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
1 Introduction

2 CATO

3 Roadmap

4 Summary
**Trend - Computer Science**

1. **Increasing core count of manycore architecture**
   - → **peak performance** node
   - Parallelisation techniques on shared memory needed
     - **First choice:** OpenMP

2. **Increasing gap between memory and CPU performance in manycore architecture**
   - → **memory** / **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:** $\sim \frac{30 \text{ minutes}}{1 \text{ simulated year}}$
- **I/O:** $\sim \frac{1 \text{ TB}}{1 \text{ simulated year}}$
Example 🌟 TBNT (III)

Setup Northern Gulf of Mexico:

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

_Figure:_ by F. Große (pers. comm.)
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 → 1km
Total occurrences in all SAGA-GIS tools:

\[ 322 \times \text{#pragma omp parallel for} \]
\[ \leftrightarrow \text{[private(...)] [reduction(...)]} \]

\[ 1 \times \text{#pragma omp critical} \]
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
- System property
- Implementations:
  - JUMP[1]
  - TreadMarks[3]

- No code changes
- No user interaction
- Easy to use

- Poor performance[20]
- Poor scalability[20]
- Neglected development
## Message Passing Interface - MPI

<table>
<thead>
<tr>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message passing</td>
<td>Additional learning effort</td>
</tr>
<tr>
<td>Processes</td>
<td>Buffer management</td>
</tr>
<tr>
<td>Explicit communication</td>
<td>Re-implementation necessary</td>
</tr>
<tr>
<td>Active development</td>
<td>RDMA</td>
</tr>
</tbody>
</table>
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

- One-sided communication: *origin* → *target*

- Synchronisation through epochs:
  - Active target synchronisation
  - Generalized active target synchronisation
  - Passive target synchronisation

- Hardware-support through RDMA

*Figure:* RMA Window[6]
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

![Diagram of pass integration](image)

**Figure:** Pass integration, based on [22]

**Listing 2:** Linking an LLVM module pass

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

1

Squar, Blesel, Jammer, Kuhn, Ludwig

CATO: Compiler Assisted S2S Transformation of OpenMP kernels
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)
A new Approach

Example 🕯 Intermediate Representation Language (II)

```assembly
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
}
```

Listing 4: Corresponding IR code (extract)

Squar, Blesel, Jammer, Kuhn, Ludwig CATO - Compiler Assisted S2S Transformation of OpenMP kernels 27/47
1 Introduction
2 CATO
3 Roadmap
4 Summary
Our solution: CATO

Based on:
- One-sided MPI-3 communication
- LLVM compiler infrastructure

CATO specification:
- Requirements
- CATO Workflow
- Memory Handling
The idea of CATO

Ambitions

CATO: Compiler Assisted s2s Transformation of OpenMP 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
The idea of CATO

**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

Original application

Run CATO

Generate IR code

Identify significant memory allocations

Identify communication patterns

Select and adapt replacement template

Perform replacement on IR

The idea of CATO
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

- 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

**Figure:** Example output of partdiff
Preliminary Results: Scaling [17]

**Figure: Strong scaling**

**Figure: Weak scaling**
1 Introduction

2 CATO

3 Roadmap

4 Summary
Excerpt of considered OpenMP Pragmas

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

\(^1\text{colour key: not planned, planned, work in progress}\)
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

⇒ Tune pass through determined environment^2 configuration

... hardware level
- Micro-benchmarks
- Adapted roofline model[23]

^2hardware+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
- ....
1 Introduction

2 CATO

3 Roadmap

4 Summary
Summary

- Aim for an easy usage
- Communication pattern $\rightarrow$ communication 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)
References


