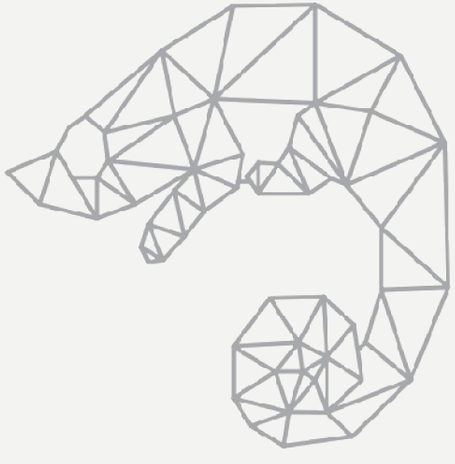
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Dr. Karl Fürlinger
Lehrstuhl für Kommunikationssysteme und Systemprogrammierung

Chameleon - Eine Taskbasierte Programmierumgebung zur Entwicklung reaktiver HPC Anwendungen



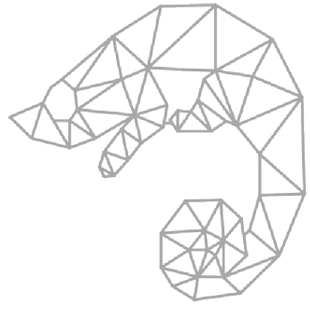
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

RWTH AACHEN UNIVERSITY

TUM

Chameleon – Überblick

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



www.chameleon-hpc.org

- Projektpartner
  - **LMU München**,
    LFE für Kommunikationssysteme und Systemprog.
  - **RWTH Aachen**,
    Lehrstuhl für Hochleistungsrechnen
  - **TU München**,
    Professur für Hardware-nahe 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 - Komplexität

	Cori[1] (NERSC)	Sierra, Summit[2] (LLNL, ORNL)	Aurora[3] (ANL)
Inst. Year	2016	2017-2018	2018-2019
Peak Perf.	>30 PF		180-450 PF
	7 MW	10 MW	
	3 TF	>40 TF	
No. of Nodes	9,300	3,400	>50,000
Node	Intel Xeon Phi (Knights Landing)	IBM Power 9 + Nvidia Volta	Intel Xeon Phi (Knights Hill)
	Cray Aries	Dual Rail EDR Infiniband	2nd gen I
Memory	DDR4 + On-package high BW memory	DRAM + stacked DRAM	7 PB DRAM+ Persistent Memory

Complex node architecture

Homogeneous manycore

Heterogeneous manycore (accelerator based)

Heterogeneous memory (regular DRAM+stacked RAM)

Persistent memory (NVRAM)

1: https://www.nersc.gov/users/computational-systems/cori/ 2: http://www.olcf.ornl.gov/summit/ 3: http://aurora.alcf.anl.gov/

Motivation – Dynamische Variabilität

- Dynamische Variabilität steigt
  - ZB. Intel Turbo Modus
  - Komplexe Speichersysteme
  - HW Fehlererkennung und Behandlung

Our Dynamic Future

Pete Beckman | Argonne National Laboratory and Northwestern Uni

Last month, as I tossed my bags in a rental car at the airport, I noticed that the car was particularly new. I was quite surprised, however, when I drove up to the first stop sign, and the car suddenly died. It was as if I had run out of gas or turned off the ignition. However, as soon as I took my foot off the brake pedal, the engine started itself back up. I pushed on the accelerator, and the car jumped forward. Over the next couple of days, I explored this advanced fuel-saving feature, trying to understand under what circumstances the car's algorithms would decide it could save gas by temporarily shutting off and how quickly I could jump forward after moving my foot from brake to accelerator as the car automatically started itself and slowly adjusted the throttle.

Dynamic power management is everywhere, from cars and mobile phones to home heating and cooling. One of the key technology changes making advanced power management possible for your automobile is the shift to fly-by-wire

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 variation also manifests itself as frequency differences among processors under Turbo Boost dynamic overclocking. This variation can lead to unpredictable and suboptimal performance 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: Edison, 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 instrumentation 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 dynamic 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.

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

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

Bulk Synchronous vs Taskbasiert (1)

Bulk Synchronous Execution (now)

Bulk Synchronous Execution (later)

