Task-parallel Programming for Reactive Numerical Simulation

Christian Terboven <terboven@itc.rwth-aachen.de> & Matthias S. Müller <mueller@itc.rwth-aachen.de>

November 7th 2019, Tokyo, Japan

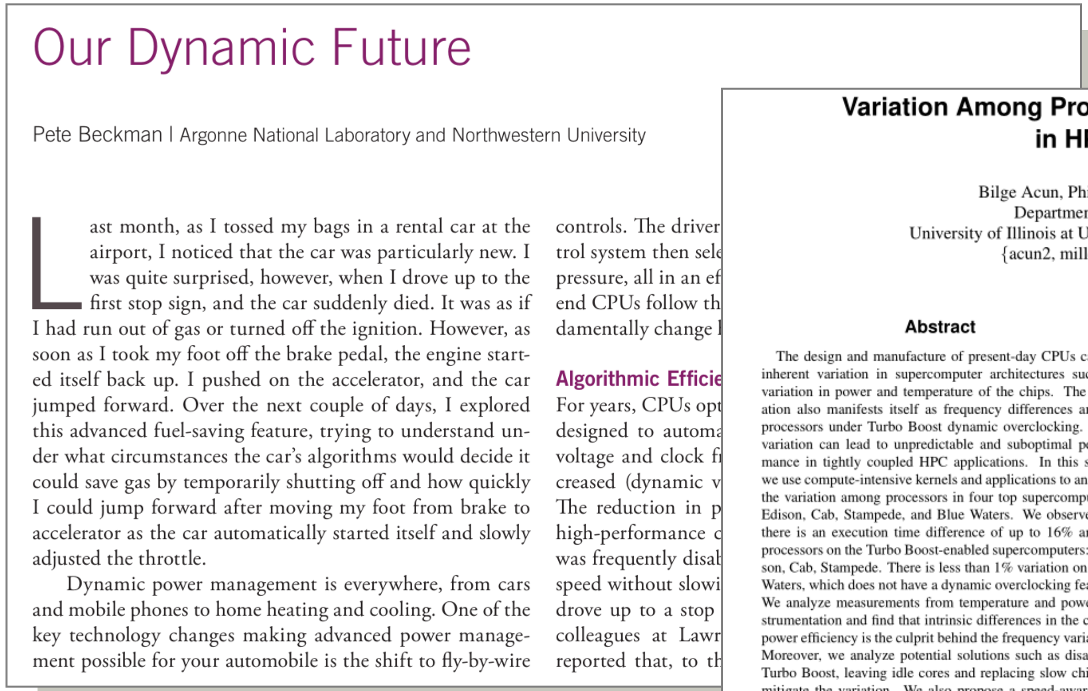
i12 High Performance Computing

RWTH AACHEN UNIVERSITY

Motivation / 1

The future is *dynamic*

- Dynamic Variability in HPC systems continues to increase
  - Processor features (example: Intel Turbo)
  - Energy Management (example: Power Capping)
  - Detection and Correction of Errors



Our Dynamic Future

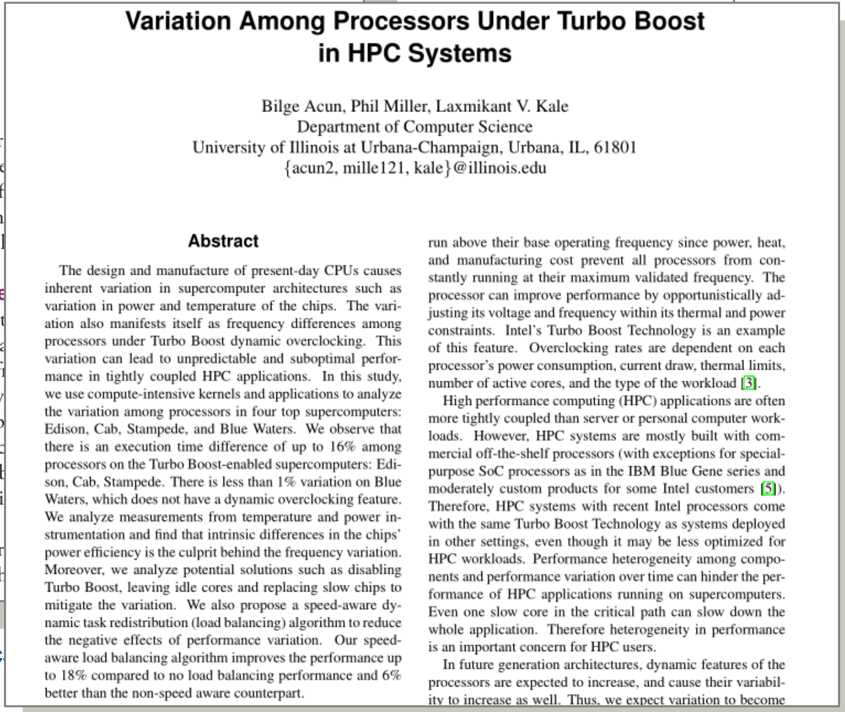
Pete Beckman | Argonne National Laboratory and Northwestern University

