;
static RegisterPass<printFunction> X("hello", "Hello World Pass", false, false);
```

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

Extended Weak Scaling

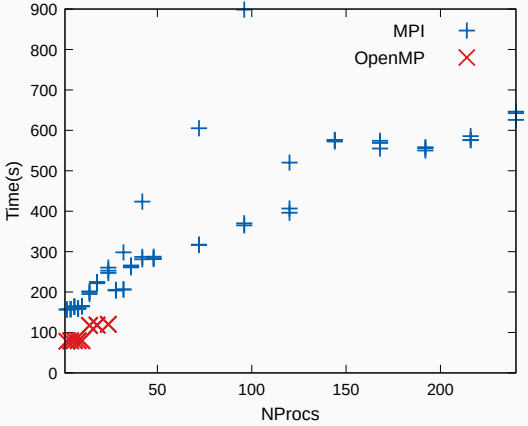


Figure 7: Using 10 nodes for weak scaling [Jam18]

Active Target Synchronisation (I)

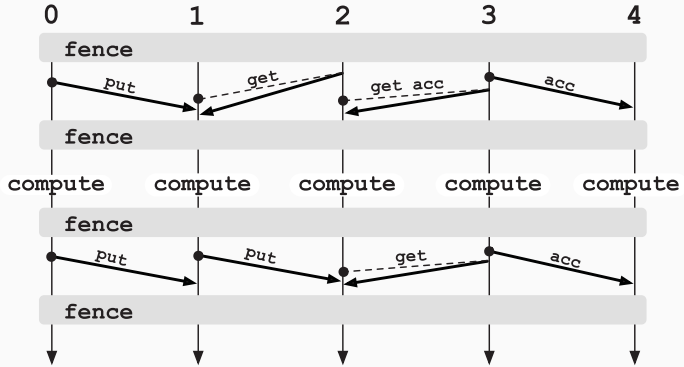


Figure 8: Collective Fence Synchronisation [HDT+15]

Active Target Synchronisation (II)

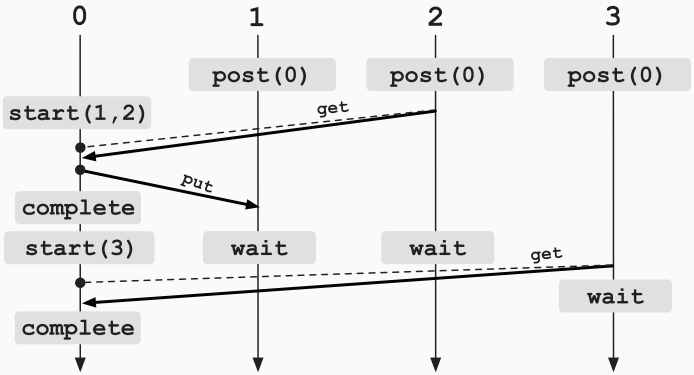
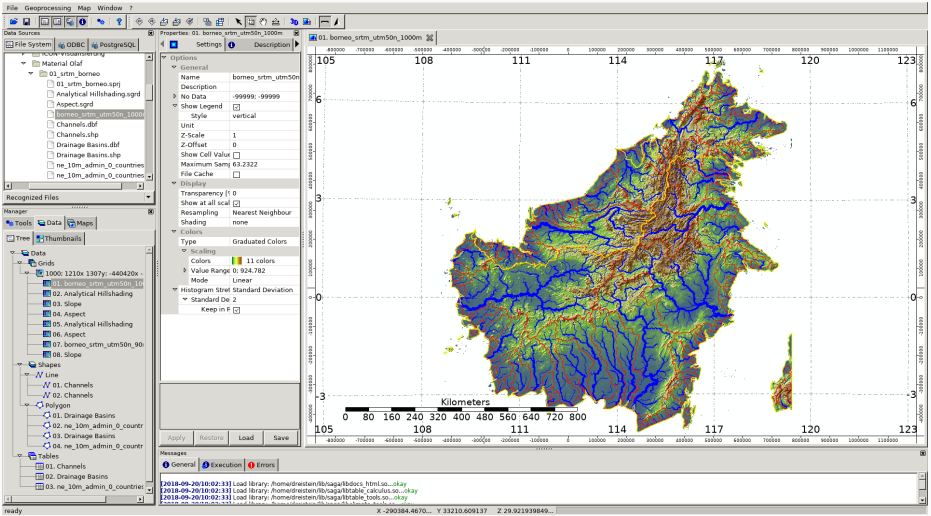
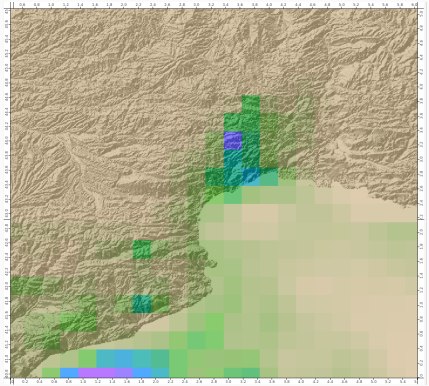


Figure 9: GATS Synchronisation [HDT+15]

Project-Example - SAGA-GIS (I)

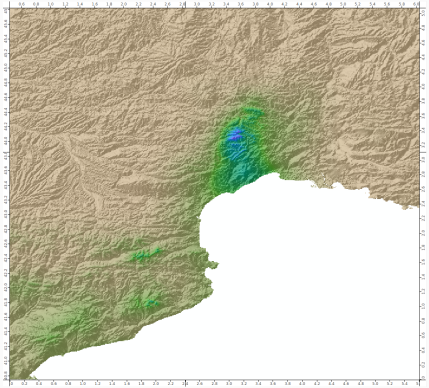


Project-Example - SAGA-GIS (II)



Downscaling

27km→1km



Project-Example - SAGA-GIS (III)

```
1  for(int y=0; y<Get_NY() && Set_Progress(y); y++)
2  {
3  #pragma omp parallel for
4  for(int x=0; x<Get_NX(); x++)
5  {
6    int Belt  = 0;  // no data
7    if( !pL->is_NoData(x, y) )
8    {
9      [...]
10   }
11   pBelt->Set_Value(x, y, Belt);
12 }
```

Listing 17: "Thermic Belt Classification"

Total occurrences in all SAGA-GIS tools:

322× **#pragma** omp parallel **for**

↪ [private(...)] [reduction(...)]

1× **#pragma** omp critical