OMP2MPI: Automatic MPI code generation from OpenMP programs

Seminar Supercomputer: Forschung und Innovation

Fabian Holst

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2023-01-10

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

Gliederung

Ich möchte, dass mein Code schneller läuft.

Ich möchte mir keine Arbeit machen.

OpenMP

- Einfach(er) programmierbar
- Parallelisierung ist transparent durch #pragma an Schleifen
- Shared Memory: Alles passiert auf einem Knoten
- Beschränkung durch Arbeitsspeicher eines Rechenknotens

OpenMP

- `#include <omp.h>`
- `#pragma omp parallel`
  `printf("Hello, world.\ n");`
- `export OMP_NUM_THREADS=16`
- `gcc -fopenmp example.c`

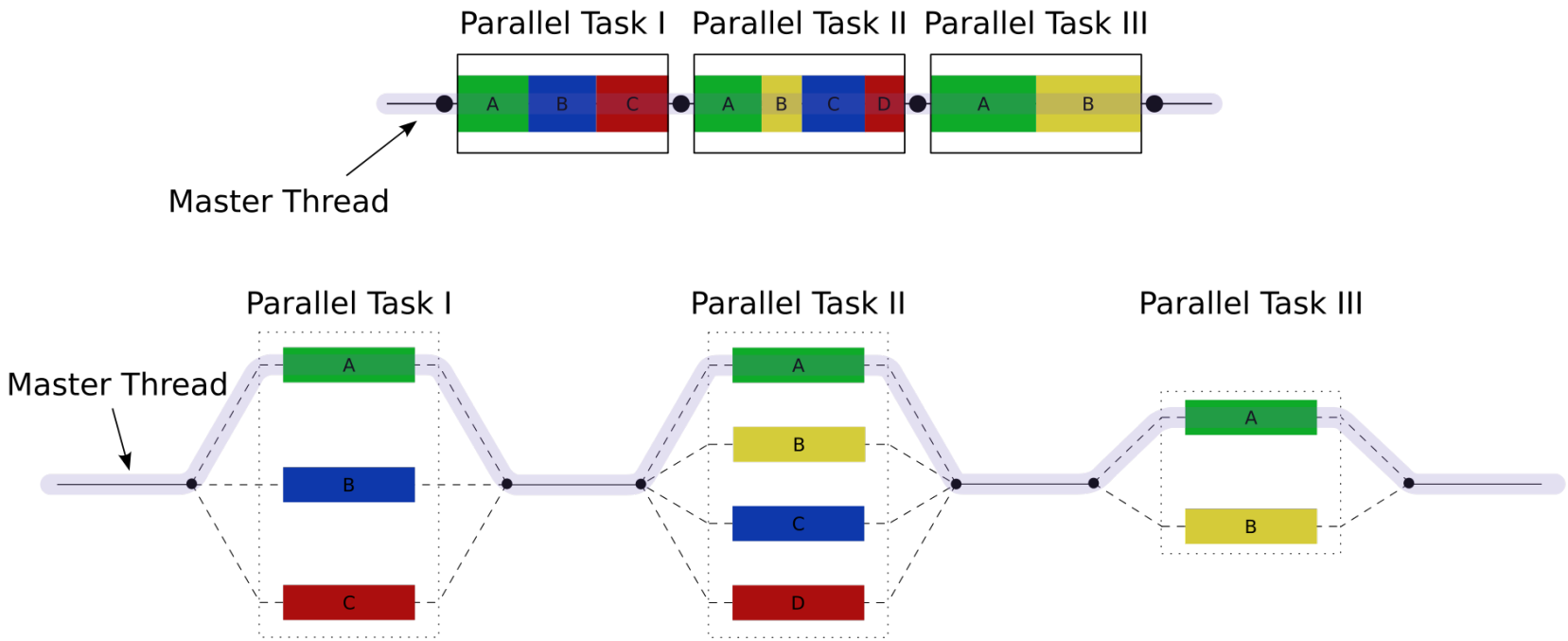


Abbildung: Fork-Join Model OpenMP [8]

MPI

- Verteilung des Codes über Knoten hinweg möglich
- Keine Einschränkung durch Memory eines einzelnen Knotens
- Programmcode schwieriger nachzuvollziehen: Fachliches und Parallelisierung gemischt