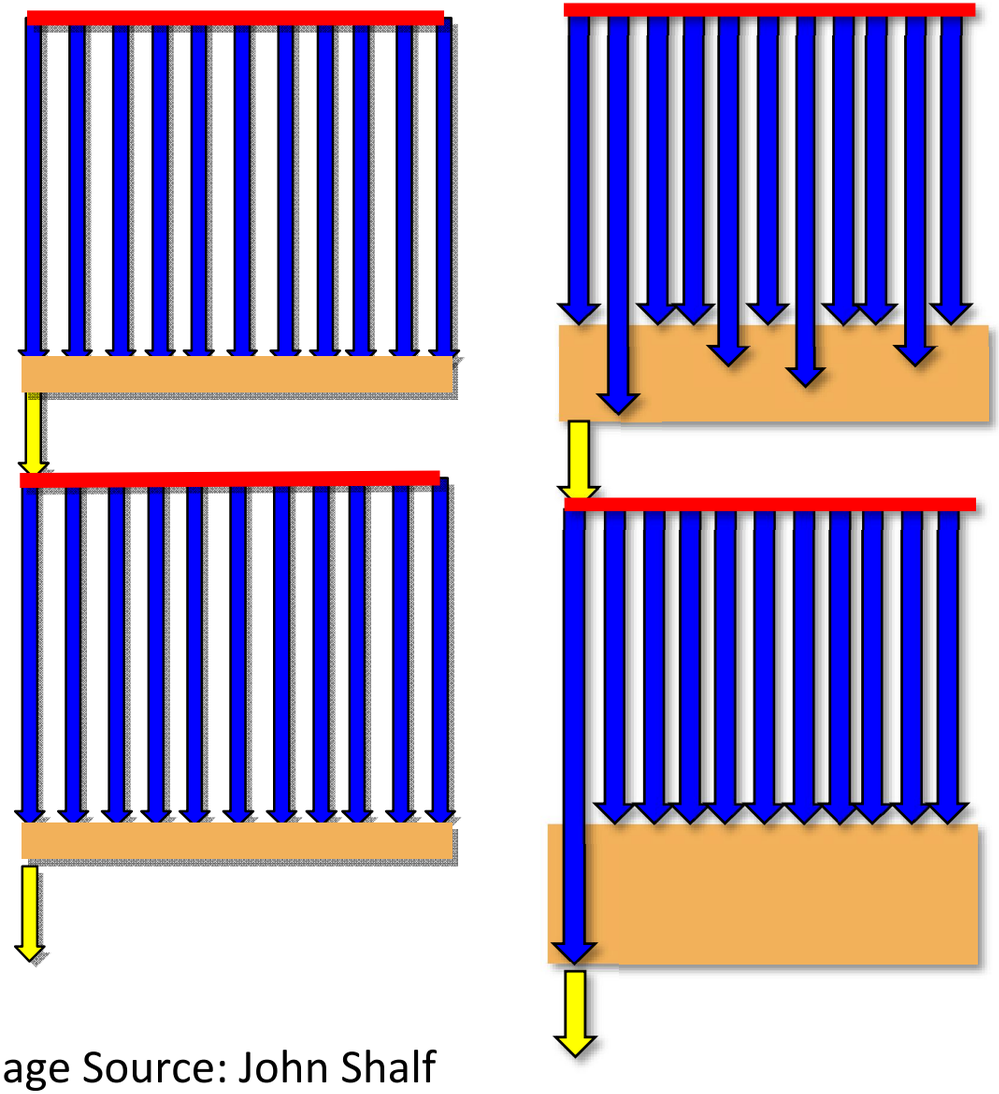


Image Source: John Shalf

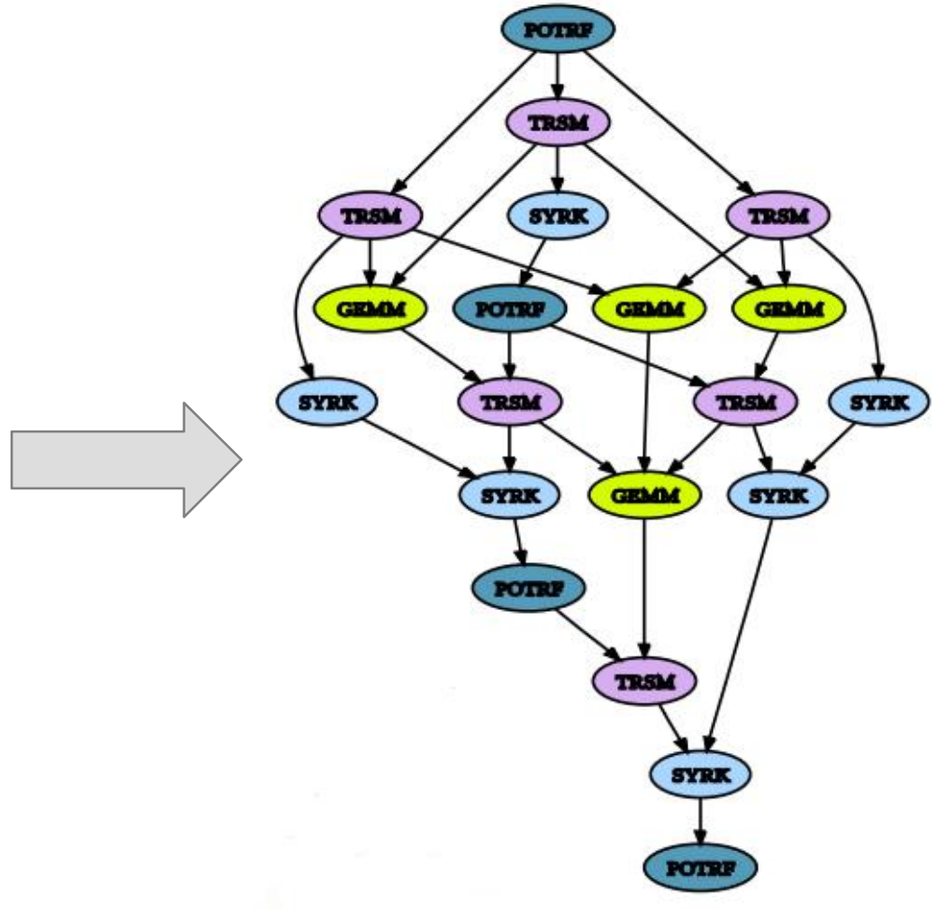


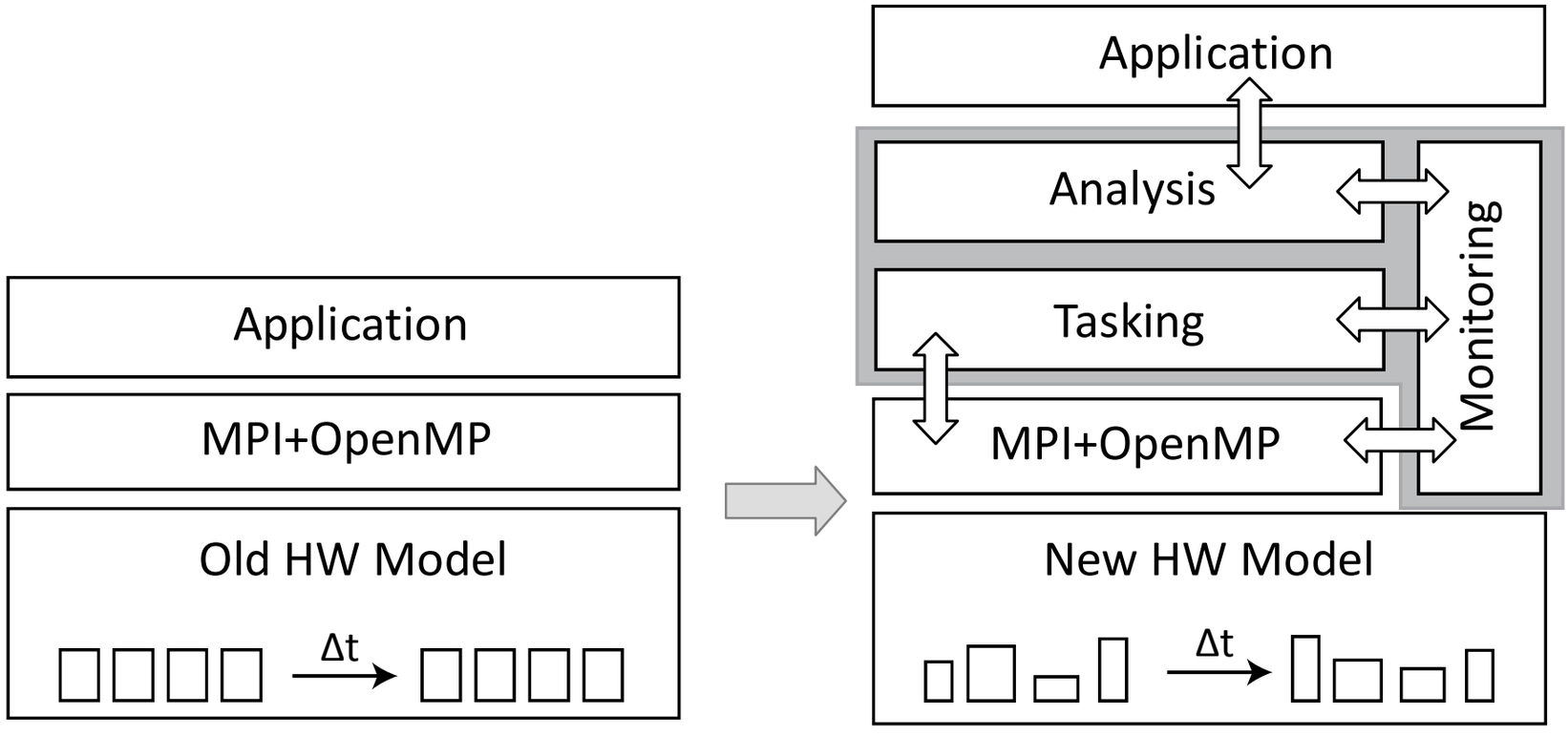
Image Source: Jack Dongarra

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

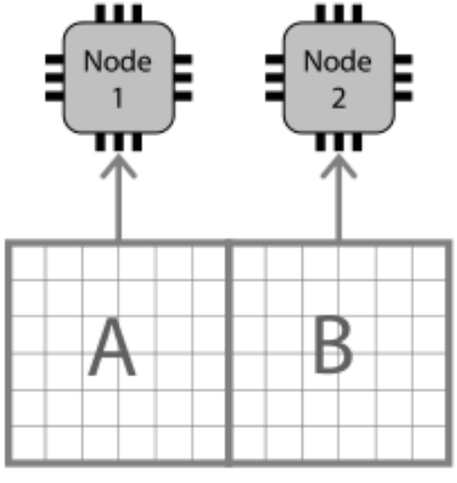
Chameleon - Ansatz



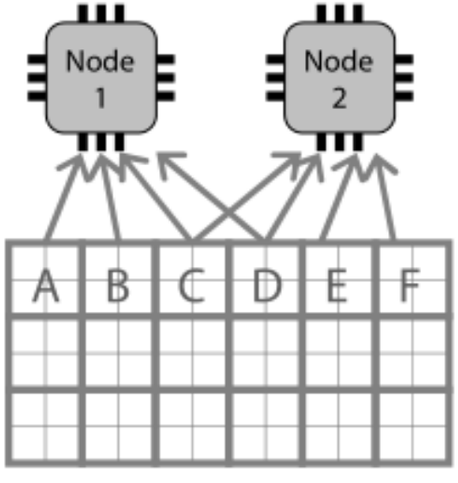
- 3 Komponenten von Chameleon
  - Tasking Erweiterung basierend auf OpenMP+MPI
  - Monitoring-Komponente für Leistungsintrospektion
  - Analyse-Komponente für Datensammlung und Aufbereitung

