LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Dr. Karl Fürlinger
Lehrstuhl für Kommunikationssysteme und Systemprogrammierung

Chameleon - Eine Taskbasierte Programmierumgebung zur Entwicklung reaktiver HPC Anwendungen

Web:
http://www.chameleon-hpc.org/

Projektpartner:

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

RWTH AACHEN UNIVERSITY

TUM

Das Chameleon Projekt – Überblick

- Chameleon
  - Projekt im 5. BMBF HPC call
  - Laufzeit: 1.4.2017 – 31.3.2020

- Projektpartner
  - **LMU München**,
    LFE für Kommunikationssysteme und Systemprogrammierung
  - **RWTH Aachen**,
    Lehrstuhl für Hochleistungsrechnen
  - **TU München**,
    Professur für Hardwarenahe Algorithmik

- Ziele:
  - Entwicklung einer **taskbasierten Programmierumgebung**, basierend auf und mit Erweiterungen von **MPI** und **OpenMP**
  - Unterstützung für **Reaktivität**, dh. Anwendung wird in die Lage versetzt auf dynamische HW Veränderungen zu reagieren

Motivation – Dynamische Variabilität

- Dynamische Variabilität steigt
  - ZB. durch Intel Turbo Modus, Power Capping
  - Komplexe Speichersysteme
  - HW Fehlererkennung und Behandlung

Our Dynamic Future

Pete Beckman | Argonne National Laboratory an

Last month, as I tossed my bags in a renta
airport, I noticed that the car was particu
was quite surprised, however, when I drov
first stop sign, and the car suddenly died.
I had run out of gas or turned off the ignition. H
soon as I took my foot off the brake pedal, the e
ed itself back up. I pushed on the accelerator, a
jumped forward. Over the next couple of days,
this advanced fuel-saving feature, trying to und
der what circumstances the car's algorithms wou
could save gas by temporarily shutting off and h
I could jump forward after moving my foot fro
accelerator as the car automatically started itself
adjusted the throttle.

Dynamic power management is everywhere
and mobile phones to home heating and cooling.
key technology changes making advanced pow
ment possible for your automobile is the shift to

Variation Among Processors Under Turbo Boost in HPC Systems

Bilge Acun, Phil Miller, Laxmikant V. Kale
Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL, 61801
{acun2, mille121, kale}@illinois.edu

Abstract

The design and manufacture of present-day CPUs causes
inherent variation in supercomputer architectures such as
variation in power and temperature of the chips. The vari-
ation also manifests itself as frequency differences among
processors under Turbo Boost dynamic overclocking. This
variation can lead to unpredictable and suboptimal perfor-
mance in tightly coupled HPC applications. In this study,
we use compute-intensive kernels and applications to analyze
the variation among processors in four top supercomputers:
Edison, Cab, Stampede, and Blue Waters. We observe that
there is an execution time difference of up to 16% among
processors on the Turbo Boost-enabled supercomputers: Edi-
son, Cab, Stampede. There is less than 1% variation on Blue
Waters, which does not have a dynamic overclocking feature.
We analyze measurements from temperature and power in-
strumentation and find that intrinsic differences in the chips'
power efficiency is the culprit behind the frequency variation.
Moreover, we analyze potential solutions such as disabling
Turbo Boost, leaving idle cores and replacing slow chips to
mitigate the variation. We also propose a speed-aware dy-
namic task redistribution (load balancing) algorithm to reduce
the negative effects of performance variation. Our speed-
aware load balancing algorithm improves the performance up
to 18% compared to no load balancing performance and 6%
better than the non-speed aware counterpart.

2016 IEEE International Parallel and Distributed Processing Symposium Workshops

Mitigating Processor Variation through Dynamic Load Balancing

Bilge Acun, Laxmikant V. Kale
University of Illinois at Urbana-Champaign, Department of Computer Science
{acun2, kale}@illinois.edu

***Abstract*—There can be performance variation among same-model processors in large scale clusters, and supercomputers that are caused by power, and temperature variations among the processors. These variations manifest itself as frequency difference of the processors under dynamic overclocking, such as Turbo Boost. Different-model processors also create an inherent variation when used in same cluster. For some tightly coupled HPC applications even one slow processor in the critical path can slow down the whole application therefore this variation is an important problem. To mitigate the performance variation among processors, we propose a speed-aware dynamic load balancing strategy which works on both homogeneous and non-homogeneous hardware. Our main idea is to provide an estimation of the task completion time based when moving a task from one processor to another on the processor speed. We show up to 30% performance improvement using our speed-aware load balancer compared to the no load balancing case. We also show that our speed-aware balancer performs 5% better than non-speed aware counterpart.**

Turbo Boost improves the clock speed and therefore the application performance [2]. However, it can also cause performance variation among processors. We observe that there exists up to 30% execution time difference among same-model processors under Turbo Boost running the same local computational kernel, as shown in Figure 1. Such variations can lead to performance degradation, especially for tightly coupled HPC applications. A slow processor in the critical path, can slow down the whole application.

To understand the cause of this performance variation, we look into the frequency and temperature of the processors. Figure 3 shows the frequency and temperature trends of tree selected same-model processors with Turbo Boost turned on in a cluster. Node 42, 48, and 70 demonstrate 3 distinct behaviors. Node-42 is a typical fast node. During the whole experiment, the temperature of Node-42 remains

Bis zu 16% Differenz in der Ausführungszeit von num. Kerneln

Bulk Synchronous vs Taskbasiert (1)

Bulk Synchronous Execution (now)

Bulk Synchronous Execution (later)