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

controls. The driver
trol system then sele
pressure, all in an ef
end CPUs follow th
damentally change l

Algorithmic Efficie
For years, CPUs opt
designed to automa
voltage and clock fr
creased (dynamic v
The reduction in p
high-performance c
was frequently disab
speed without slowi
drove up to a stop
colleagues at Lawr
reported that, to th



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

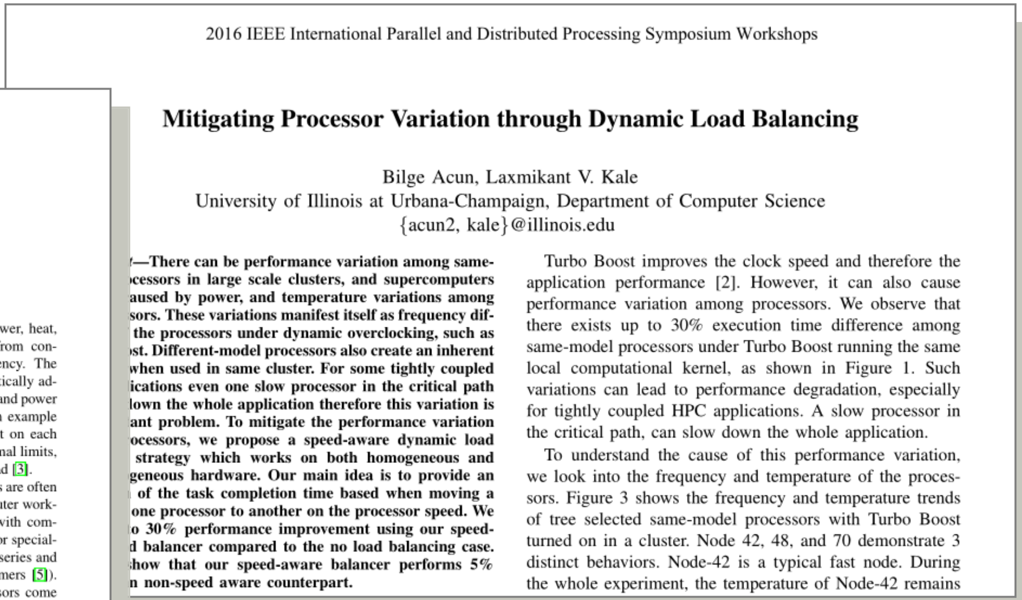
Abstract

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.

run above their base operating frequency since power, heat, and manufacturing cost prevent all processors from constantly running at their maximum validated frequency. The processor can improve performance by opportunistically adjusting its voltage and frequency within its thermal and power constraints. Intel's Turbo Boost Technology is an example of this feature. Overclocking rates are dependent on each processor's power consumption, current draw, thermal limits, number of active cores, and the type of the workload [3].

High performance computing (HPC) applications are often more tightly coupled than server or personal computer workloads. However, HPC systems are mostly built with commercial off-the-shelf processors (with exceptions for special-purpose SoC processors as in the IBM Blue Gene series and moderately custom products for some Intel customers [5]). Therefore, HPC systems with recent Intel processors come with the same Turbo Boost Technology as systems deployed in other settings, even though it may be less optimized for HPC workloads. Performance heterogeneity among components and performance variation over time can hinder the performance of HPC applications running on supercomputers. Even one slow core in the critical path can slow down the whole application. Therefore heterogeneity in performance is an important concern for HPC users.

In future generation architectures, dynamic features of the processors are expected to increase, and cause their variability to increase as well. Thus, we expect variation to become