Chameleon – Tasking

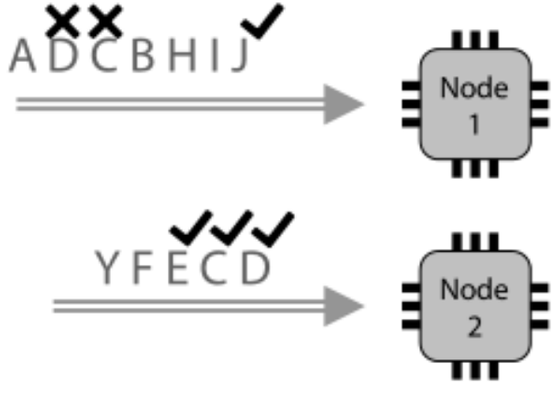
- Erweiterung des Task-Konzepts in OpenMP
  - Tasking seit v3.0 in OpenMP unterstützt (knotenlokal)
  - Neuere OpenMP Features: Datenabhängigkeiten, Task Abbruch (cancellation)
- Prototypische Chameleon Erweiterungen:
  - **Knotenlokal**: Datenaffinität und dynamische Task Prioritäten
  - **Knotenübergreifend**: 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 (✗)

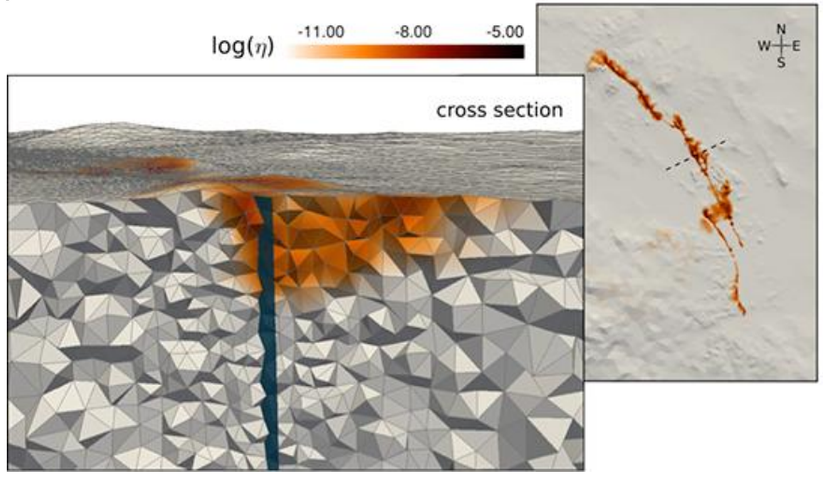
Chameleon – Monitoring / Introspection

- Basierend auf IPM:
  - IPM: Integrated Performance Monitor
  - Leichtgewichtiges Werkzeug zur Überwachung von MPI Anwendungen
  - Entwickelt in Zusammenarbeit zw. LBNL und LMU München

- Für Chameleon
  - Erweiterung um OpenMP Monitoring (via OpenMP Tools Interface OMPT)
  - Erweiterung vom Postmortem-Werkzeug zum Online-Werkzeug
  - Definition einer Introspektions Schnittstelle zw. Anwendung und Monitor

Chameleon – Testanwendungen und -szenarien

- Zwei Anwendungen: SeisSol und sam(oa)²



- SeisSol: simuliert komplexe Erdbebenszenarien und Ausbreitung von seismischen Wellen
  - Eigene Implementierung einer Taskqueue mit Vorrangrelationen
  - Anwendungsfall in Chameleon: knotenlokales Tasking
  - Dynamische Task Prioritäten
  - Task Affinity

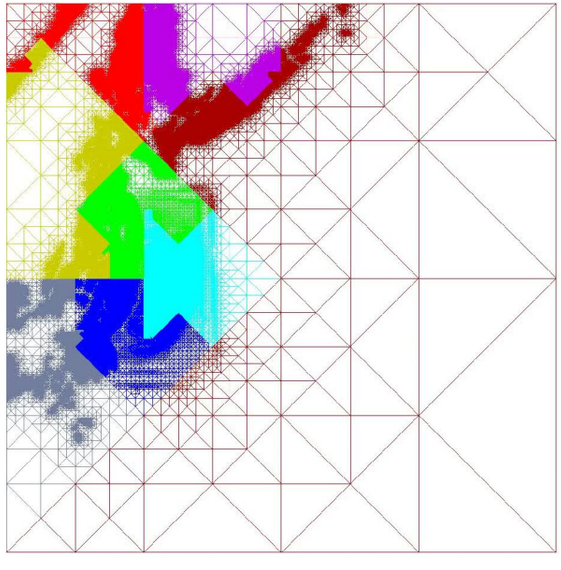
- sam(oa)²
  - Adaptive Mesh Refinement auf Dreiecksgittern
  - Sierpinksi Raumfüllende Kurve und 1D Lastbalancierung
  - Anwendungsfall in Chameleon: Knotenübergreifendes Tasking

