Programming Models and Tools: Programmer's Expectations

Christian Terboven <terboven@itc.rwth-aachen.de>

April 6th, 2017

Motivation

- In the EU CoE project POP, and the German DFG project ProPE, we are developing a standardized performance engineering approach

https://pop-coe.eu/

- Performance Audit -> Performance Plan -> Proof-of-Concept

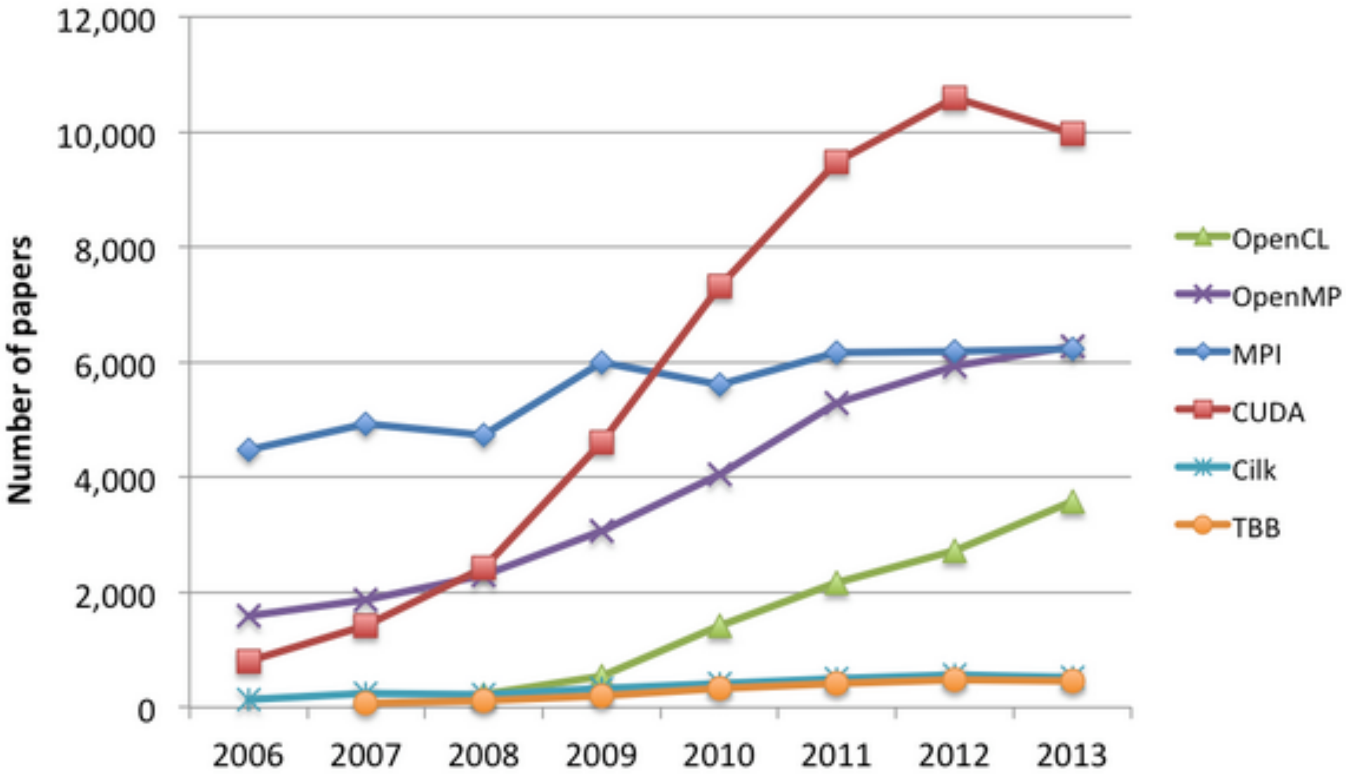
- In this talk, I will present my observations of what HPC users and HPC consultants expect from HPC programming models and tools

Agenda

- Popularity of Programming Models
- POP Performance Engineering Process
- Requirements for Models and Tools
- Conclusion

Programming Models

- **Popularity of Parallel Programming Models in terms of papers**



Papers mentioning parallel programming langages.
Data according to Google Scholar (Feb. 2014)