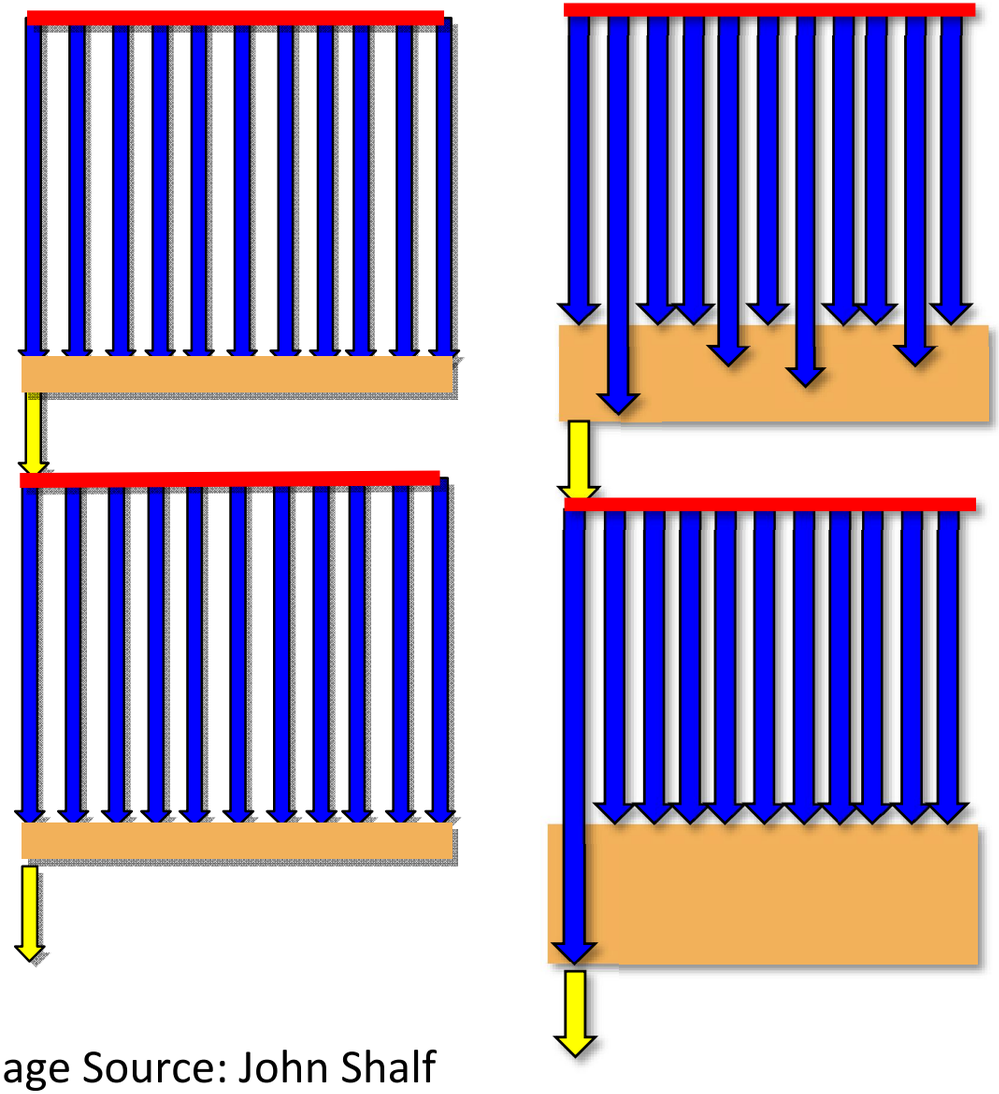


Image Source: John Shalf

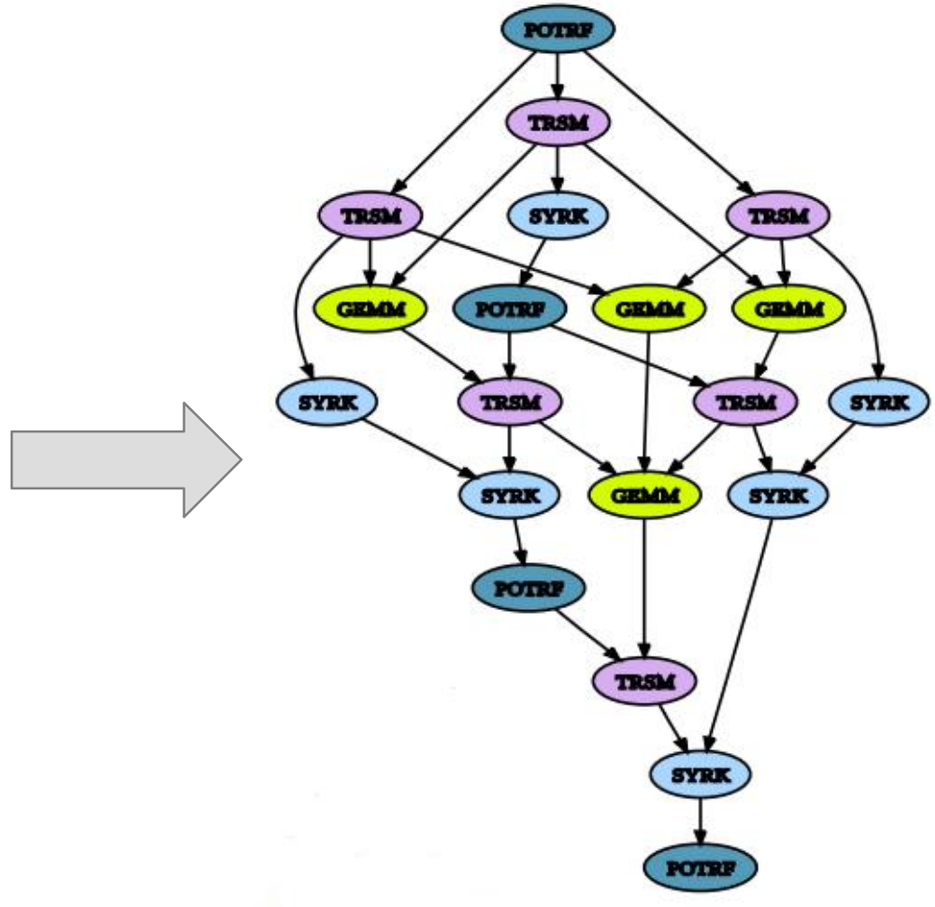


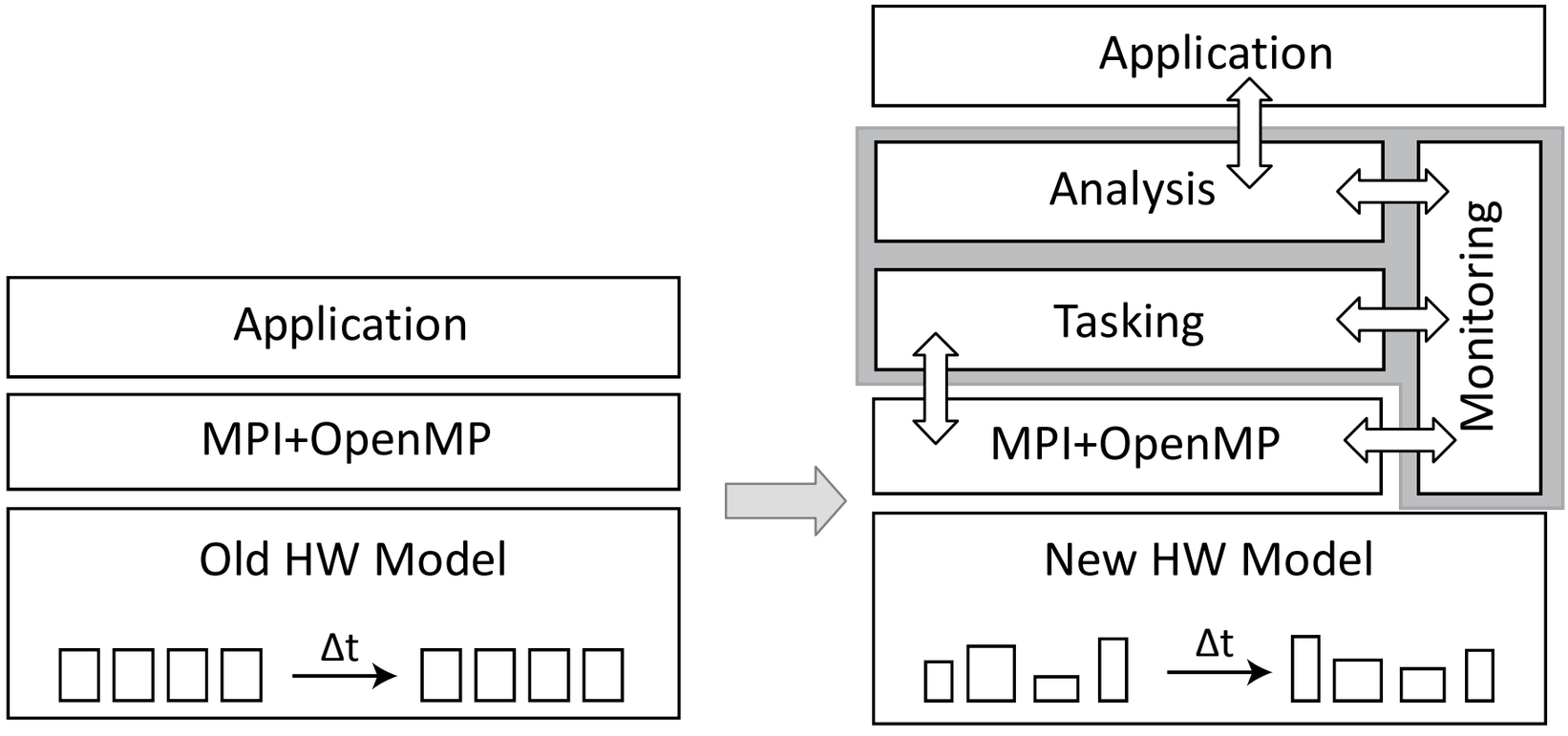
Image Source: Jack Dongarra

Bulk Synchronous vs Taskbasiert (2)

- Taskbasierte Programmiermodelle sind besser für dynamische Variabilität geeignet
  - Weniger globale Synchronisationspunkte
  - Stattdessen: Punkt-zu-Punkt Synchronisation und Eingabe-Ausgabeabhängigkeiten

- Existierende taskbasierte Ansätze:
  - Programmiermodelle wie **Charm++, HPX, Legion**, …
  - Das sind neue Programmiersprachen / Programmiersysteme
  - Aber die meisten HPC Anwendungen verwenden **MPI** und **OpenMP**

Chameleon - Ansatz

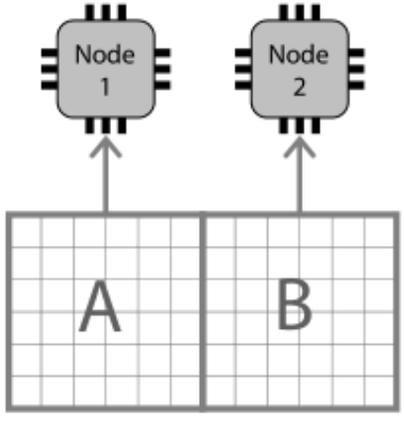


- 3 Komponenten von Chameleon
  - **Tasking Erweiterung** basierend auf OpenMP+MPI
  - **Monitoring**-Komponente für Introspektion
  - **Analyse**-Komponente für Datensammlung und Aufbereitung