Pilotstudie – Anwendung sam(oa)²

- sam(oa)²: Finite-Volumen und Finite-Elemente Simulationen auf adaptiven Dreiecksgittern (AMR)
  - Entwickelt an der TUM: M. Bader, O. Meister, P. Samfass

- Lösung von Hyperbolischen Partiellen Differentialgleichungen
  - Anwendungsfall: Tsunami Simulation

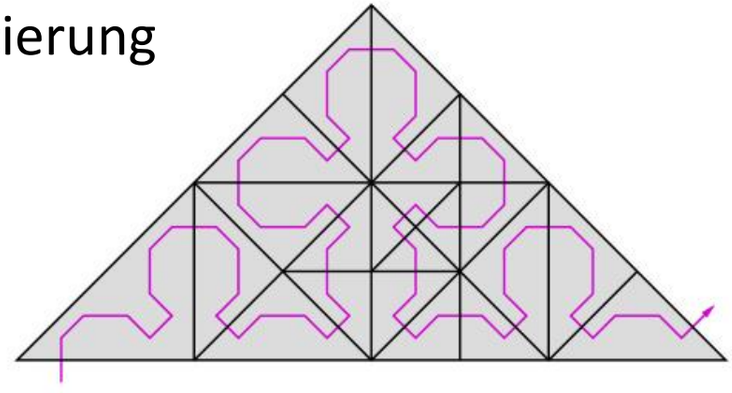
- Verwendet Sierpinksi Raumfüllende Kurven für Lastverteilung und –balancierung

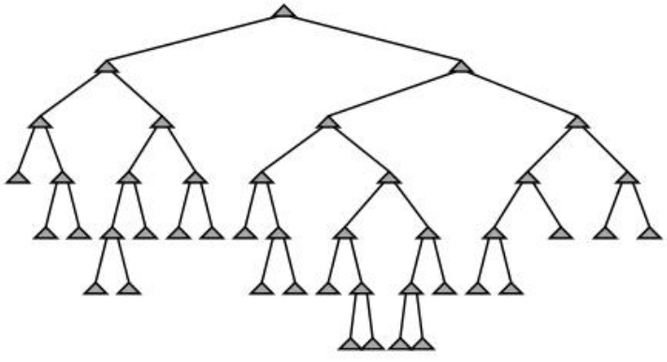


Source: M. Bader / P. Samfass

sam(oa)²

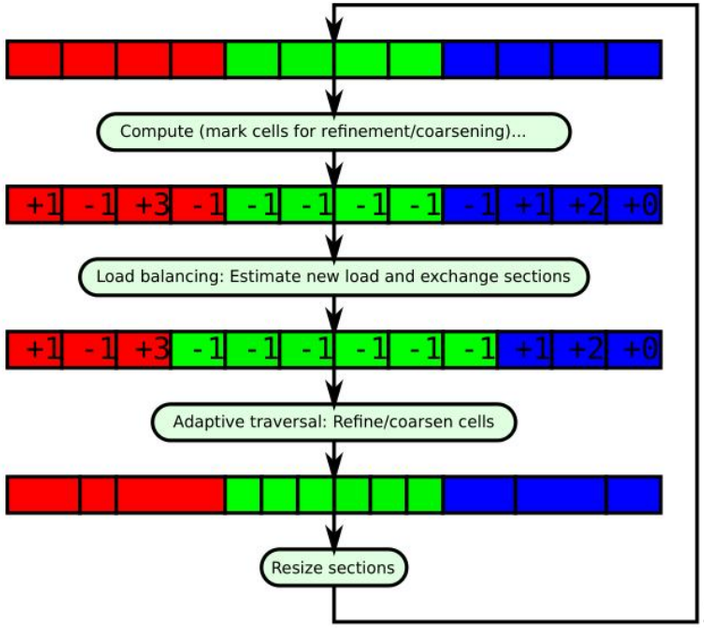
- Lastbalancierung basierend auf Abschnitten der Sierpinksi Kurve
  - Sog. “sections”.
  - 1D Lastbalancierung





Source: M. Bader / P. Samfass

- “Chains-on-Chains Partitioning”



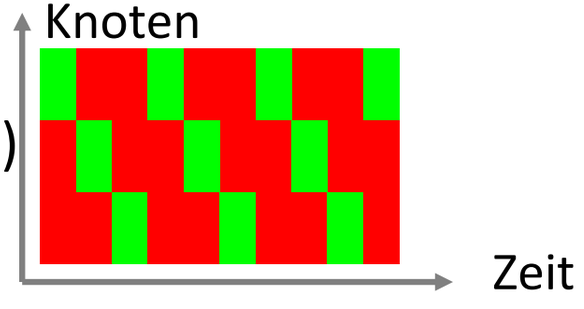
Source: M. Bader / P. Samfass

Pilotstudie: Workstealing in Sam(oa)²

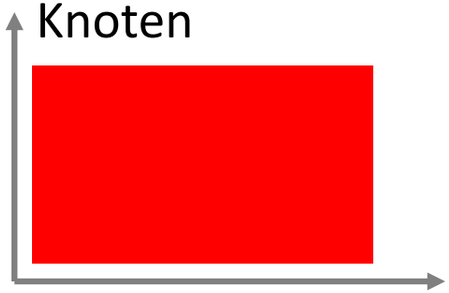
- Workstealing Implementierung in Sam(oa)²
  - Ein dedizierter Thread zur Ermittlung der Leistungseigenschaften und für Sehlen von entfernten Sections
  - Atomare Zuweisung von Sections via MPI-3 RMA