(c) Simon McIntosh-Smith 2014

my tag

GPU performance

THE standards

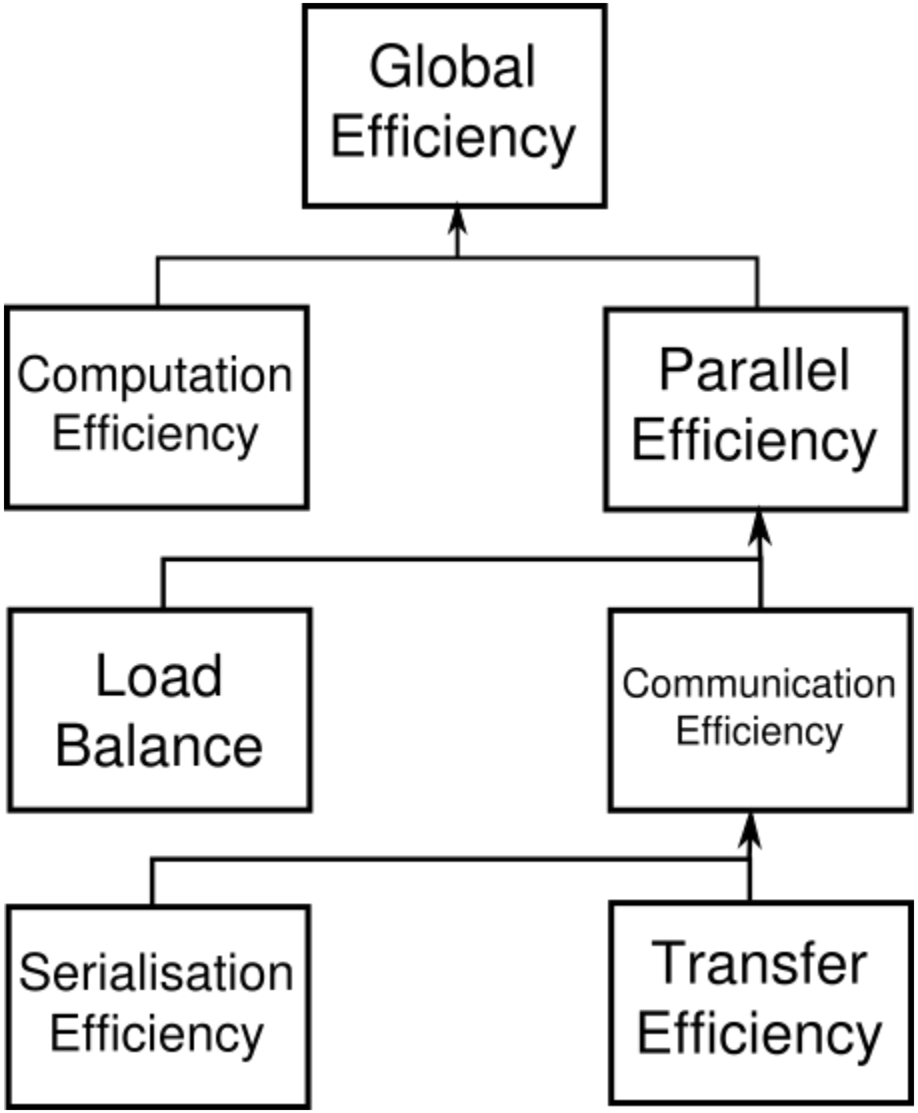
vendor neutrality

tasking

POP Performance Engineering Process

Performance Engineering

- **GE: overall efficiency**
  - → Overhead?
  - → Poor scaling?
- **PE: reveals inefficiency in splitting computations over processes**
  - → Uneven work distribution?
  - → Communication overhead?
- **CE: overall time in useful computation**
  - → Good IPC?

