Productivity with current HPC programming models

Matthias S. Müller, RWTH Aachen University
VI-HPS 10th Anniversary Workshop

June 23, 2017 Seeheim

IT Center | RWTH AACHEN UNIVERSITY

Motivation and Disclaimer

- Parallel programming is *hard*
- We need to be scientific about solving these problems
- We would all like parallel programming to be eaiser and more fun, but to accomplish that, we need to focus on the real problems

Comparing two programming models:

First – are you comparing programming *models*, programming *systems*, or *implementations* of programming systems?

- • Answer – Almost always implementations
- • Implication – No paper should be accepted that claims to compare X to Y when all it does is compare an implementation of X on Z to an implementation of Y on Z

Source: Bill Gropp “Thinking about parallelism and programming” SC2016

Disclaimer II and more information

Productivity and Software Development Effort Estimation in High-Performance Computing

Der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University vorgelegte Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften von

Sandra Juliane Wienke, Master of Science

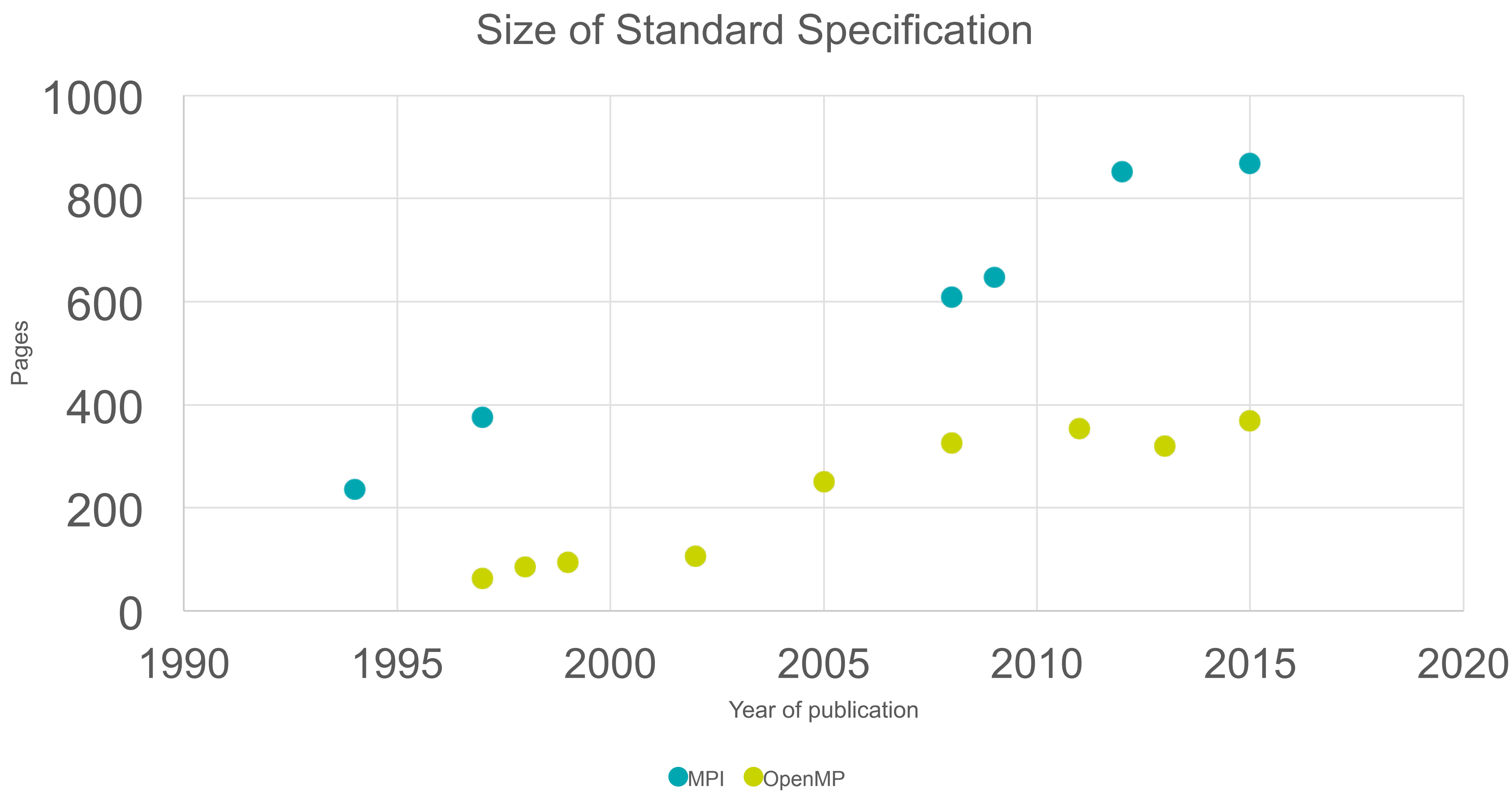
aus Berlin-Wedding