MPI

- `#include <mpi.h>`
- `MPI_Init(&argc, &argv);`
- `MPI_Send(buf, 256, MPI_CHAR, other_rank,0, MPI_COMM_WORLD);`
- `MPI_Recv(buf, 256, MPI_CHAR, other_rank,0, MPI_COMM_WORLD,MPI_STATUS_IGNORE);`
- `MPI_Finalize();`
- `mpicc example.c`
- `mpiexec -n 4`

OpenMp vs MPI

Tradeoff: Codewritability/readability vs. Parallelisierungsgrad
Beides zusammen wäre aber doch gut

Distributed Memory

- Verteile das OpenMP Programm auf mehrere Knoten mit einer Abstraktionsschicht die einen zusammenhängenden Speicherbereich simuliert
- Kein explizites Message-passing, alles in der Abstraktionsschicht versteckt
- Erlaubt kein finetuning
- Eingeschränkter einsatzbereich
- Benötigt Runtimeenvironment

Source to Source Compiler

- Übersetze OpenMP Programm in MPI Programm
- Lesbarkeit von OpenMP, parallelisierbarkeit von MPI
- Programmierer muss sich nicht um Message Passing kümmern

Das Paper

OMP2MPI: Automatic MPI code generation from OpenMP programs

Albert Saà-Garriga
Universitat Autonòma de Barcelona
Edifici Q,Campus de la UAB
Bellaterra, Spain
albert.saa@uab.cat

David Castells-Rufas
Universitat Autonòma de Barcelona
Edifici Q,Campus de la UAB
Bellaterra, Spain
david.castells@uab.cat

Jordi Carrabina
Universitat Autonòma de Barcelona
Edifici Q,Campus de la UAB
Bellaterra, Spain
jordi.carrabina@uab.cat

Abbildung: Screenshot des Papertitles [1]

- 6 Zitierungen auf Semantic Scholar [5], 13 auf Google Scholar [6]

Lösungsidee des Papers

- Übersetze OpenMP Direktiven in äquivalente MPI Direktiven
- Resultat soll lesbarer MPI-Code sein, der von Experten veränderbar / tunebar bleibt
- Als C++ Plugin für das Mercurium Compilerframework, per Konfiguration aktivierbar
- Suche nach OpenMP Pragmas die mit `target mp` versehen sind

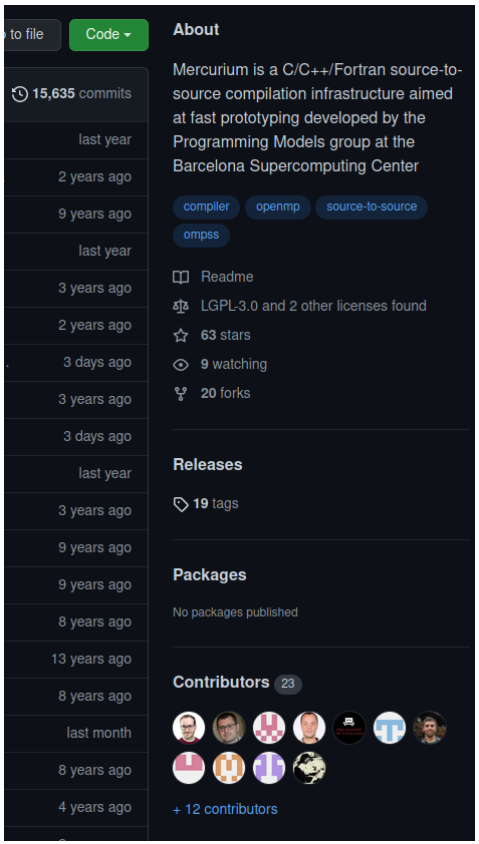
Code des S2S Compilers

- Verfügbar auf Github [4]
- Insgesamt 13 Commits in 2 1/2 Monaten, alle vom Account des ersten Paper Autors Albert Saà-Garriga
- Code des Compilerplugins in `trans/trans_phase.cpp` (5268 Zeilen)
- Im Repo sind .svn Ordner, vielleicht wurde intern mit SVN gearbeitet