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

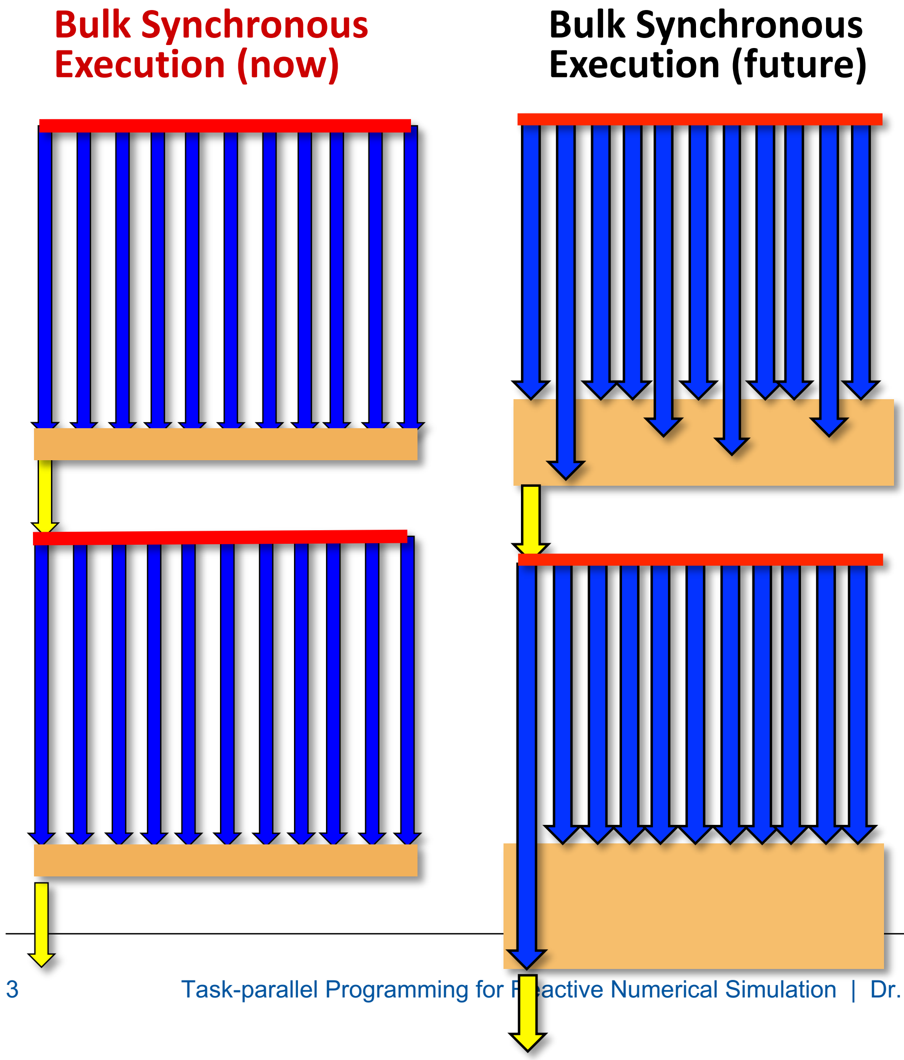
**—There can be performance variation among same-
cessors in large scale clusters, and supercomputers
aused by power, and temperature variations among
sors. These variations manifest itself as frequency dif-
the processors under dynamic overclocking, such as
st. Different-model processors also create an inherent
when used in same cluster. For some tightly coupled
ications even one slow processor in the critical path
lown the whole application therefore this variation is
ant problem. To mitigate the performance variation
ocessors, we propose a speed-aware dynamic load
strategy which works on both homogeneous and
geneous hardware. Our main idea is to provide an
of the task completion time based when moving a
one processor to another on the processor speed. We
o 30% performance improvement using our speed-
d balancer compared to the no load balancing case.
how that our speed-aware balancer performs 5%
n non-speed aware counterpart.**

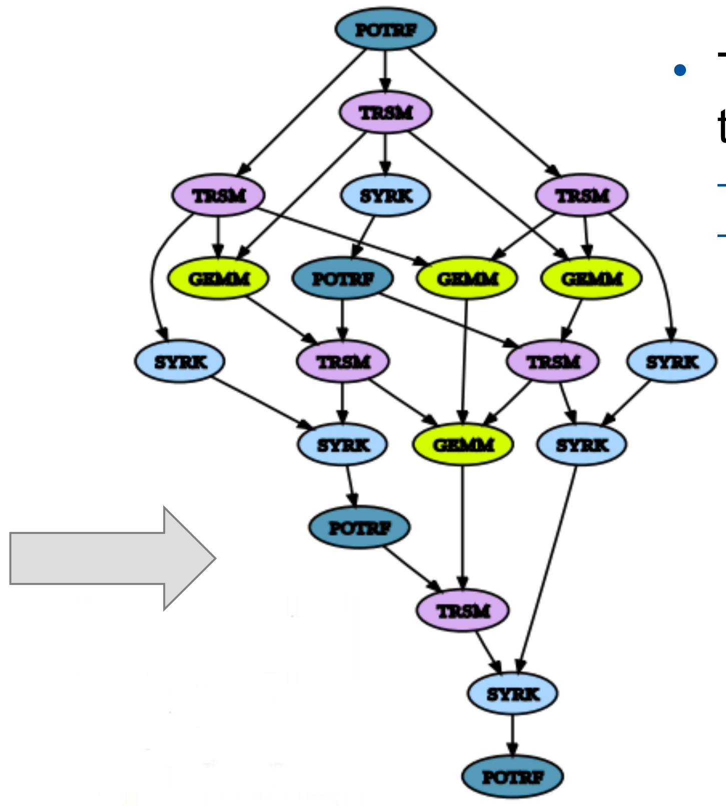
Turbo Boost improves the clock speed and therefore the application performance [2]. However, it can also cause performance variation among processors. We observe that there exists up to 30% execution time difference among same-model processors under Turbo Boost running the same local computational kernel, 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