- See chapter 10 of Sandra Wienke's thesis: "Methodology of Development Effort Estimation in HPC"
- However:
  - Solid statistics to make valid statements about real programming model is not available

Some personal opinion about the productivity of programming models

Size of the standard

Evolution of MPI and OpenMP Standard



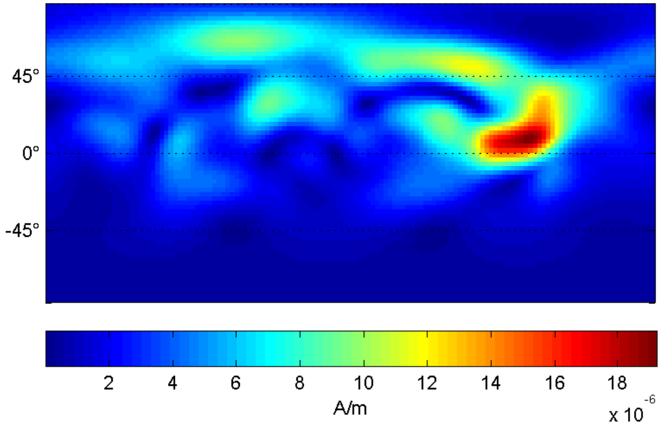
OpenMP vs MPI

1 : 0

Productivity

Case Study: NINA [1]

- Software* for the solution of Neuromagnetic INverse lArge-scale problems
- Implementation
  - Basis: serial C code
  - OpenMP-tuned: blocked matrix-vector multiplication, vectorization, alignment on pages, data affinity
  - OpenMP-target: OpenMP-tuned (adapted to KNC) + target directives for offloading
  - OpenACC: up to 16 streams for parallel async. execution of kernels, pinned memory
  - CUDA: up to 16 streams, dynamic parallelism and completely asynchronous execution to minimize interaction with host, highly optimized reduction, pinned memory



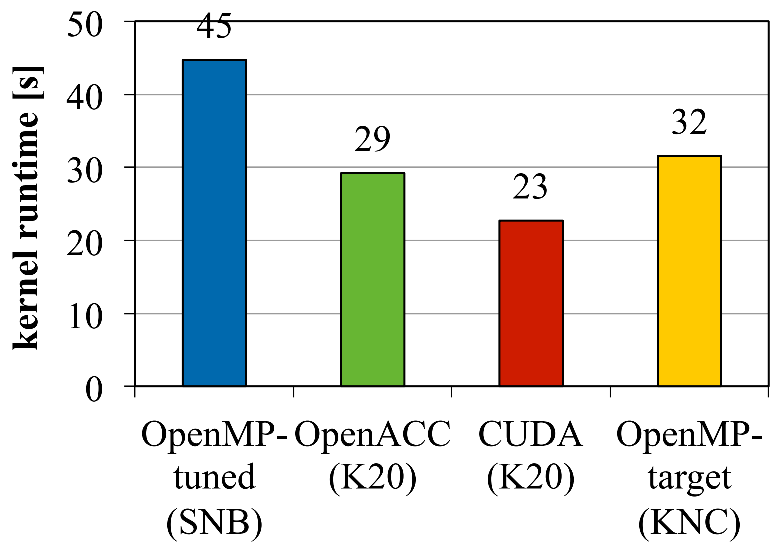
* M. Bücker, R. Beucker, and A. Rupp. Parallel Minimum p-Norm Solution of the Neuromagnetic Inverse Problem for Realistic Signals Using Exact Hessian-Vector Products. SIAM Journal on Scientific Computing, 30(6): 2905–2921, 2008.