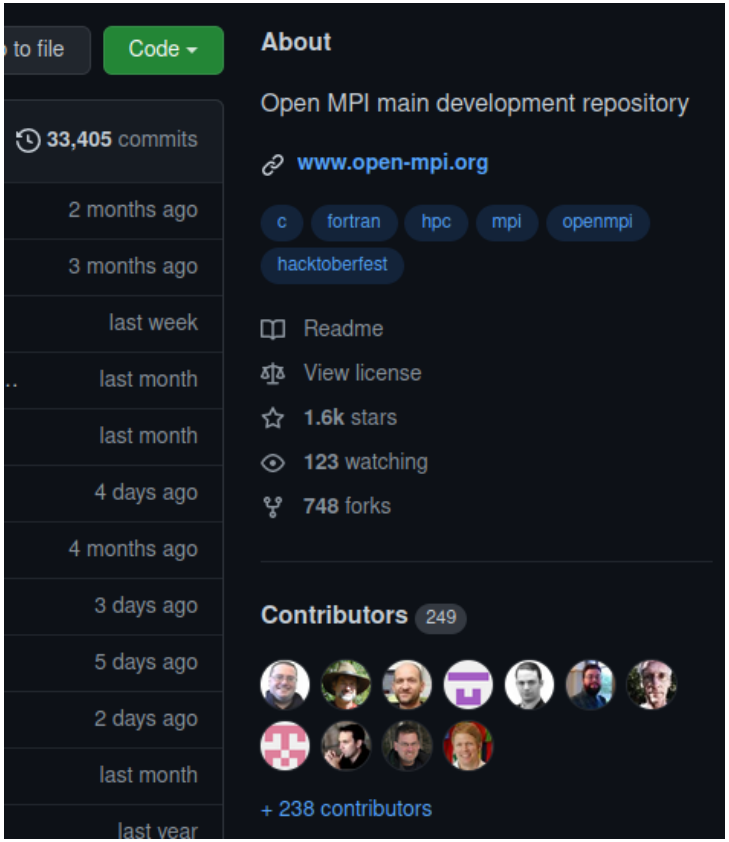
Das Mercurium Compilerframework

- C/C++/Fortran source-to-source compilation infrastructure
- Ausgerichtet auf Prototyping mit schnellen Entwicklungsiterationen
- Entwickelt von der Programming Models Gruppe am Barcelona Supercomputing Center
- Plugins können als Kompilierphasen geladen werden
- Verfügbar auf Github unter LGPL-Lizenz [3]

Das Mercurium Compilerframework



(a) Screenshot vom Mercurium Github-Projekt. 63 stars, 9 watching, 20 forks, 23 contributors [3]



(b) Screenshot vom OpenMPI Github-Projekt. 1.6k stars, 123 watching, 748 forks, 249 contributors [7]

Mehr zum Plugin

- "Code transformations are implemented to the source code which implies that there is no need to know or modify the internal syntactic representation of the compiler."[1]
- "valid by construction"[1]
- Einbindung in die Stufen des Compilers
- Aufsetzen auf den Abstract Syntax Tree(AST)
  - Repräsentation der Quelltext Struktur
  - Liste der Symbole
  - Kontext

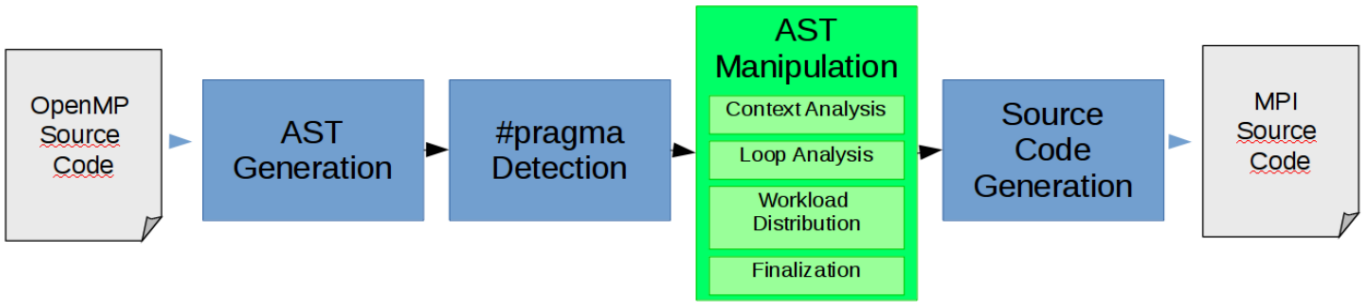


Abbildung: OMP2MPI in der Mercurium Pipeline [1]