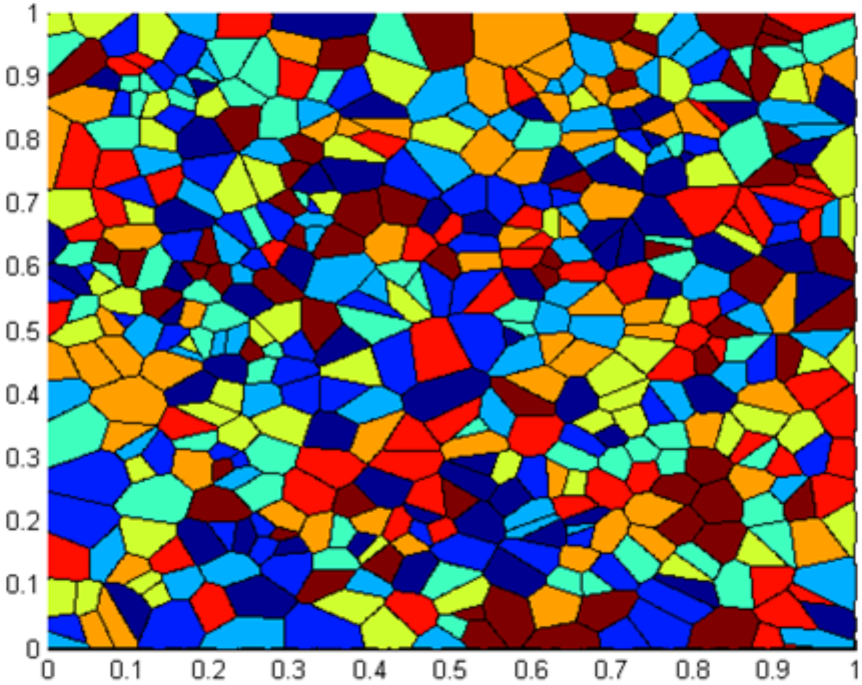


Optimization of RWTH IMM code GraGLe2D

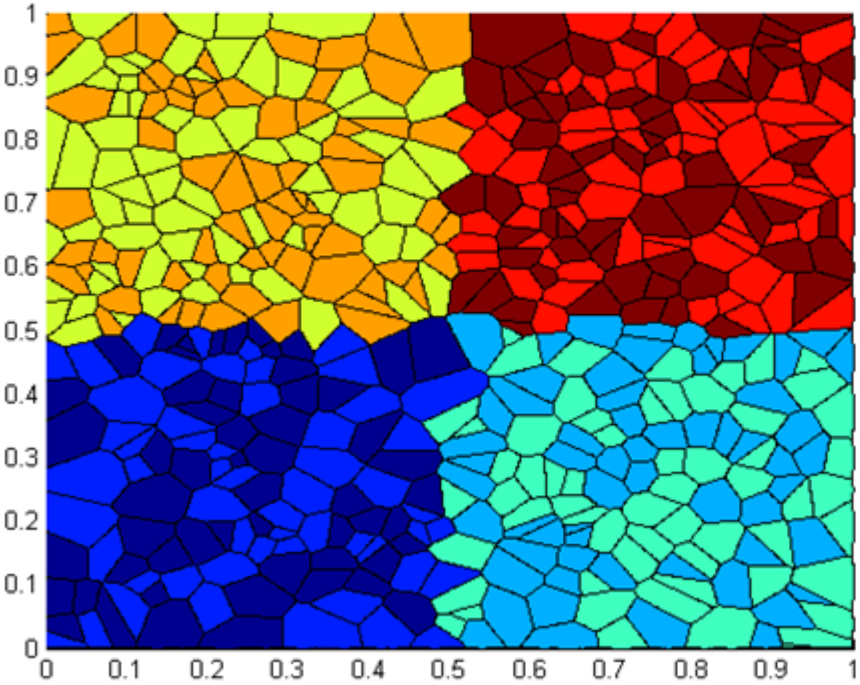
- **Performance issue: scalability of the OpenMP application over board boundaries on a big cc-NUMA-machine (BCS)**
- **Analyzed aspects:** Performance Engineering Process
  - → Thread binding → bound manually, with OpenMP Affinity model
  - → Load imbalance → minor load imbalance, not serious
  - → Data placement → suboptimal due to a potential sharing of a single memory page by threads on two sockets/boards
  - → Remote memory access → a lot, due to suboptimal work distribution
  - → Serial operations → suboptimal, lots of unnecessary arithmetic operations
- **Optimizing strategies:** Current Research: Standardization
  - → Using a scalable malloc routine instead of the system default malloc
  - → Improve the load distribution: Adjacent loads processed by adjacent threads
  - → Eliminate redundant/expensive operations, such as div/sqrt
  - → (Redesign the algorithm)
  - → Vectorizing loops manually