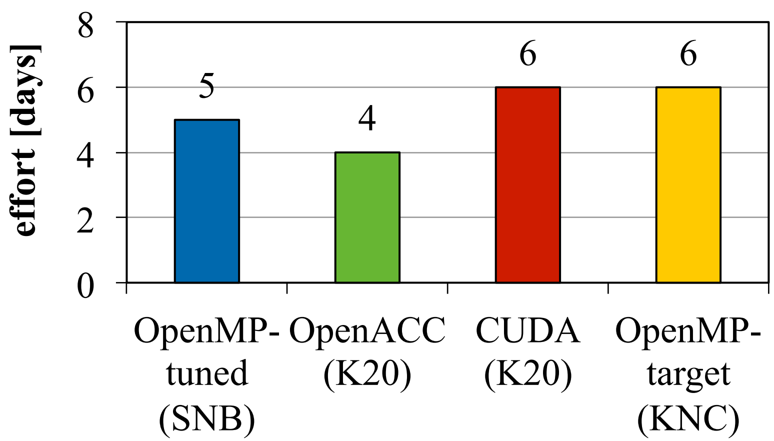
[1] Wienke, Sandra, Dieter an Mey, and Matthias S. Müller. "Accelerators for Technical Computing: Is It Worth the Pain? A TCO Perspective." *Supercomputing*, 2013, 330-42.

NINA – TCO[7]

- One-time costs C_{ot}
  - Per node
    - HW purchase: Bull list prices from 2013 (!)
    - Building/infrastructure: as annual costs since it is amortized over 25 years
    - OS/env. installation: -
  - Per node type
    - OS/env. installation: -
    - Programming effort: Full-time employee costs 272.86 € a day
- Annual costs C_{pa}
  - Per node
    - HW maintenance: 8.2% of HW purchase costs
    - Building/infrastructure: 200,000€ per year, divided by 1.6MW, multiplied by max. power consumption of each node
    - OS/env. maintenance: 4 admins, 75% maintenance cluster (~2300 nodes): 180,000€ / 2300 = 78€ per node and year
    - Power consumption: PUE 1.5, regional electricity costs 0.15 €/k
  - Per node type
    - OS/env. maintenance: -
    - Software/compiler: -
    - Application maintenance: - (small kernels)