Zielbild

Rank0 als Koordinator
Shared Memory Zugriffe in OMP $\rightarrow$ Datenaustäusche in MPI
Shared Variables über rank0 kommuniziert

Zielbild

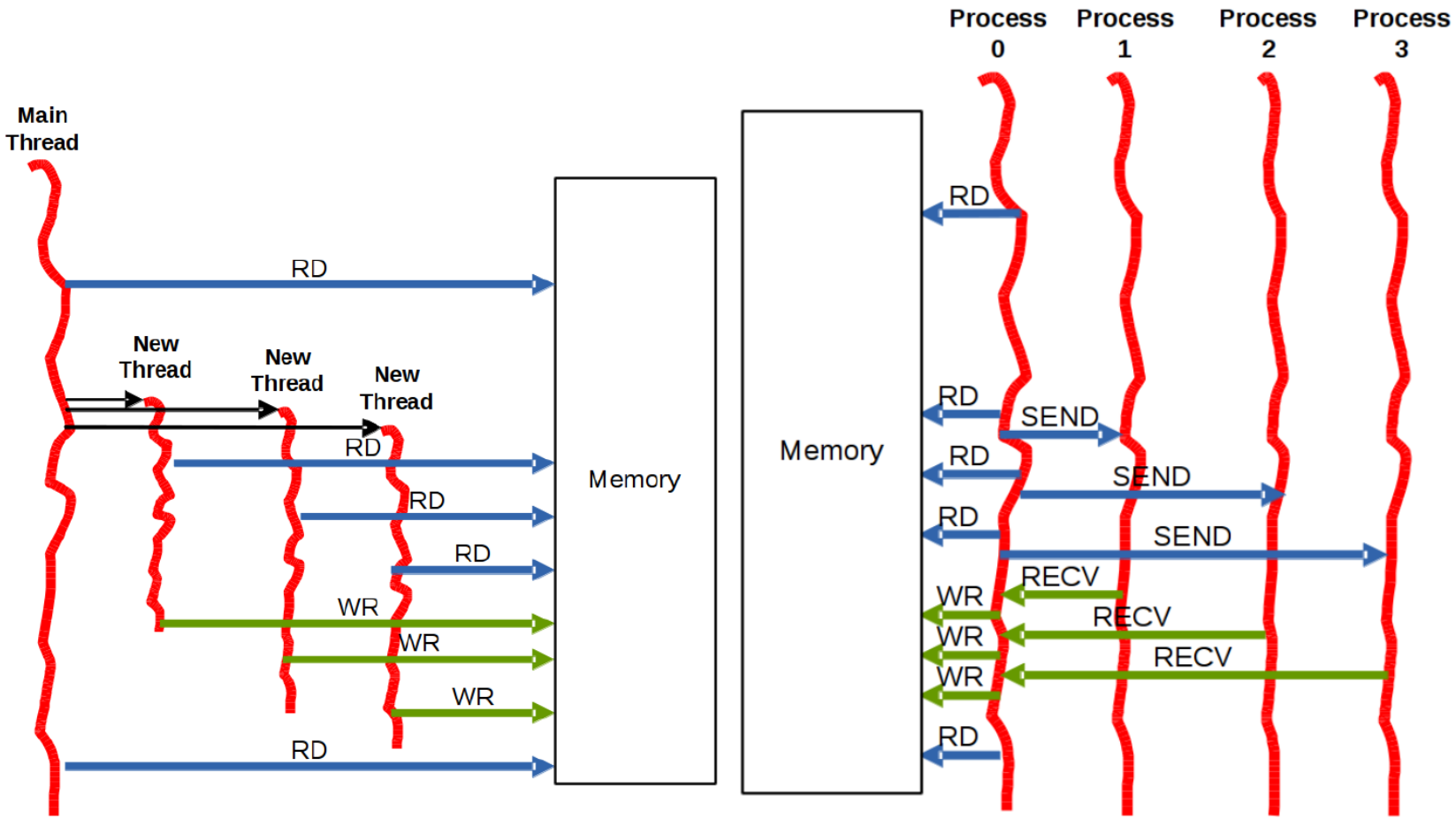


Abbildung: Datenaustausch OMP vs. MPI [1]

Besteht aus 4 Schritten:

1. Context Analysis
2. Loop Analysis
3. Workload Distribution
4. Finalize

Context Analysis

Variablen:

- In MPI sind erstmal alle Variablen process private
- In OpenMP als private markierte Variablen also kein Problem
- Was ist mit shared variablen in OpenMP? Hier muss etwas getan werden

Schleifen:

- Suche nach parallelisierten, verschachtelten Schleifen
- Vermeide MPI Initialisierungen für die innere Schleife mehrfach zu tätigen
  - Ziehe diese außerhalb der äußeren Schleife.