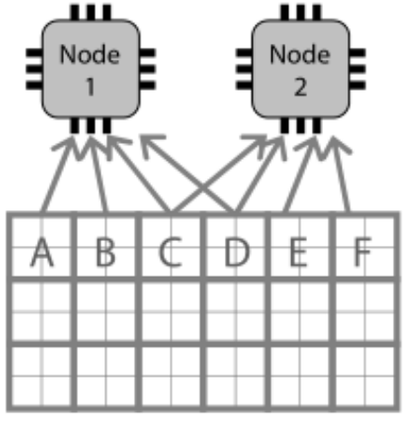
Chameleon – Tasking

- Erweiterung des Task-Konzepts in OpenMP
  - Tasking seit v3.0 in OpenMP unterstützt (knotenlokal)
  - Neuere OpenMP Features: Datenabhängigkeiten, Task Abbruch (cancellation)

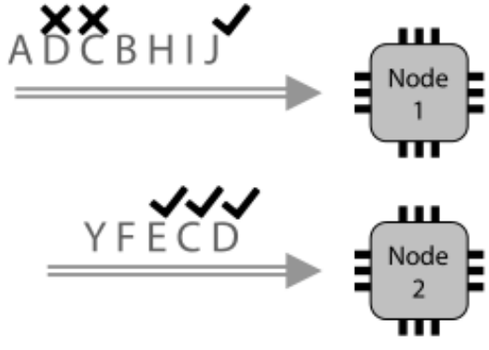
- Prototypische Chameleon Erweiterungen für Tasking:
  - **Knotenlokal**: Datenaffinität und dynamische Task Prioritäten
  - **Knotenübergreifend**: Replikation von Tasks und selektive Ausführung



Decomposition into tasks (A, B,...) and mapping



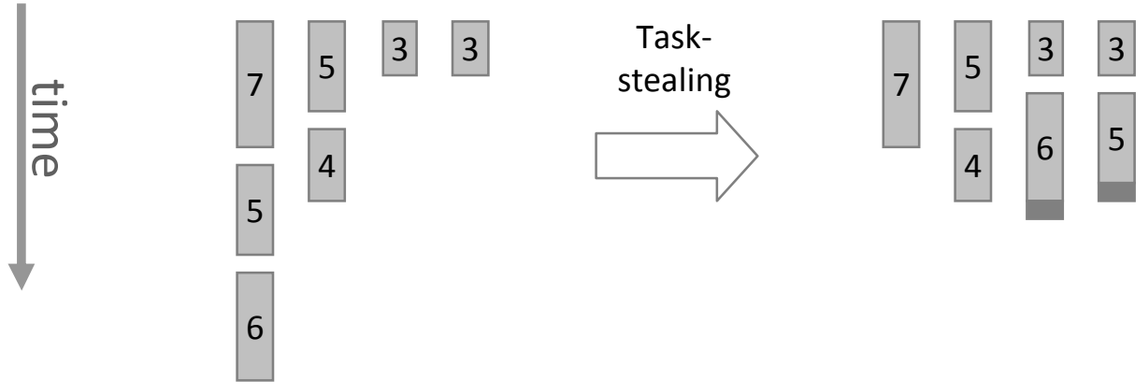
Overdecomposition and replication (e.g., tasks C, D)



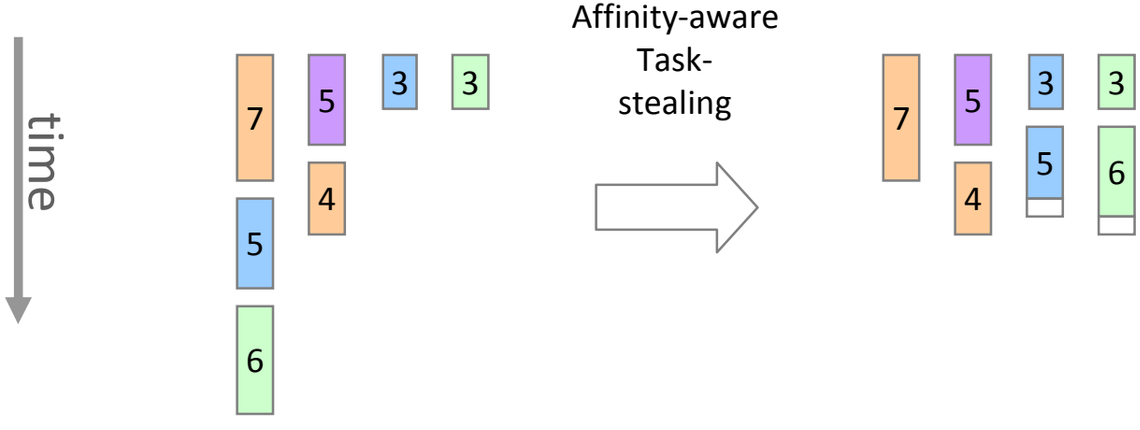
Task execution (✓) with cancellation (✗)

Knotenlokal: Datenaffinität

- OpenMP Tasks sind gut zur (knotenlokalen) Lastbalancierung geeignet (freie Threads “stehlen” Arbeit von überlasteten Threads)



- Problem: Datenlokalität wird dabei nicht berücksichtigt, aber mit NUMA immer wichtiger



NUMA node 0

NUMA node 1

NUMA node 2

NUMA node 3

Datenlokalität für OpenMP Tasks (1)

- Lösungsvorschlag: Neue **affinity** Klausel für OpenMP Tasks

```
#pragma omp task [clause…] affinity(list)
```

- Beispiel:

```
int a[100];
...
#pragma omp task affinity(a[0])
{
   // task that makes use of a[0]
}
```

- Affinity Klausel
  - `list` spezifiziert einen oder mehrere Speicherbereiche auf die ein Task zugreifen wird (data references)
  - Auch array slices und iteratoren werden unterstützt
  - Ist ein Hinweis für die Laufzeitumgebung zur Optimierung der Ausführung
  - Ausführung nahe den Daten wird dadurch nicht erzwungen

Datenlokalität für OpenMP Tasks (2)

- Prototypische Implementierung und Evaluation im Rahmen von Chameleon
  - Erweiterung der LLVM OpenMP Laufzeitumgebung (frei verfügbar, basierend auf Intel's OpenMP Laufzeit)