NINA – Effort & Performance

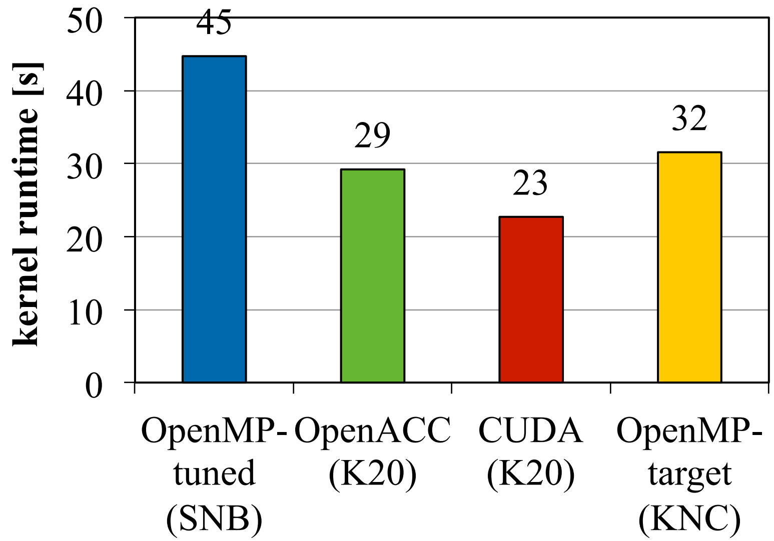


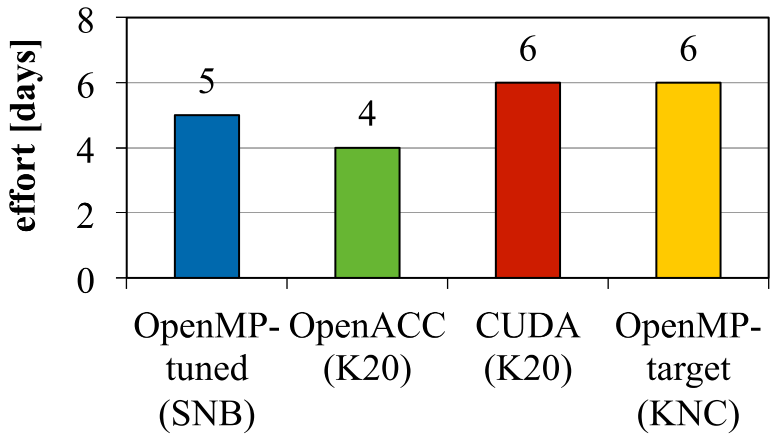


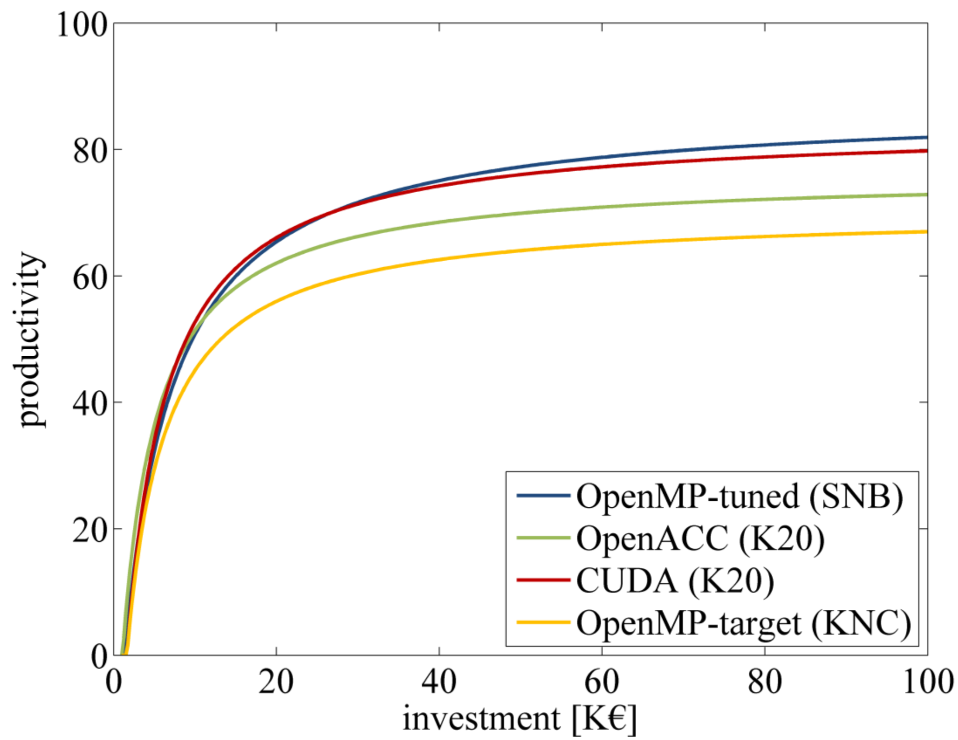
	Hardware
SNB	2-socket Intel Sandy Bridge @ 2.7 GHz, 16 cores
K20	SNB + 1 NVIDIA Kepler K20x
KNC	SNB + 1 Intel Xeon Phi 5110p

$$\text{productivity} = \text{outputs} / \text{inputs} = \sum \textit{app runs} \,/\, \text{TCO}[\$]$$

NINA – Productivity







$$\text{productivity} = \text{outputs}/\text{inputs} = \sum \textit{app runs} \,/\, \text{TCO}[\$]$$

OpenMP vs MPI

2 : 0

Suitability for Exascale

Suitability of Programming Models for Exascale

Exascale Concepts for Programming Models
Session at ISC 2016, June 19, 2016, Frankfurt

ExaGASPI
04:00 pm - 04:20 pm
Mirko Rahn, Fraunhofer ITWM

MPI+X for Exascale
04:20 pm - 04:40 pm
Bill Gropp, University of Illinois at Urbana-Champaign