Context Analysis: Shared Variablen

- Unterscheide shared Variables nach: Die Variable wird in der Schleife ...
  - IN: nur gelesen -> Muss am Anfang der Schleife ausgetauscht werden
  - OUT: nur geschrieben -> Muss am Ende der Schleife ausgetauscht werden
  - INOUT: sowohl gelesen als auch geschrieben -> Muss am Anfang und am Ende der Schleife ausgetauscht werden

Loop Analysis

- Untersuche Schleifenindices:
  - Startwert
  - Endwert
  - Inkrement
  - Abbruchbedingungen
- Manche Schleifen können nicht parallelisiert werden, hier wird dann alles im Masterprozess erledigt
  - "complex not linear increments on iterator"[1]
  - "multiple cases on condition"[1]

Workload Distribution

- Iterationen einer Schleife auf mehrere Worker aufteilen
- Iterationen einzeln verteilen -> könnte großen Overhead verursachen
- Stattdessen IterationAnzahl/numberOfRanks/10 als Paketgröße

Workload Distribution: Static / Dynamic

- Unterscheidung ob im OpenMP Pragma static oder guided
- Static:
  - Round robbin an die Worker
  - Nächster Batch an Iterationen wird erst verteilt wenn alle Worker fertig sind
- Dynamic:
  - Freie Worker bekommen neue Iterationen