Achieved performance (1)

- **Load distribution:**
  - → Work binding to threads, before and after optimization
  - → Similar color: adjacent threads



Before



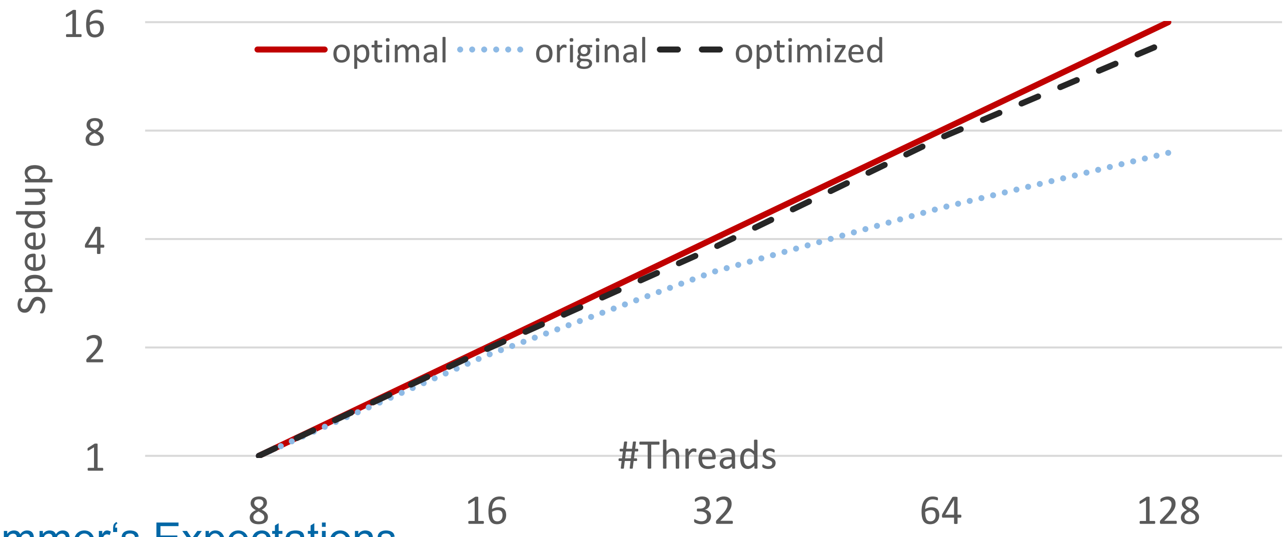
After

Achieved performance (2)

- **Runtime using 128 threads**

	Application	Parallel Regions
Original run time(s)	341.49	212.41
Optimized run time(s)	154.68	33.15
Speedup	2.2	6.4

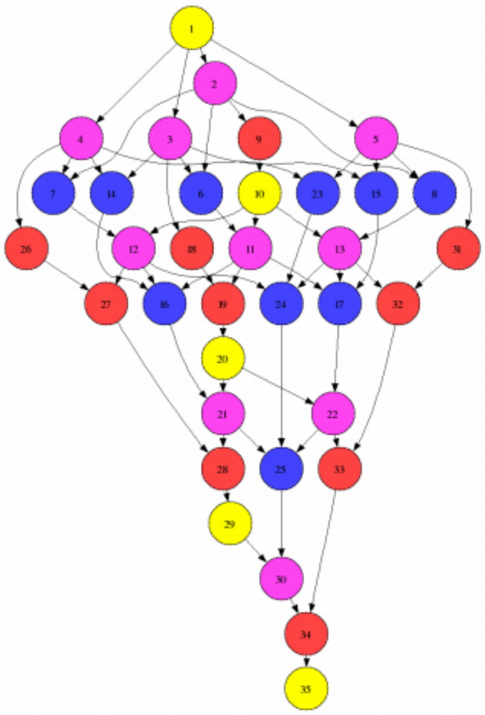
- **Scalability of parallel regions:**



Requirements for Programming Models and Tools

Tool Support

- **Support for Performance Analysis**
  - → Following the PE process
  - → Analyzing architecture-specific behavior
  - → Differentiating between application and programming model
- **Insight into Innovative Features**
  - → How to use them correctly
  - → Example: Tasking
    - → Granularity of Tasks?
    - → Use of cut-off mechanism?
- **Support for Correctness Checking?**