Motivation / 2

Tasking to the rescue


Bulk Synchronous Execution (now)
Bulk Synchronous Execution (future)



- Tasking is well-positioned to react to dynamic system behavior
  - Less global synchronization
  - More p2p synchronization

Image Source: John Shalf

Image Source: Jack Dongarra

Agenda

- Intra-node: Task Affinity
- Inter-node: Task Migration and Replication
- Outlook: AI- and Simulation-Based Engineering at Exascale
- Conclusions

Intra-node: Task Affinity

Realization in OpenMP

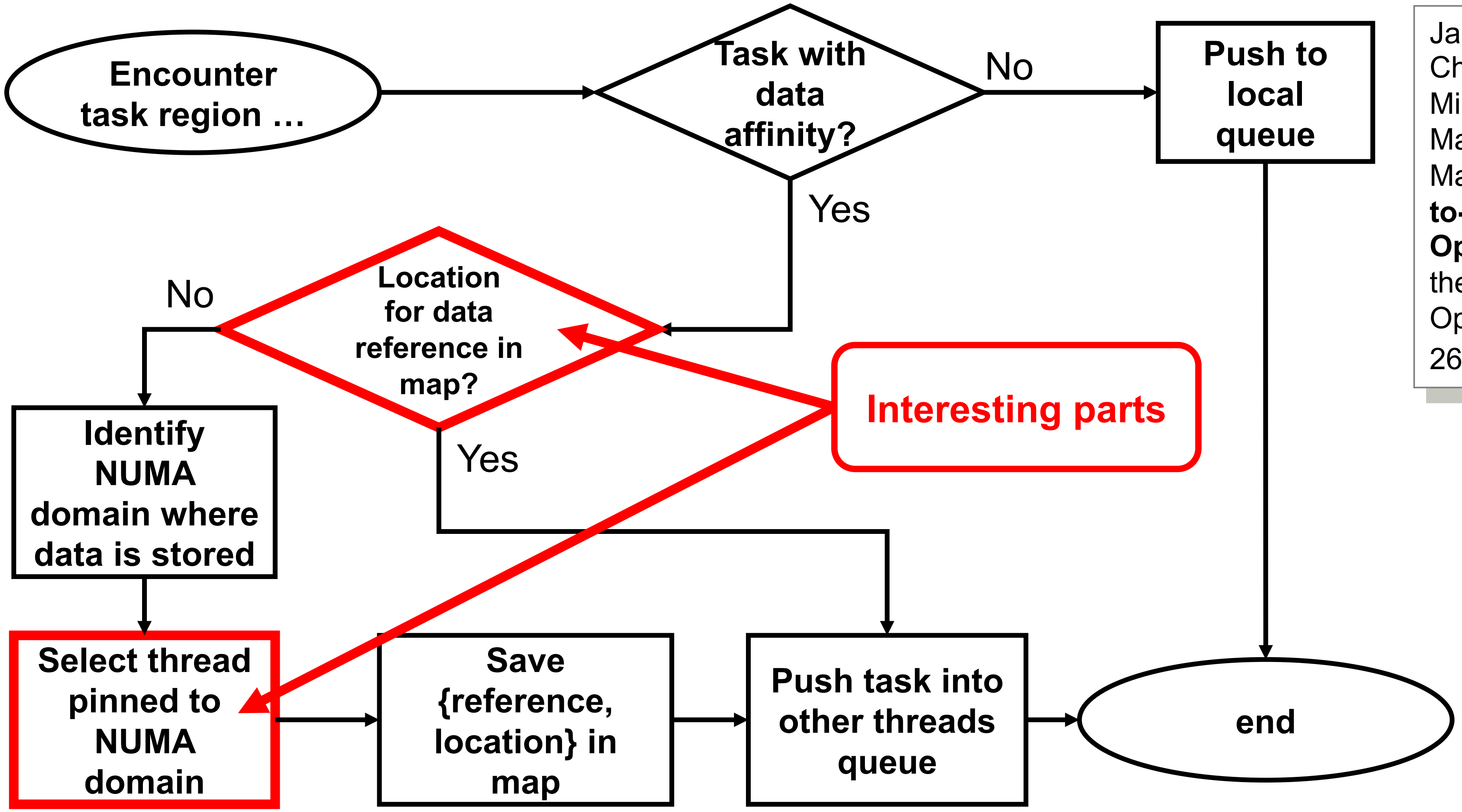
Support for task affinity is part of OpenMP 5.0 released on November 8th, 2019

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

```
int a[N];   // N is large
...
#pragma omp task affinity(a[x-y])
{
   // task that makes use of a[x], ...
}
```