Workload Distribution: Static / Dynamic

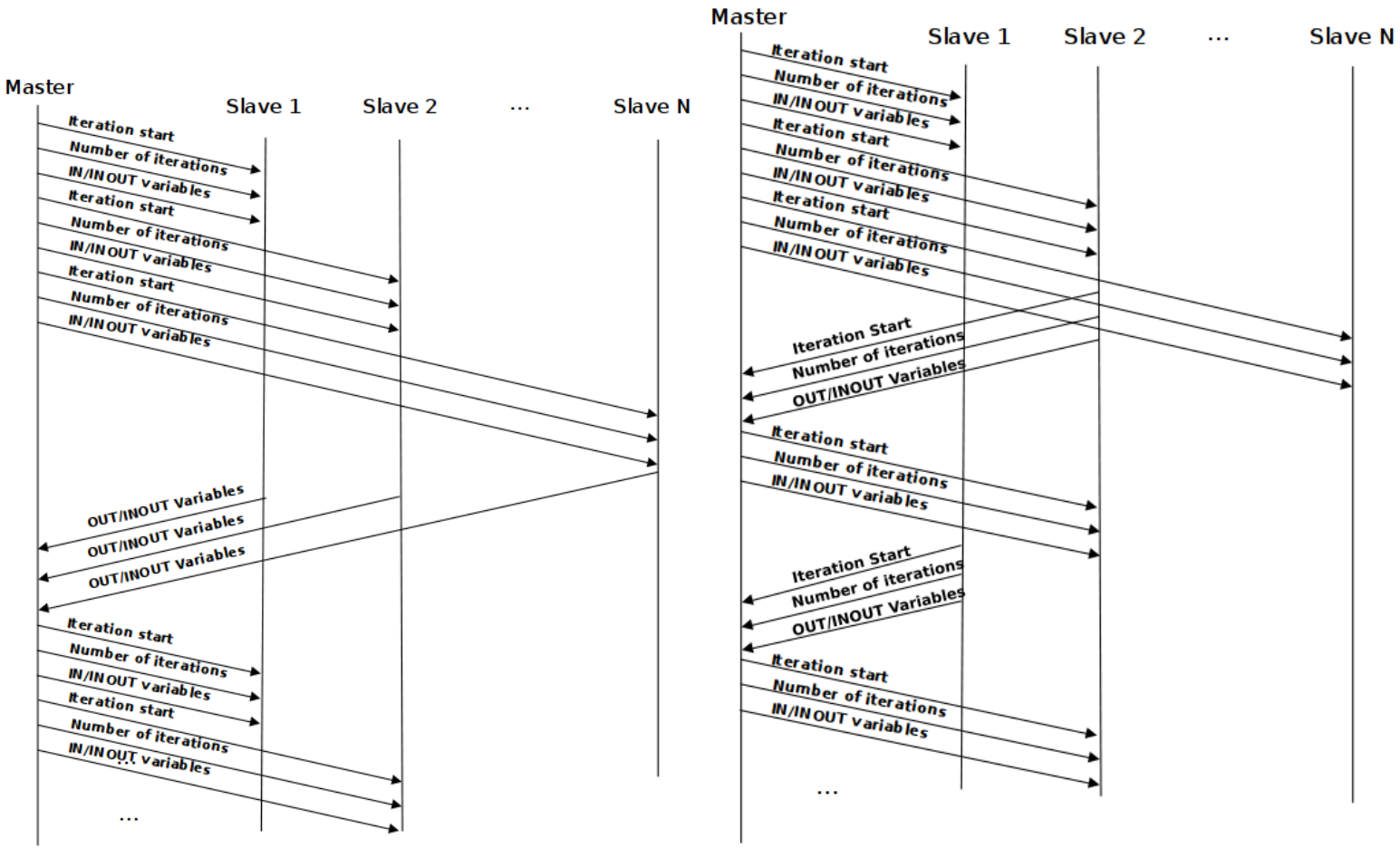


Abbildung: Static vs Dynamic distribution [1]

Finalize

- Was ist mit dem Code außerhalb von Schleifen?
- Alles was nicht parallelisiert wurde, wird in if(rank==0){} Blöcke gewrapped damit nur rank0 sie ausführt
- Außerdem werden die nötigen MPI_Finalize aufrufe hinzugefügt.

Leistungsevaluation

- Tests "were executed in 64 CPU's E7-4800 with 2.40 GHz(Bullion quadri module)", später sprechen sie von "64 cores"[1]
- 64 Cores + 2.4Ghz (Turbotakt) -> E7-4830?

Intel® Xeon® Processor E7-2870	Discontinued	Q2'11	10	2.80 GHz	2.40 GHz	30 MB Intel® Smart Cache
Intel® Xeon® Processor E7-4807	Discontinued	Q2'11	6		1.86 GHz	18 MB Intel® Smart Cache
Intel® Xeon® Processor E7-4820	Discontinued	Q2'11	8	2.27 GHz	2.00 GHz	18 MB Intel® Smart Cache
Intel® Xeon® Processor E7-4830	Discontinued	Q2'11	8	2.40 GHz	2.13 GHz	24 MB Intel® Smart Cache
Intel® Xeon® Processor E7-4850	Discontinued	Q2'11	10	2.40 GHz	2.00 GHz	24 MB Intel® Smart Cache
Intel® Xeon® Processor E7-4860	Discontinued	Q2'11	10	2.67 GHz	2.26 GHz	24 MB Intel® Smart Cache
Intel® Xeon® Processor E7-4870	Discontinued	Q2'11	10	2.80 GHz	2.40 GHz	30 MB Intel® Smart Cache
Intel® Xeon® Processor E7-8830	Discontinued	Q2'11	8	2.40 GHz	2.13 GHz	24 MB Intel® Smart Cache

Abbildung: Prozessoren der E7 Familie [2]