- Szenarien
  - 1: Nur ein langsamer Knoten

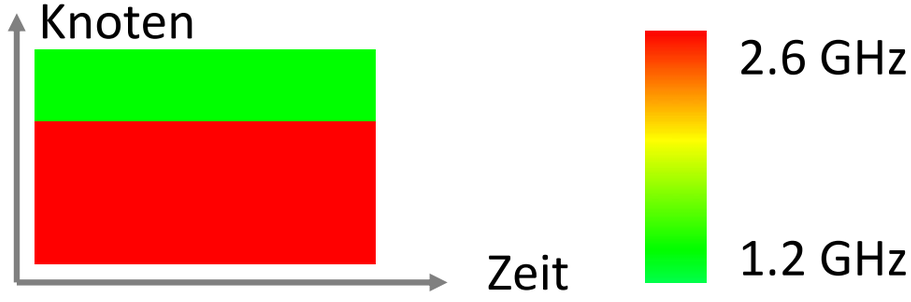
  - 2: Wechselnde Verlangsamung (Round-Robin)

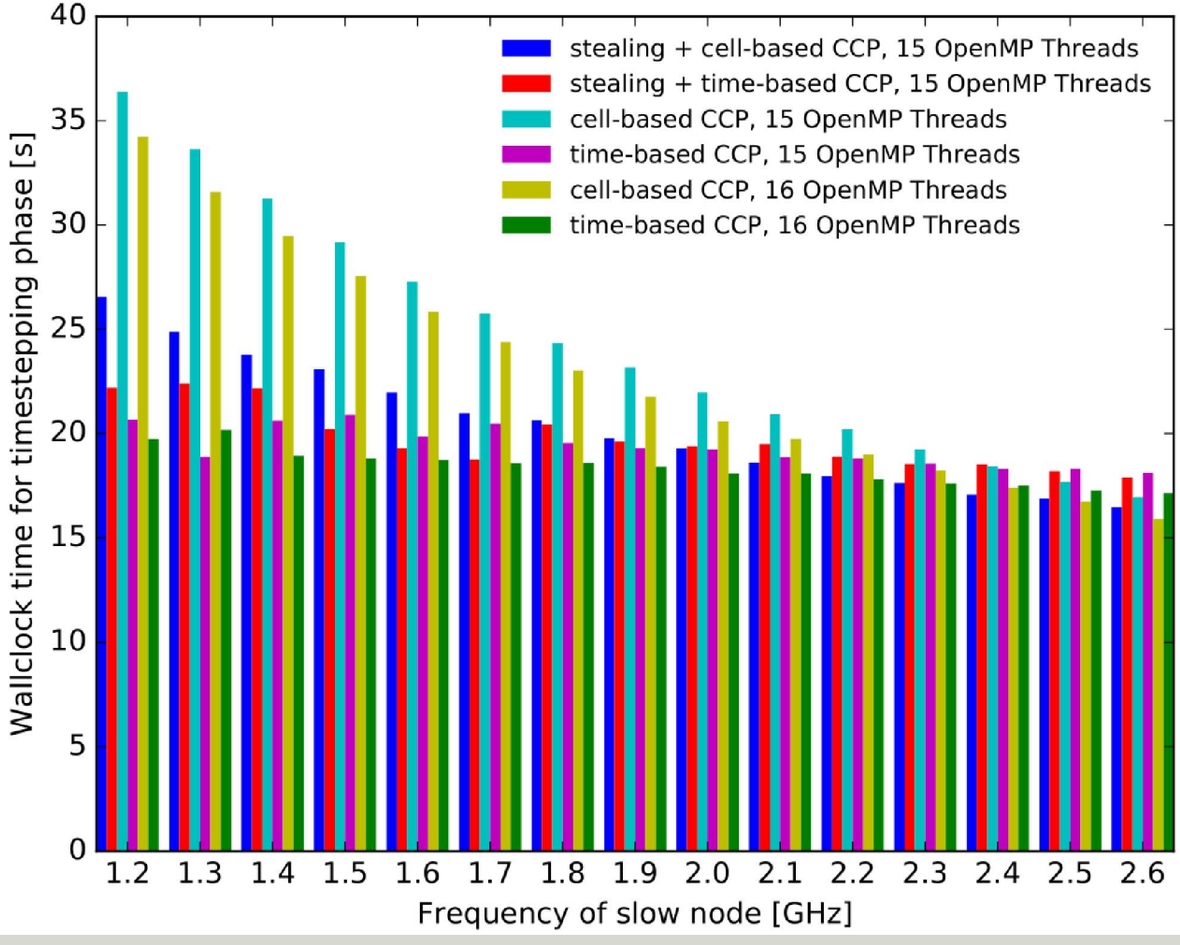


  - 3: Workstealing an Stelle der existierenden CCP Lastbalancierung (mit starkem Lastungleichgewicht in der Anwendung)