- Komponenten:
  - Affinity-aware task creation (Tasks in der Nähe der Daten erzeugen)
    Zwei Modi: "**temporal**" und "**domain**"
  - NUMA-aware task-stealing (zuerst von "nahen" Threads stehlen)

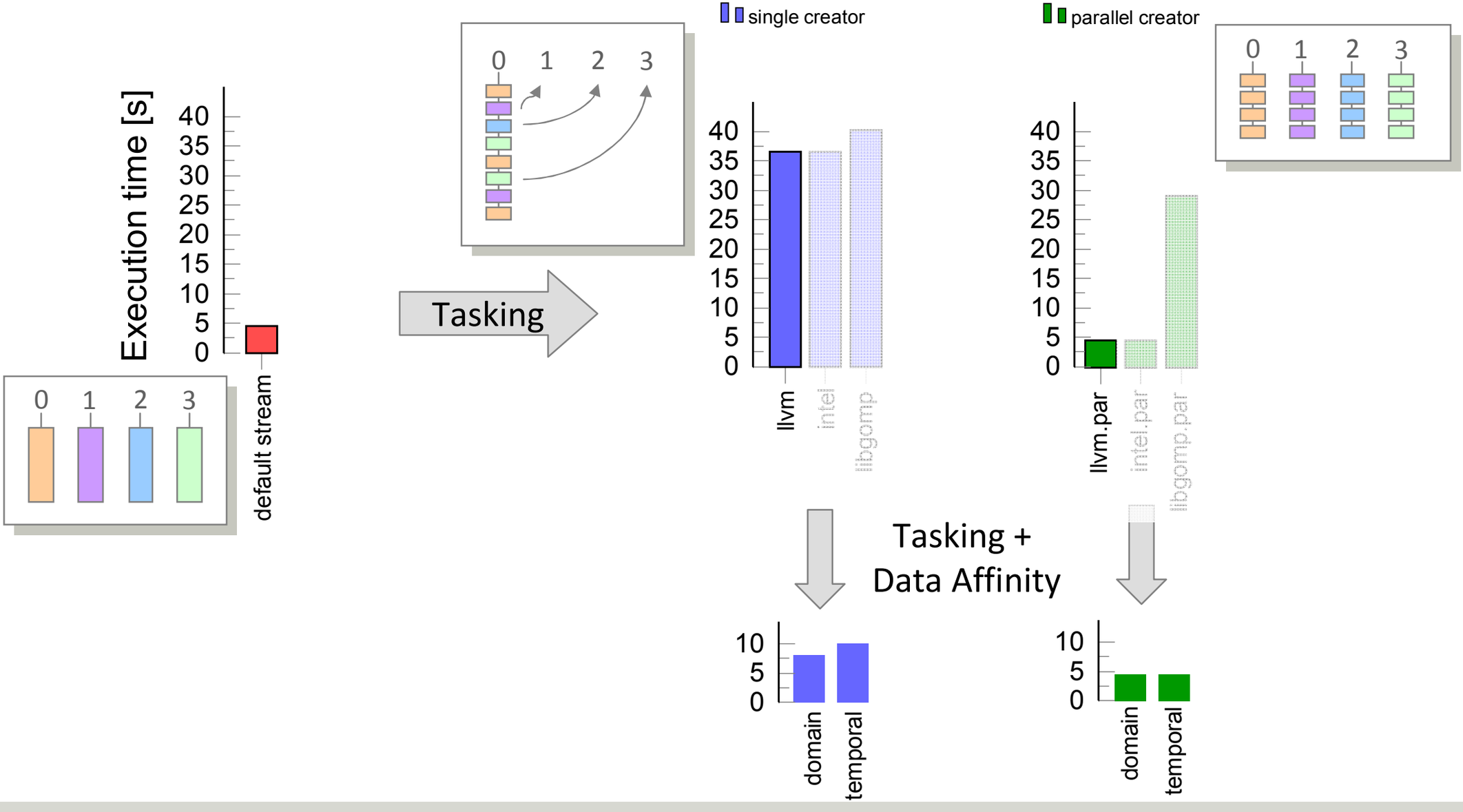
- IWOMP Publikation:

Jannis Klinkenberg, Philipp Samfass, Christian Terboven, Alejandro Duran, Michael Klemm, Xavier Teruel, Sergi Mateo, Stephen L. Olivier, and Matthias S. Müller. **Assessing Task-to-Data Affinity in the LLVM OpenMP Runtime**. Proceedings of the 14th International Workshop on OpenMP, IWOMP 2018. September 26-28, 2018, Barcelona, Spain.

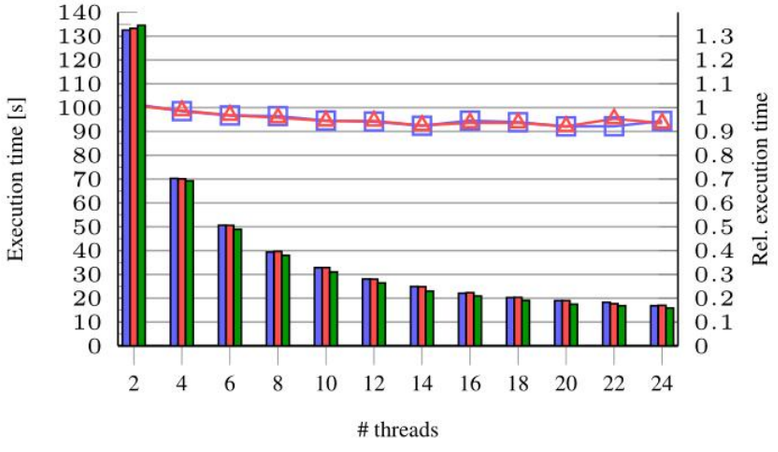
Aufnahme in die nächste Version (v5.0) der OpenMP Spezifikation beschlossen