- Programmer specifies data used by task
- Recommended to execute task closely to data location
  - Do not prohibit task stealing & load balancing
- Runtime identifies the location of the data and schedules task to a close thread
- Clear separation between dependencies and affinity

Selected implementation details



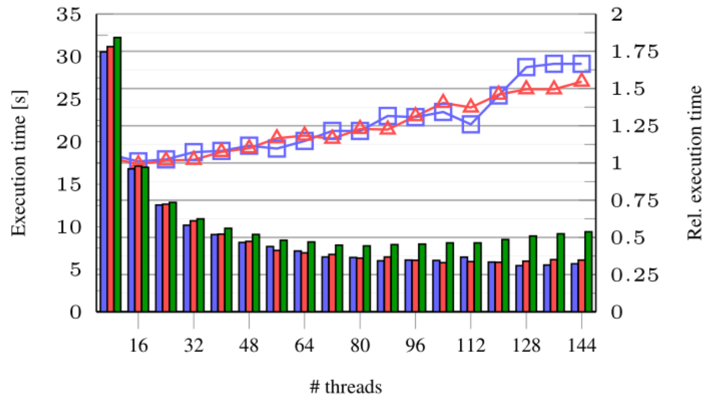
Encounter task region …
Task with data affinity?
No
Push to local queue
Yes
No
Location for data reference in map?
Interesting parts
Identify NUMA domain where data is stored
Yes
Select thread pinned to NUMA domain
Save {reference, location} in map
Push task into other threads queue
end

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.

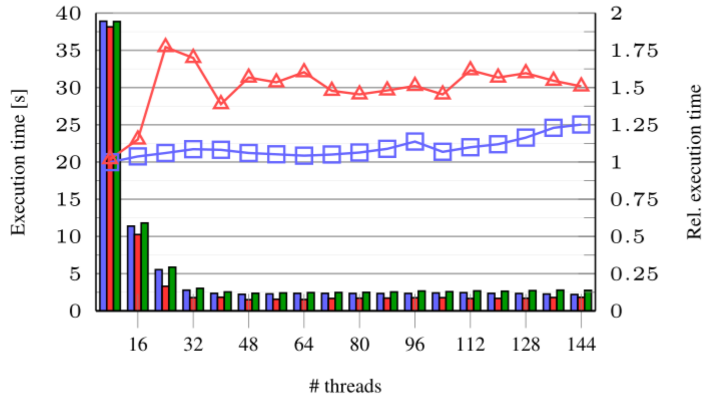
Evaluation with Applications

How much can this improve applications?

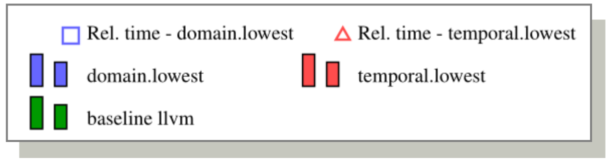
- Little improvements on standard 2-socket systems, more improvement on larger systems



(d) Merge sort on 8-socket



(h) Health on 8-socket



  - Works well working with a lot of data & single task creator scenarios & tasks created in parallel but not all close to data
  - Not much room for improvement when: parallel task creator scenarios & tasks are already created where data is located

Intel® Xeon® E7-8860v4 (codename Broadwell)
8 sockets, 18 cores per socket = 144 cores
2.2 GHz base frequency, 1 TB memory

Inter-node: Task Migration and Replication

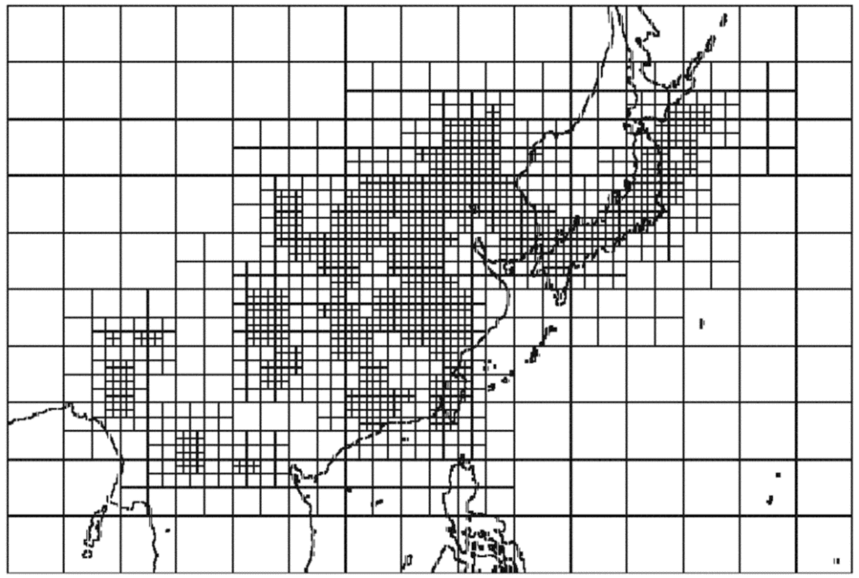
Motivation: Real-world code example

Dynamic variability caused by application

- Showcase application: **sam(oa)²**
  - Finite-Element and Finite-Volume simulations of dynamic adaptive meshes
  - Space Filling Curves (SFC) and Adaptive Meshes for Oceanic And Other Applications (Tohoku Tsunami 2011)
  - Developed at TU Munich