Leistungsevaluation

- Test auf Polybench benchmark

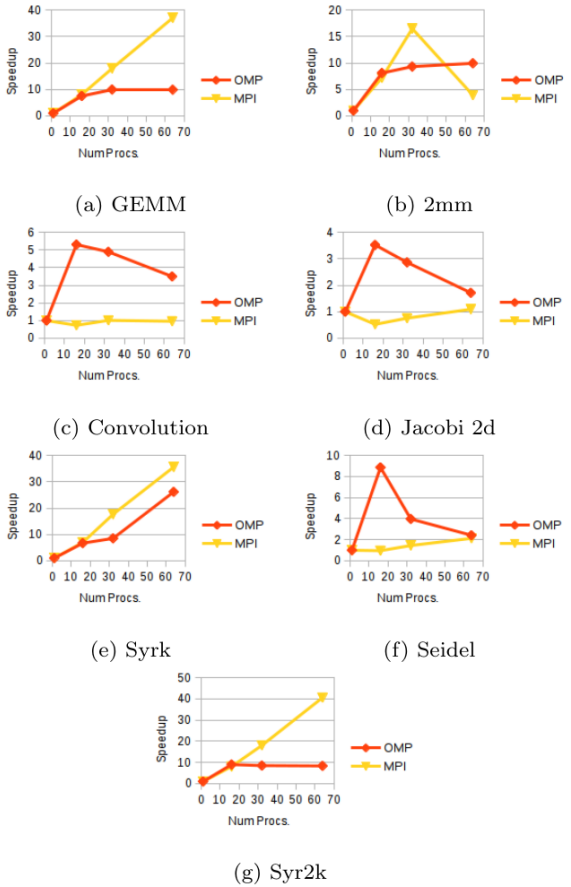


Abbildung: Ergebnisse [1]

PolyBench/C

- GEMM = Matrix-multiply C=alpha.A.B+beta.C
- 2mm = 2 Matrix Multiplications (D=A.B; E=C.D)
- Convolution = keine Entsprechung in der Doku, es gibt correlation
- Jacobi 2d = 2-D Jacobi stencil computation
- Syrk = symmetric rank-k update
- Seidel = 2-D Seidel stencil computation
- Syr2k = Symmetric rank-2k update

[9]

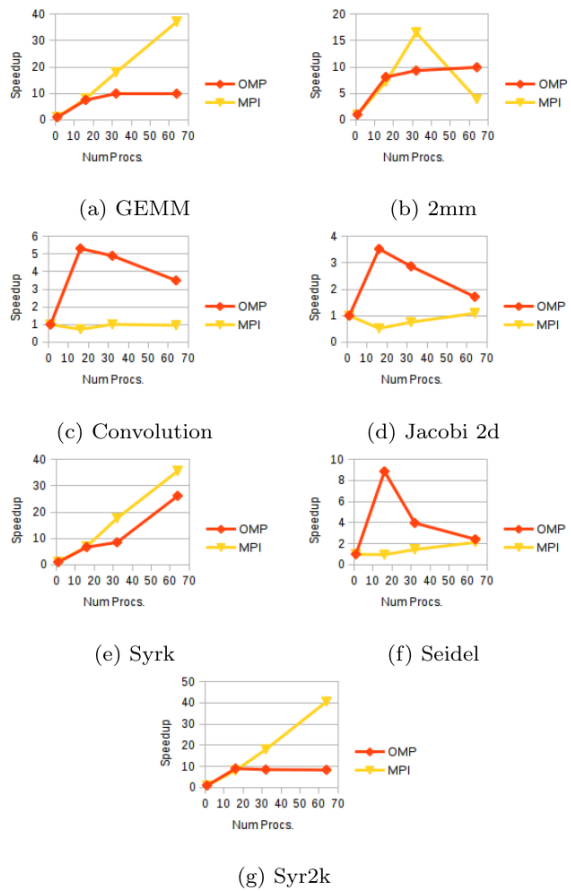
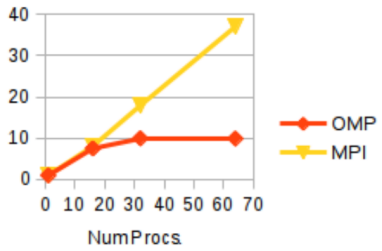
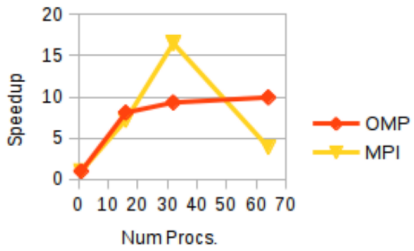


Abbildung: Ergebnisse [1]