Datenlokalität für OpenMP Tasks - Ergebnisse

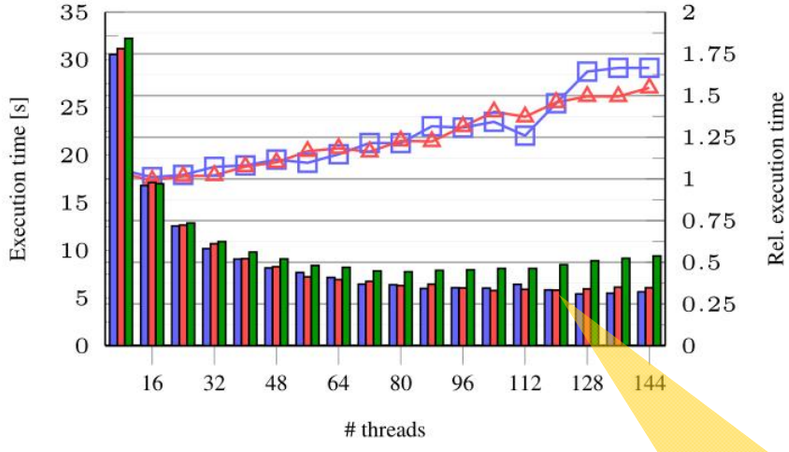
- Beispiel: Stream benchmark
  - 8-Sockel Broadwell System, 2.2 GHz, 1 TB Speicher



Weitere Ergebnisse, Rekursive Anwendungen

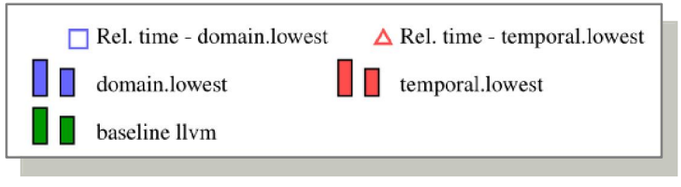


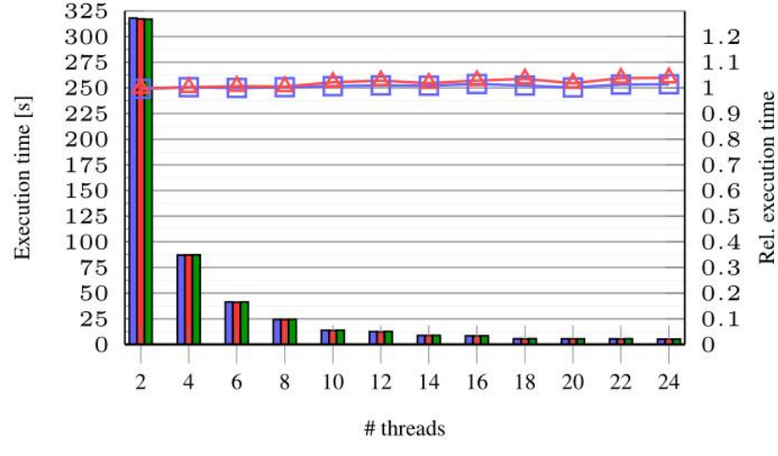
(c) Merge sort on 2-socket



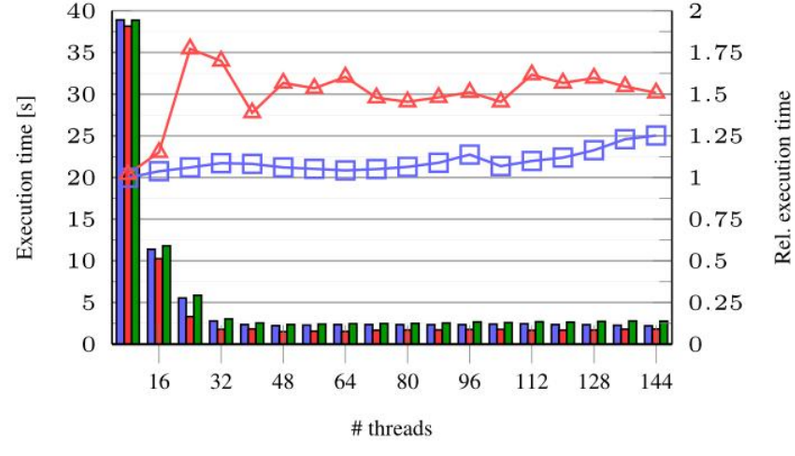
(d) Merge sort on 8-socket

Kein Vorteil bei 2-Sockel System, aber bei 8-Sockel System bis zu 1.75x schneller