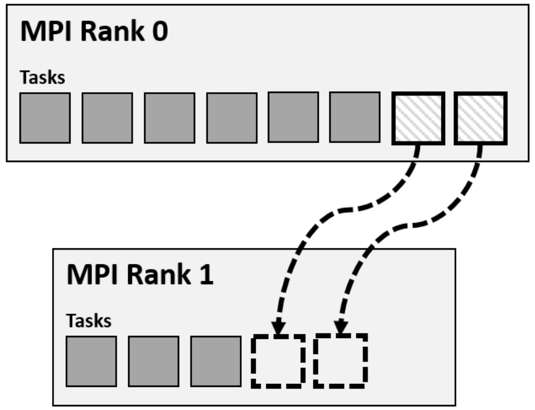
- Depending on situation either refinement or coarsening of cell / section

- Refinement leads to load imbalances
  - after each iteration
  - intra and inter node



Chameleon Approach: Migratable Tasks + Self Introspection

- **Migratable task**
  - Basic unit of work without side effects
  - Action + data items (input and/or output)
  - Can be executed locally or migrated to another rank



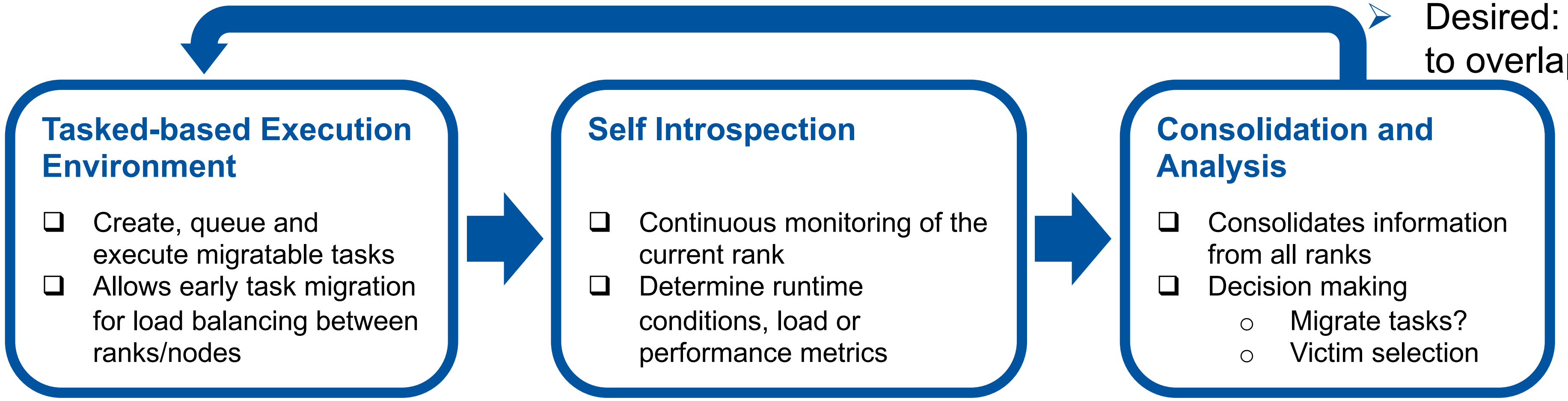
1. Based on periodically collected introspection data detect imbalance dynamically at runtime

   Result: Rank 0 is significantly slower or has more work

2. Migrate tasks and data to Rank 1

3. Prioritized execution of migrated tasks at Rank 1 + send back results or outputs

➢ Desired: Migrate as soon as possible to overlap communication and computation



Tasked-based Execution Environment

- ❑ Create, queue and execute migratable tasks
- ❑ Allows early task migration for load balancing between ranks/nodes

Self Introspection

- ❑ Continuous monitoring of the current rank
- ❑ Determine runtime conditions, load or performance metrics

Consolidation and Analysis

- ❑ Consolidates information from all ranks
- ❑ Decision making
  - Migrate tasks?
  - Victim selection