Szenario 1: Ein langsamer Knoten

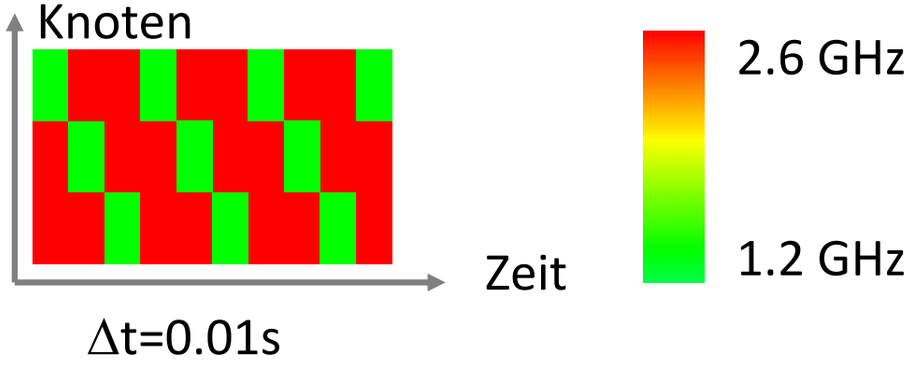




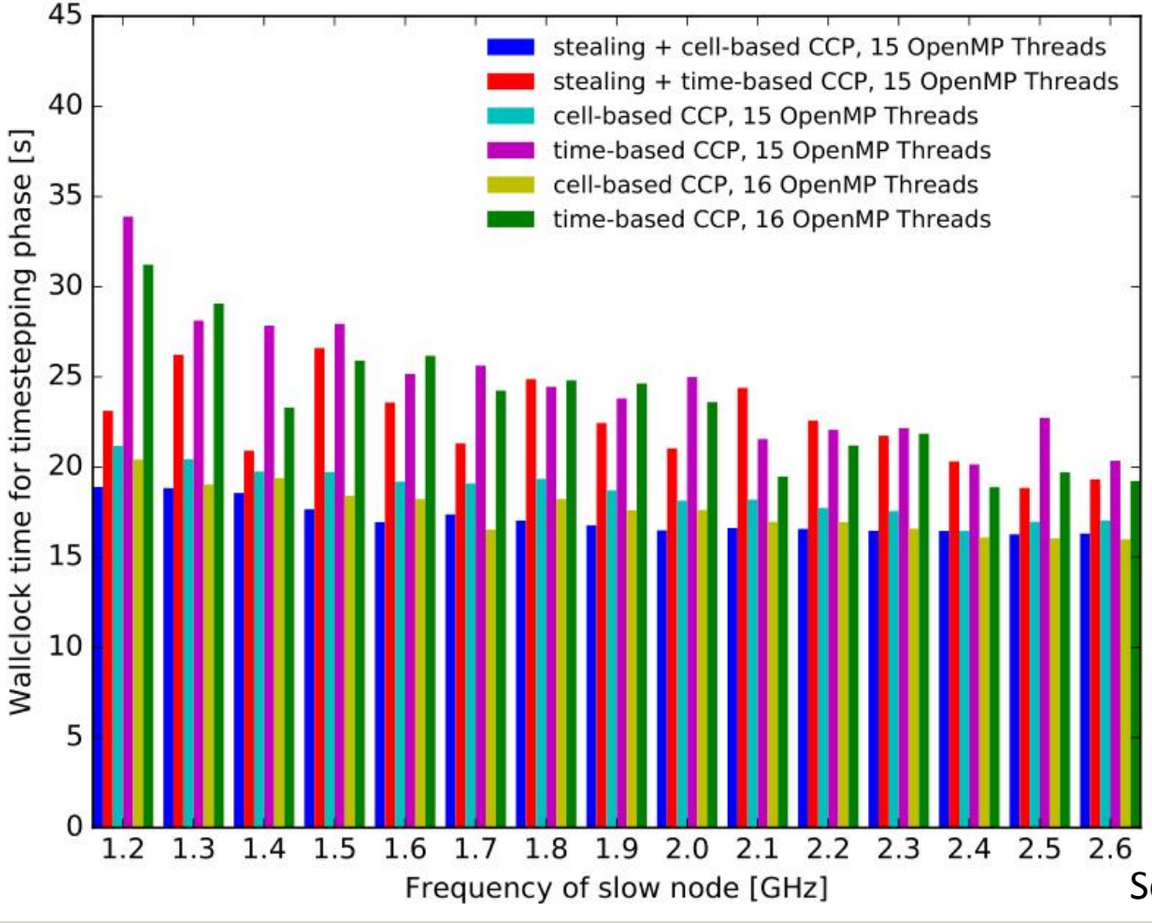
- Zeit-basiertes CCP Lastbalancierung funktioniert gut
  - Für diesen Anwendungs-fall zu erwarten
- Workstealing nicht ganz so effektiv
  - Overheads
  - Beschränkt auf einen Teil der Berechnung (Zeitschritt)