(g) Health on 2-socket

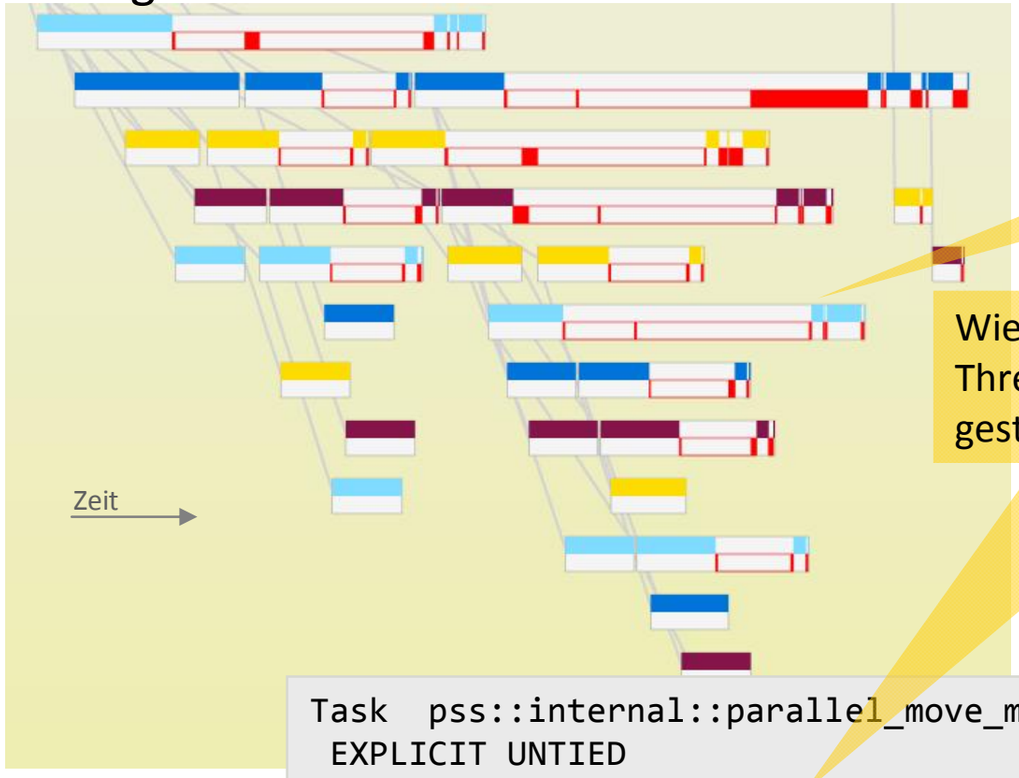


(h) Health on 8-socket

OpenMP Monitoring

- Monitoring von OpenMP Tasks
  - Deutliche Vereinfachung durch neues OMP-T Interface (OpenMP v5.0)
  - Profiling und Tracing Ansätze im Rahmen von Chameleon untersucht

Tracing:



Zeitachse der Tasks mit Eltern-Kind Beziehung und Aktivität von Threads bzw. Wartezeit

Wie oft wurde ein Task von einem Thread erzeugt, ausgeführt, und gestohlen

Wo wurden Task ausgeführt, und wie lange dauerte die Ausführung

Profiling:

```
Task  pss::internal::parallel_move_merge<int*, int*, int*, std::less<int> >
 EXPLICIT UNTIED
   TID  createC    execC   stealC        execT         minT         maxT   Core(s)
     0       15       13        0  0.000843339  4.19728e-05  0.000143174   23
     1        2        2        1  0.000204701  4.91664e-05  0.000155535   13
     3        0        2        2  9.94541e-05  4.77023e-05  5.17517e-05   14
   Sum       17       17        3   0.00114749  4.19728e-05  0.000155535
```

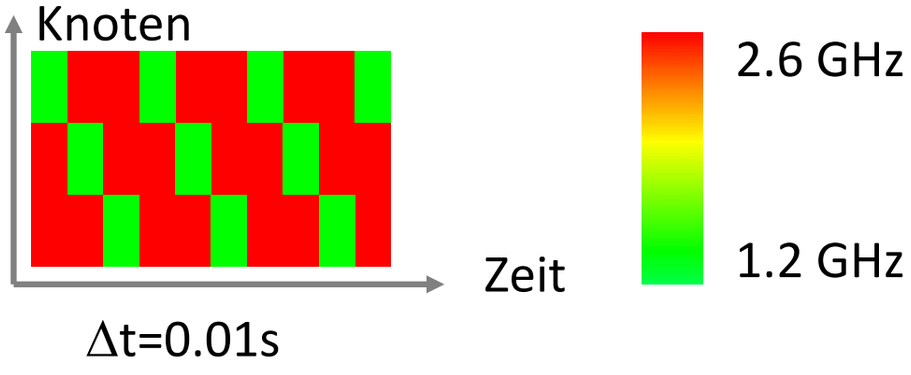
Knotenübergreifendes Tasking: Sam(oa)²

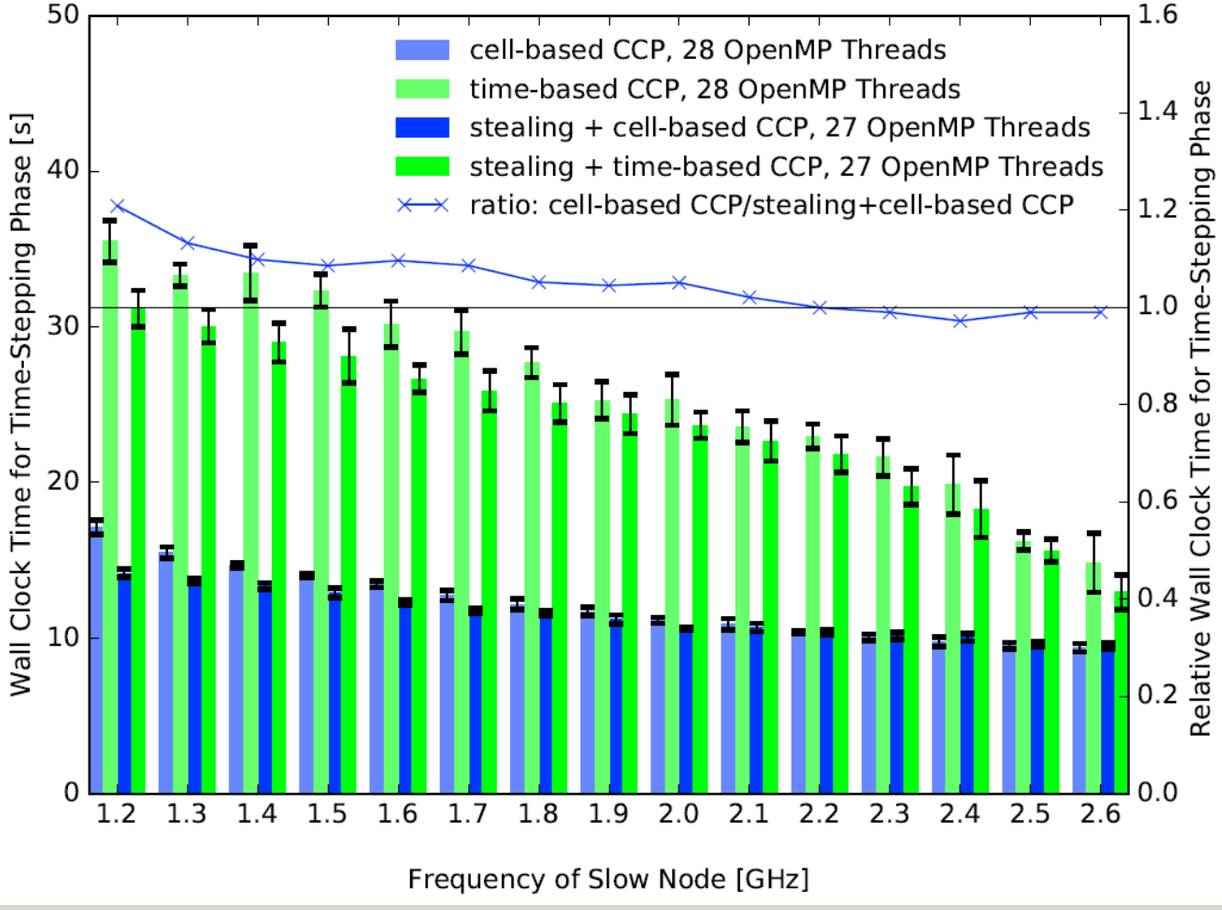
- Case Study: Sam(oa)² Finite-Volumen und Finite-Elemente Simulationen auf adaptiven Dreiecksgittern (AMR)
  - Natürliche Lastungleichgewichte durch AMR
  - Existierendes Balancierungsverfahren: "Chains-on-Chains Partitioning" (CCP)
  - Zeit-basiertes CCP und Zellen-basiertes CCP