Results Experiments – SW-induced Imbalances with sam(oa)^2

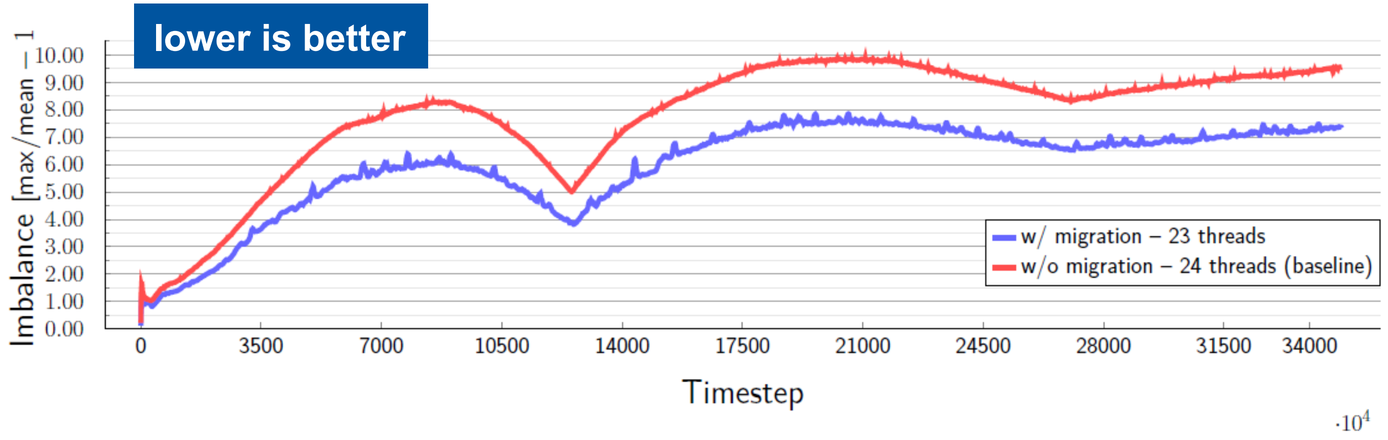


Figure 3: Load imbalances between ranks per time step in sam(oa)^2 for an application run with 32 nodes/ranks

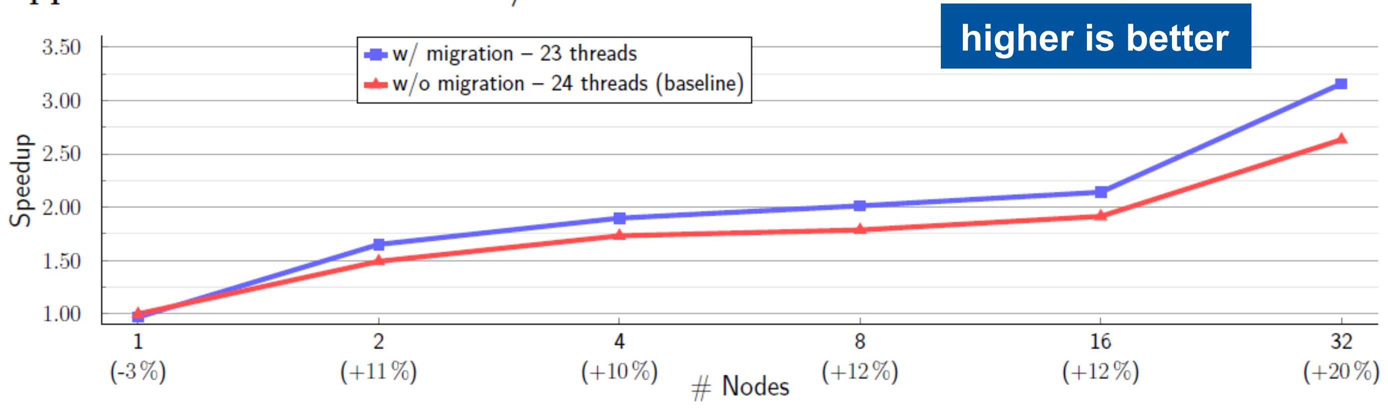


Figure 4: Strong scaling experiments with Tohoku tsunami in 2011 for complete application. Relative speedup to single node base line

- Simulated 60 minutes of Tohoku tsunami in 2011

➢ Reduce degree of imbalance

Outlook: AI- and Simulation-Based Engineering at Exascale

Challenges at Exascale

Tasking may be employed to provide efficient and scalable coupling of SW components

- CFD simulations cannot live without modeling approaches
  - Becomes worse in multi-physics and multi-scale phenomena, or with interactions such as combustion
  - Will be complemented with data-based models

- At Exascale, the amount of data may exceed the Exabyte range for single simulation runs
  - In-situ data reduction, extraction and interpretation will hence be unavoidable