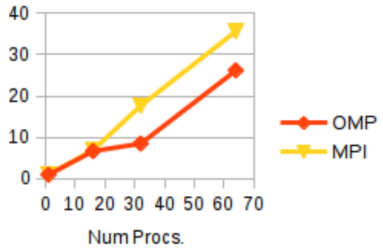
Leistungsevaluation



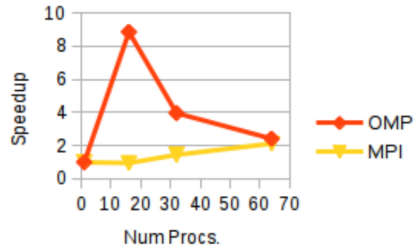
(a) GEMM



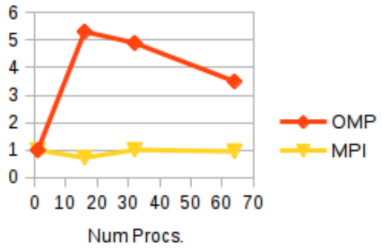
(b) 2mm



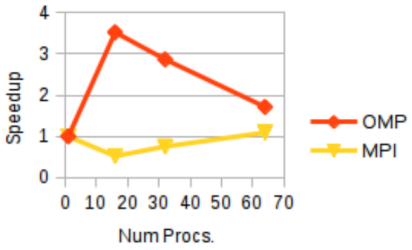
(e) Syrk



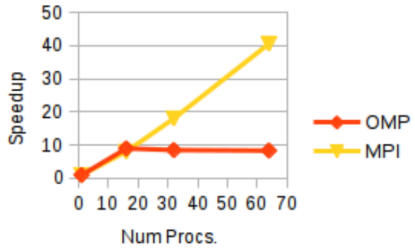
(f) Seidel



(c) Convolution



(d) Jacobi 2d



(g) Syr2k

Abbildung: Ergebnisse [1]

PolyBench/C

- PolyBench/C hat 30 verschiedene Tests
- Was ist mit dem Rest?
- Keine Erwähnung von Wiederholung der Testläufe
- Weak scaling? Polybench hat nicht direkt eine feingranulare Einstellung der Problemgröße, aber es gibt die Flags MINI_DATASET, SMALL_DATASET, STANDARD_DATASET, LARGE_DATASET, EXTRALARGE_DATASET
- Dataset options orientieren sich an memory requirements, 16KB, 128KB, 1MB, 25MB, 120MB

[9]

Future Work

- Mehr Indizierungen von Schleifen übersetzen können
- Es werden nur Point to Point Kommunikationen von MPI benutzt -> Auch Broadcast / AllReduce von MPI unterstützen

Eigene Einschätzung

- Wirkt sehr Prototypenhaft
- Nur einfache Konstrukte sind übersetzbar
- Leistungsevaluation recht unaussagekräftig
- Austausch der Daten über rank0 nicht ideal

Zusammenfassung

- Idee Einfachheit und Mächtigkeit der Programmierparadigma zu kombinieren ist gut
- Implementation als Plugin vereinfacht Entwicklung des S2S-Compilers
- Performance-Verbesserung fraglich, Verteilung auf mehrere Compute-Nodes nicht eruiert

Literatur I

[1] Albert Saà-Garriga; David Castells-Rufas; Jordi Carrabina. "OMP2MPI: Automatic MPI code generation from OpenMP programs". In: (Juni 2015).

[2] *Intel® Xeon® Processor E7 Family*. URL: https://ark.intel.com/content/www/us/en/ark/products/series/59139/intel-xeon-processor-e7-family.html.

[3] *Mercurium Github Page*. 19. Dez. 2022. URL: https://github.com/bsc-pm/mcxx.

[4] *OMP2MPI*. 2. Jan. 2023. URL: https://github.com/sdruix/OMP2MPI.

Literatur II

[5] *OMP2MPI: Automatic MPI code generation from OpenMP programs overview page*. 19. Dez. 2022. URL: https://www.semanticscholar.org/paper/OMP2MPI%3A-Automatic-MPI-code-generation-from-OpenMP-Sa%C3%A0-Garriga-Castells-Rufas/daa701c4d6aee1a817fa8e04ca5e8b439feda8ec.

[6] *OMP2MPI: Automatic MPI code generation from OpenMP programs searched on google scholar*. 19. Dez. 2022. URL: https://scholar.google.com/scholar?cites=18144075142242873184&as_sdt=2005&sciodt=0,5&hl=de.

[7] *Open MPI Github Page*. 19. Dez. 2022. URL: https://github.com/open-mpi/ompi.

Literatur III

[8] *OpenMP Fork join visualization*. URL: https://en.wikipedia.org/wiki/File:Fork_join.svg.

[9] *PolyBench/C the Polyhedral Benchmark suite*. URL: http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/.