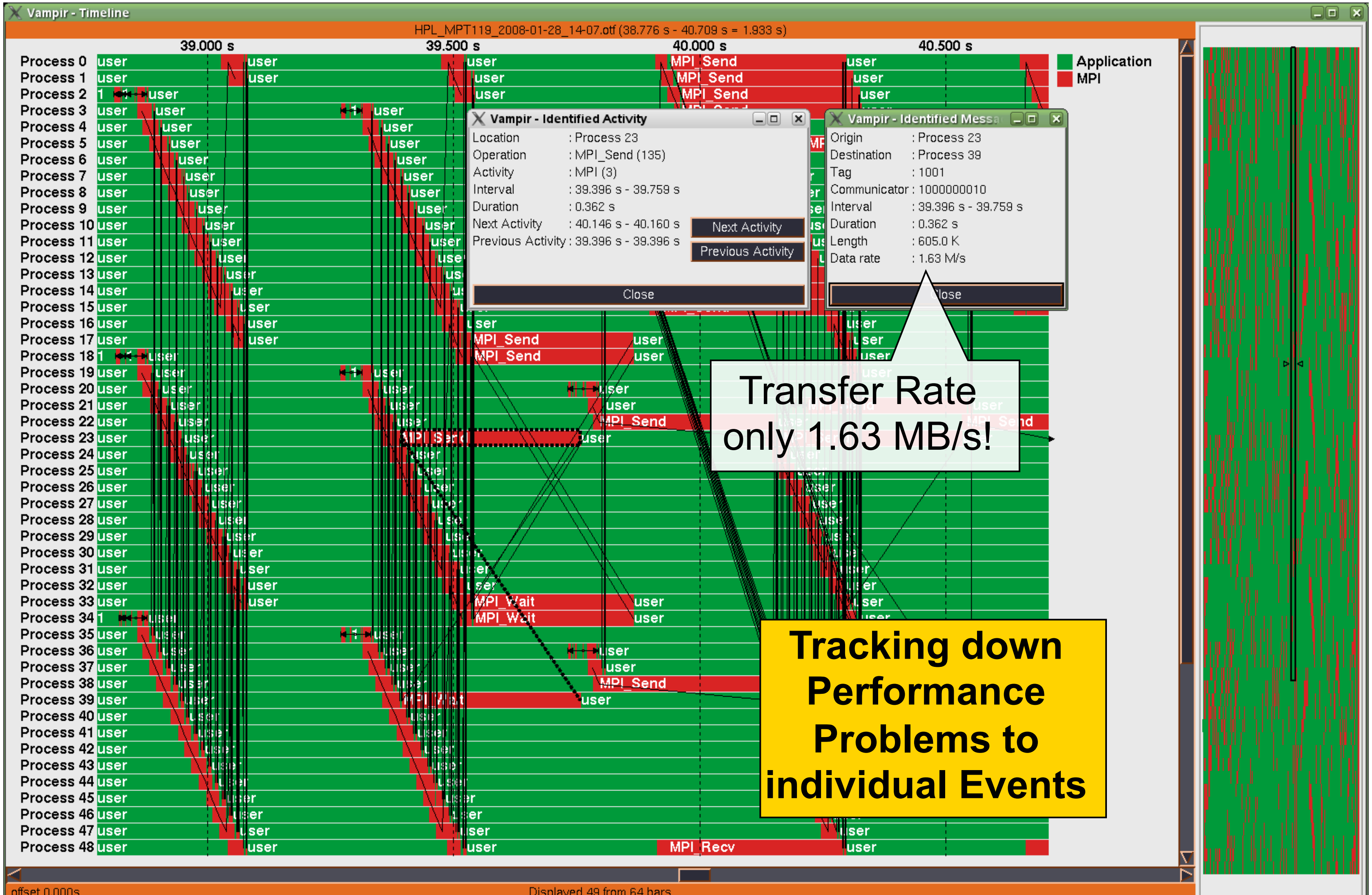
OpenMP - Taking Good Care of the Node in Exascale?
04:40 pm - 05:00 pm
Christian Terboven, RWTH Aachen University