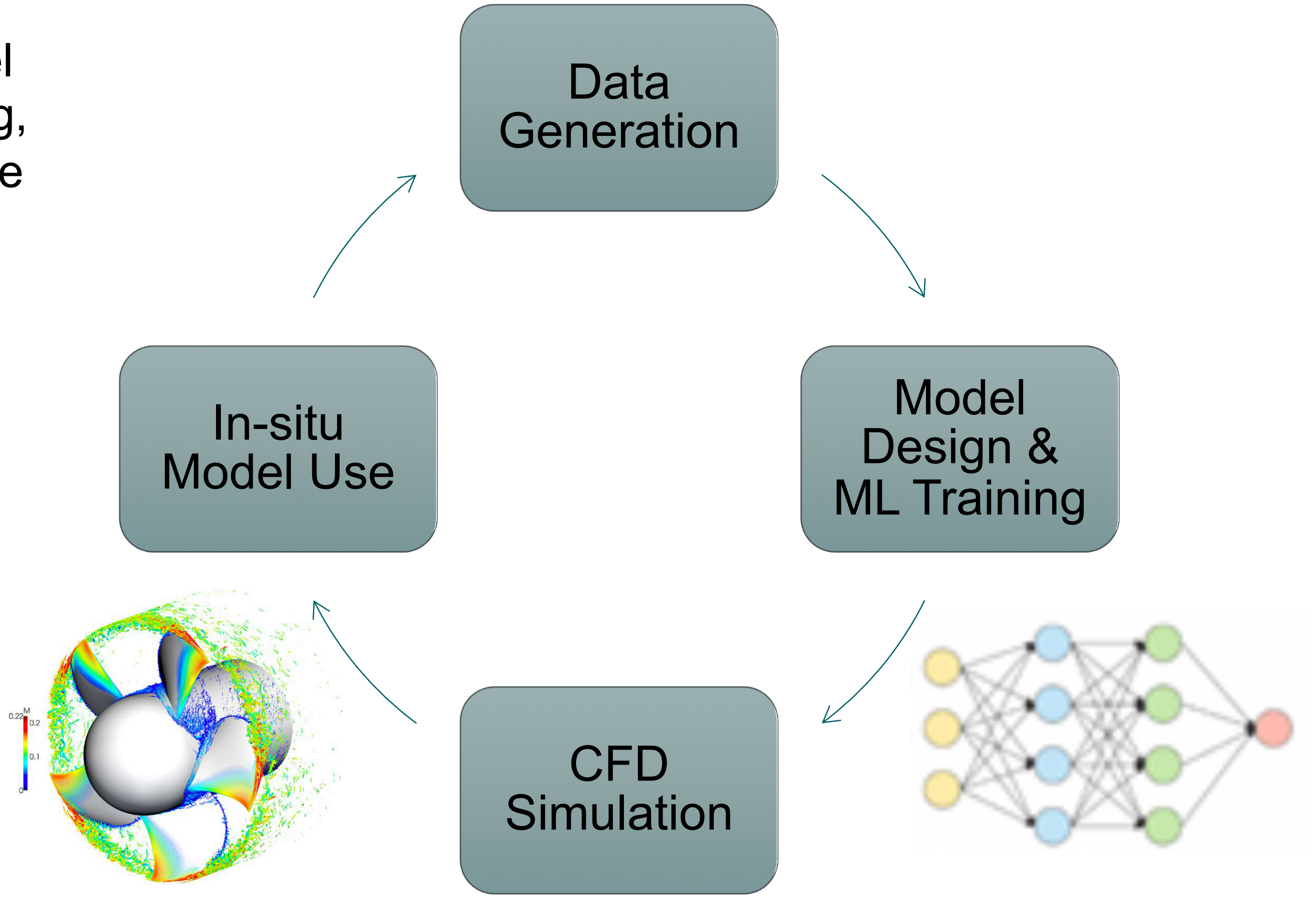
- To utilize HPC resources efficiently, software and workflows must scale to high CPU counts
  - In compute-drive applications, analyses are frequently a posteriori, necessitating to have the data on disk
  - As the field of parallel and scalable ML and DL is progressing, those algorithms become feasible to be intertwined with simulation codes implementing full loops

- FZJ’s Modular Supercomputing as a prominent heterogeneous pre-Exascale architecture

Challenges at Exascale

Tasking may be employed to provide efficient and scalable coupling of SW components

- Key expectation: As the field of parallel and scalable ML and DL is progressing, those algorithms become feasible to be intertwined with simulation codes implementing full loops



Conclusions

Reactive Task-parallel Programming

Tasking model is becoming more attractive

- Tasking brings advantages for dynamic systems
- Affinity brings performance improvements
  - Including support for complex memory hierarchies
- Reactive MPI+OpenMP task migration for fine-granular load balancing
  - Robustness against HW- and work-induced imbalances
- Key expectation: As the field of parallel and scalable ML and DL is progressing, those algorithms become feasible to be intertwined with simulation codes implementing full loops

Invitation to collaborate

- Future research direction: runtime work for intra-node and inter-node tasking
- Exchange with RIKEN expected to continue
- Also: see proposals from 2018 meeting

Vielen Dank
für Ihre Aufmerksamkeit