- Implementierung von entferntem Task Stealing
  - Ein dedizierter Thread zur Ermittlung der Leistungseigenschaften und zum Stehlen von entfernten Sections (Tasks)
  - Atomare Zuweisung von Sections via MPI-3 RMA

Philipp Samfass, Jannis Klinkenberg, and Michael Bader. **Hybrid MPI+OpenMP Reactive Work Stealing in Distributed Memory in the PDE Framework sam(oa)²**. Proceedings of the 2018 IEEE International Conference on Cluster Computing (CLUSTER). September 10-13 2018, Belfast, UK

Szenario: Wechselnde Verlangsamung einzelner Knoten





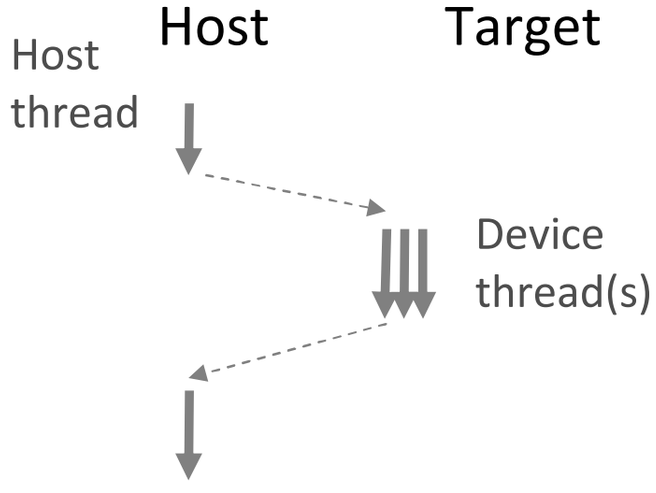
- Zeit-basiertes CCP funktioniert hier sehr schlecht
  - Vorhersage immer wieder falsch
- Workstealing liefert hier die besten Ergebnisse
  - Bis zu 20% Verbesserung
- Weitere Experimente:
  - Stealing auch für AMR-induzierte Lastungleichgewichte
  - Verbesserung der Skalierung

Nächste Schritte

- Vereinfachtes Konzept zum knotenübergreifenden Tasking (WIP)
  - Idee: Abwandlung des OpenMP “target” Mechanismus

- OpenMP Target:
  - Spezifikation von Code und Datenumgebung zur Ausführung auf Beschleunigerhardware (“target”)

```
#pragma omp target map(a,b,c)
{
  // target region
  for(int i=0; i<N; i++) {
    c[i] = a * b[i];
  }
}
```

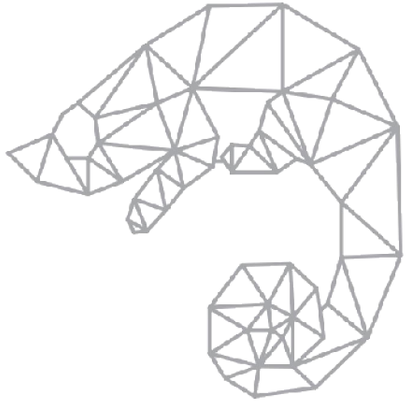


- Abwandlung für Chameleon:
  - Target sind andere MPI Prozesse statt Beschleuniger
  - Entfernte Ausführung möglich, aber nicht vorgeschrieben

Zusammenfassung

- Chameleon
  - Taskbasierte Programmierung für OpenMP+MPI
  - Ziel: Optimierung der Ausführung durch Laufzeitmessung (“Introspektion”)
  - Knotenlokal: Dynamische Priorisierung, Affinität
  - Knotenübergreifend: Replikation und selektive Ausführung (keine autom. Migration)

- Status
  - Vorschlag zur Task-Daten Affinität in OpenMP Spezifikation übernommen
  - Case-study zum knotenübergreifenden Tasking
  - Monitoring für OpenMP tasks in Arbeit
  - “Target” basiertes Konzept für knotenübergreifendes Tasking in Arbeit



www.chameleon-hpc.org