OpenMP vs MPI

3 : 1

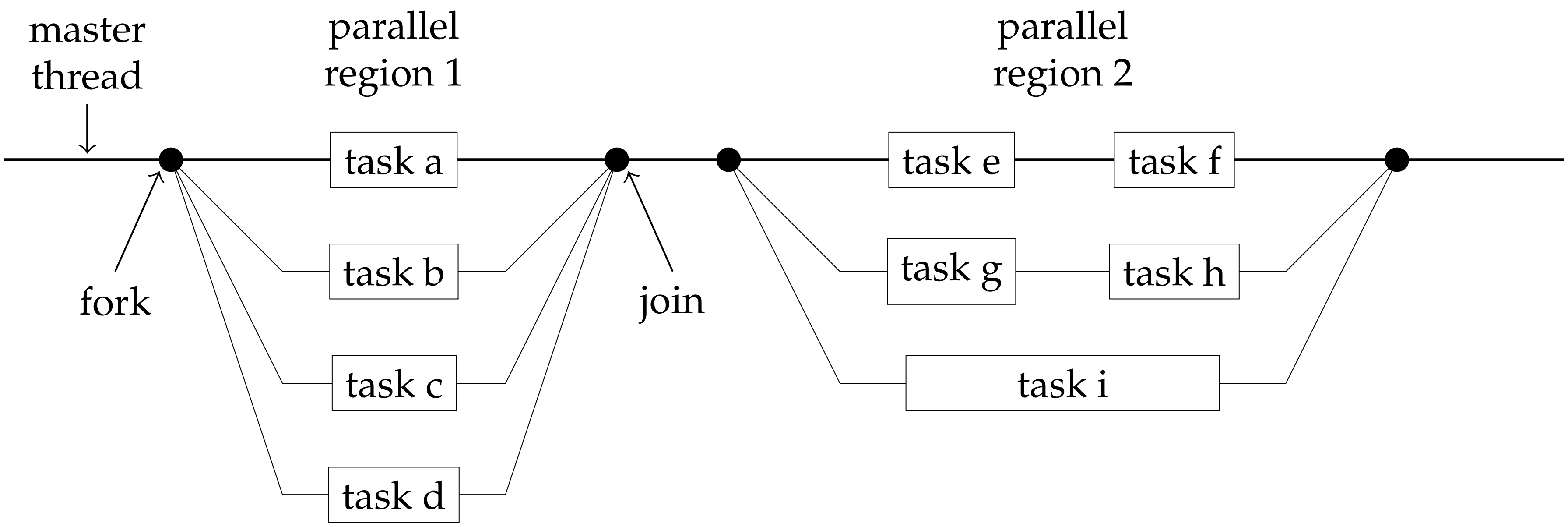
Complexity of Programming Model

HPL on SGI ICE using SGI MPT



OpenMP complexity

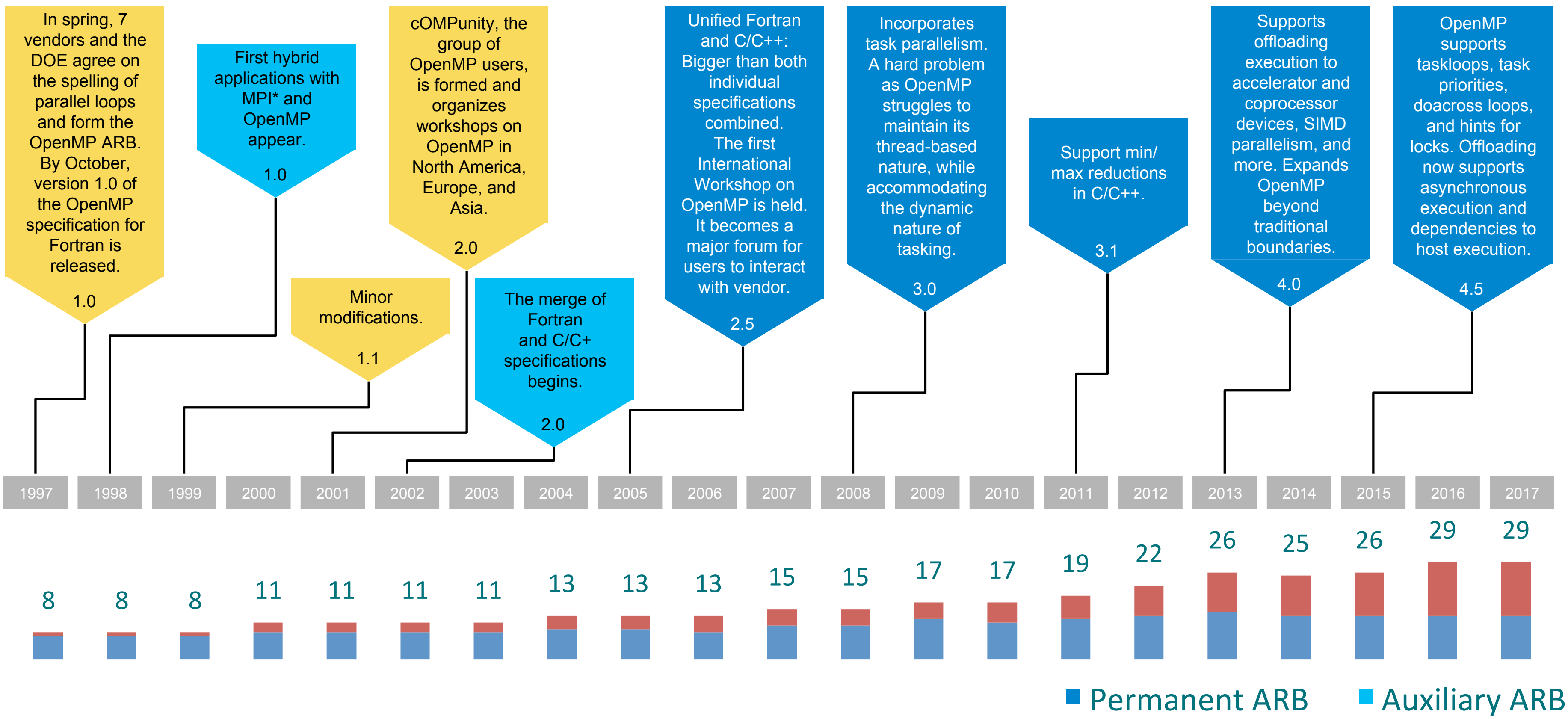
Fork-join model



OpenMP vs MPI

4 : 1

20 Years of OpenMP® History

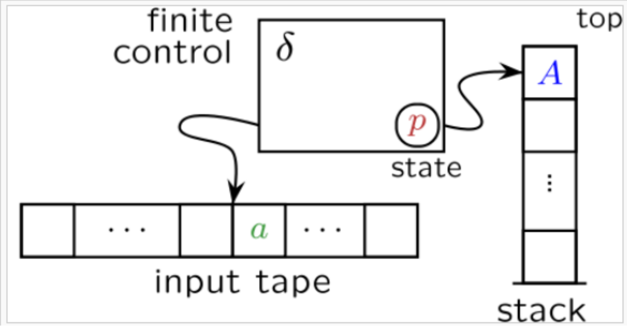


Thread-local Epoch Generation (1/2)

Push Down Automaton (PDA)

- Transition depends on input symbol and stack
- PDA (stack) is required to describe the semantic of OpenMP
- In addition: output alphabet (Mealy)