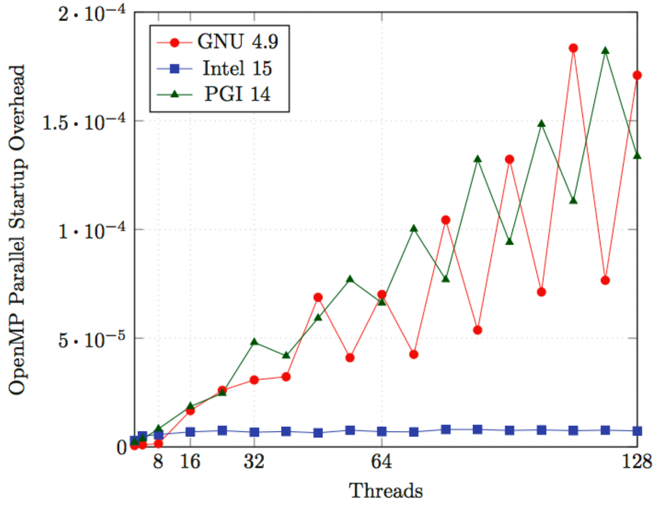
* image from BSC

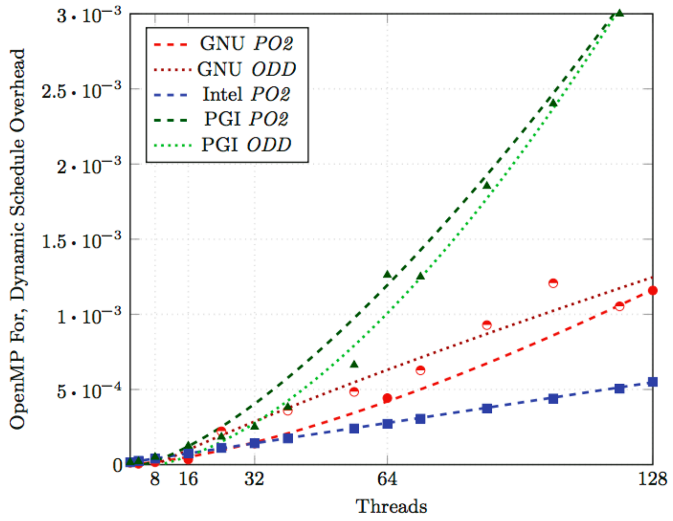
Feature Support

- **Programmers have a hard time coping with ”new” memory types**
  - → Transactional Memory
    - →OpenMP’s solution: annotated locks and critical regions
  - → Locality / Memory Affinity
  - → Non-volatile Memory
    - →OpenMP’s solution: memory management API (in development)
    - →See TR5: www.openmp.org/…
- **Abstractions improve productivity**
- **Standards + Standard Interface**
  - → SPPEXA project MYX -> XMPT
  - → Our group also contributes to OMPT (the OpenMP Tools Interface)

Insight + Reproducibility + Reliability

- **Features (abstractions) should work on a range of architectures**
  - → Difference between theory and praxis: scaling of primitives
  - → Bad example: overhead of OpenMP Parallel Region startup: high differences between implementations even on the same system
  - → Similar: overhead of dynamic loop scheduling
- **Do not break things with updates**





Conclusion

Programmer's Expectations

- **From yesterday's discussion: Application Analysis and Tuning is a never-ending process**
  - → Ongoing refinement of the Performance Engineering (PE) Process
  - → Opportunities for automatization?
- **High Quality tools needed for the PE process**
  - → Opportunities for improvement
- **Lessons from applying the PE process**
  - → The number of programming models employed is increasing
  - → Programming Models have to be complemented with tools
  - → Programming Models should be "reliable"

Thank for your attention.

Christian Terboven <terboven@itc.rwth-aachen.de>

RWTH AACHEN UNIVERSITY