Source: M. Bader / P. Samfass

Szenario 2: Dynamische Verlangsamung

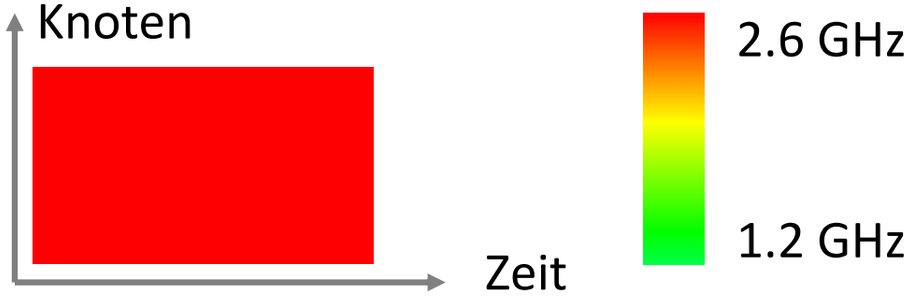


- Zeit-basiertes CCP funktioniert deutlich schlechter
  - Vorhersage falsch
- Workstealing liefert hier die besten Ergebnisse

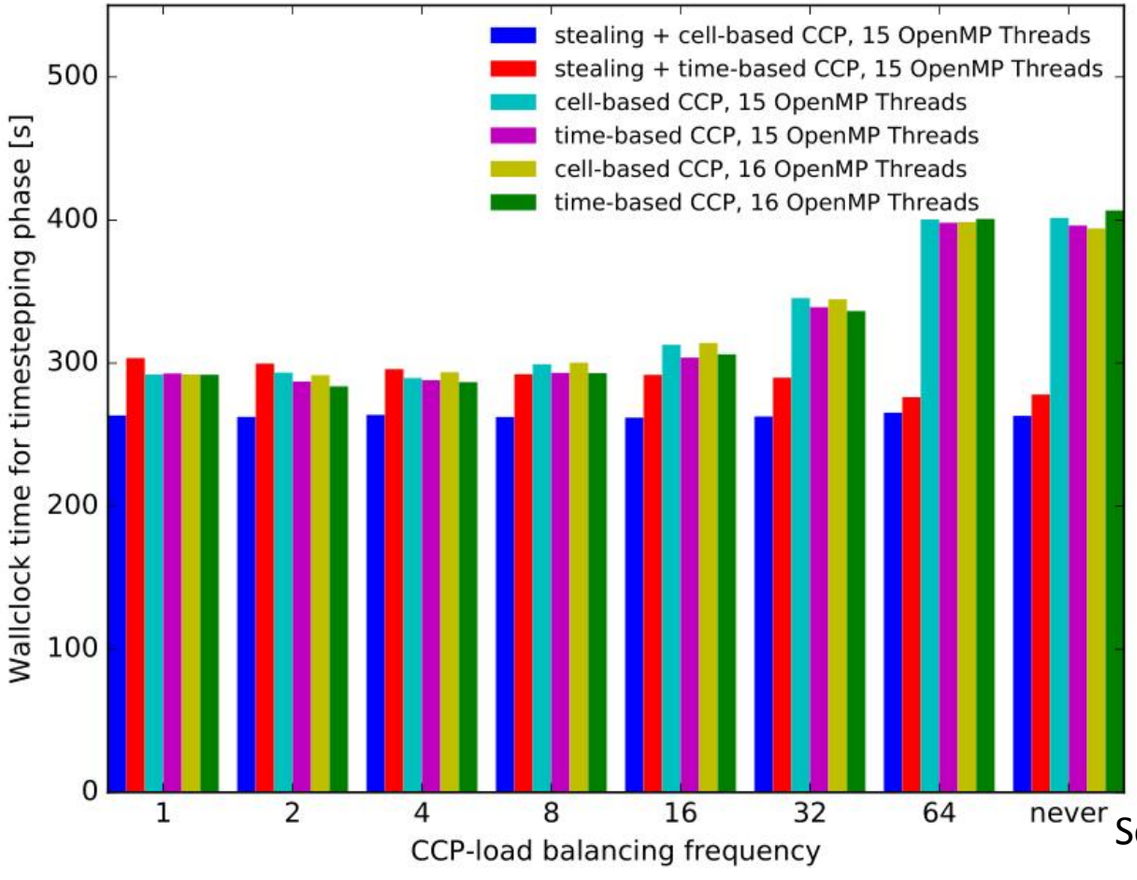


Source: M. Bader / P. Samfass

Szenario 3: Workstealing als Alternative zu CCP



- Workstealing eignet sich als Alternative zum anwendungsgetriebenen Loadbalancing

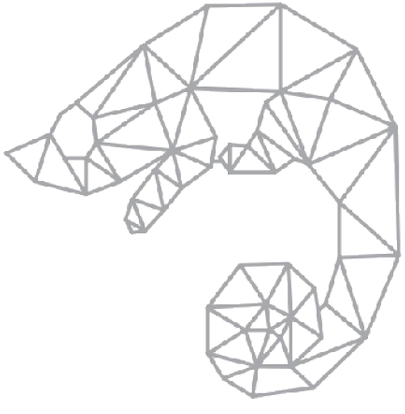


Source: M. Bader / P. Samfass

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
  - Pilotstudie zu Workstealing in sam(oa)²
  - Introspektion API Draft, demnächst verfügbar auf der Chameleon Webpage
  - WIP: Task Replikation und selektive Ausführung
  - WIP: Monitoring und Leistungs Introspektion



www.chameleon-hpc.org