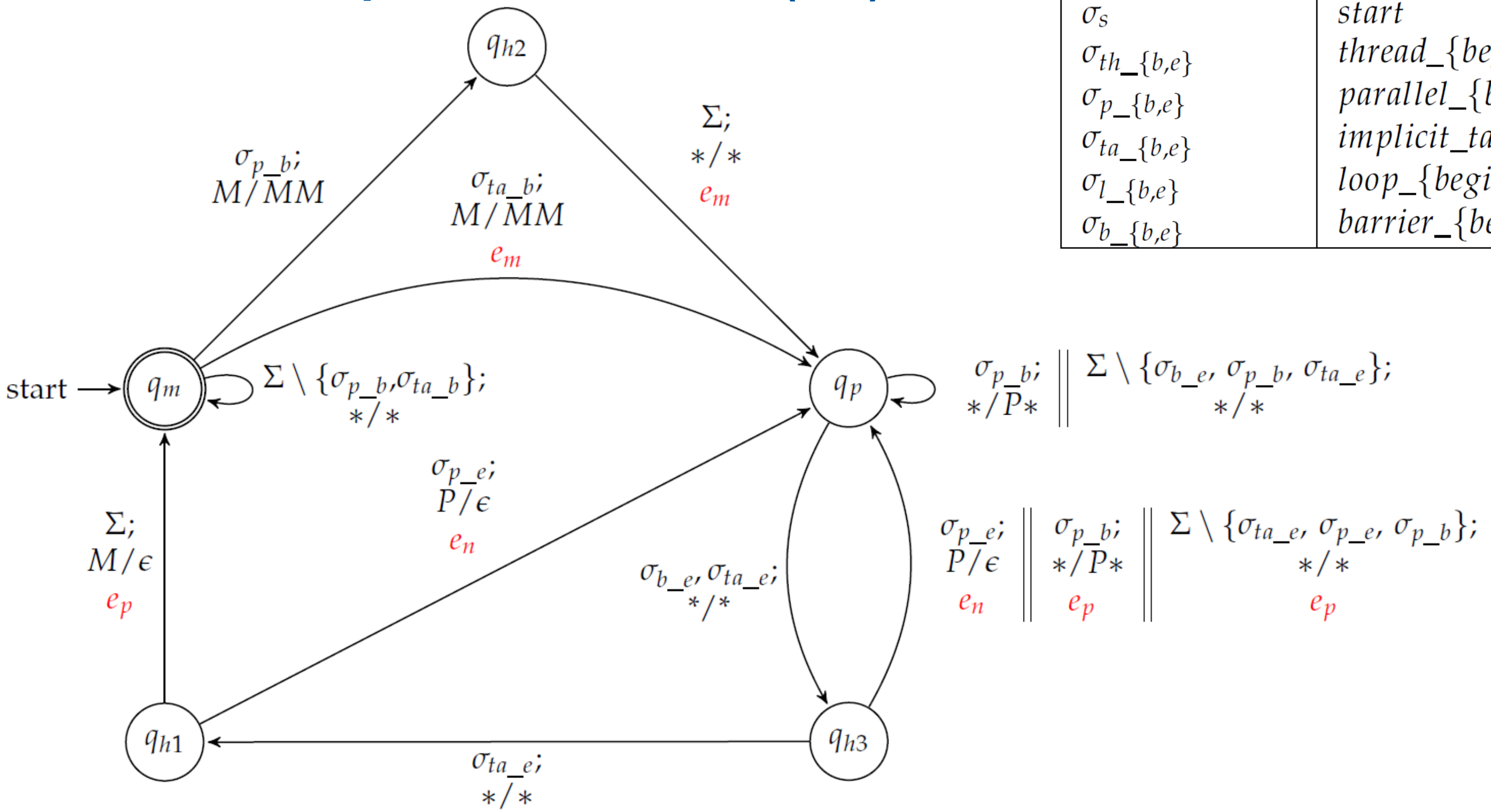
- 9-tuple: $M=(Q, \Sigma, \Omega, \Gamma, \delta, \lambda, q_0, Z, F)$, with
  - $\boldsymbol{Q=\{q_m, q_p, q_{h1}, q_{h2}, q_{h3}, q_s\}}$ **set of states**
  - $\boldsymbol{\Sigma}$ **set of OMPT events (input alphabet)**.
  - $\boldsymbol{\Omega=\{e_m, e_p, e_n, \epsilon\}}$ **output alphabet** (e.g. master epech, parallel epoch)
  - $\boldsymbol{\Gamma=\{\mathrm{M}, \mathrm{P}, \epsilon\}}$ **stack alphabet.**
  - $\delta = Q \times \Sigma \times \Gamma \rightarrow \mathrm{P}(Q \times \Gamma)$ transition
  - $\lambda = Q \times \Sigma \times \Gamma \rightarrow \Omega$ output function.
  - $q_0 = q_m$ initial state
  - $Z = M$ initial stack symbol ist
  - $F = \{q_m\}$ set of accepting states.



Quelle: Wikipedia

Thread-local Epoch Generation (2/2)

Input Symbol	OMPT Event
σ_s	*start*
$\sigma_{th\_\{b,e\}}$	*thread_{begin, end}*
$\sigma_{p\_\{b,e\}}$	*parallel_{begin, end}*
$\sigma_{ta\_\{b,e\}}$	*implicit_task_{begin, end}*
$\sigma_{l\_\{b,e\}}$	*loop_{begin, end}*
$\sigma_{b\_\{b,e\}}$	*barrier_{begin, end}*

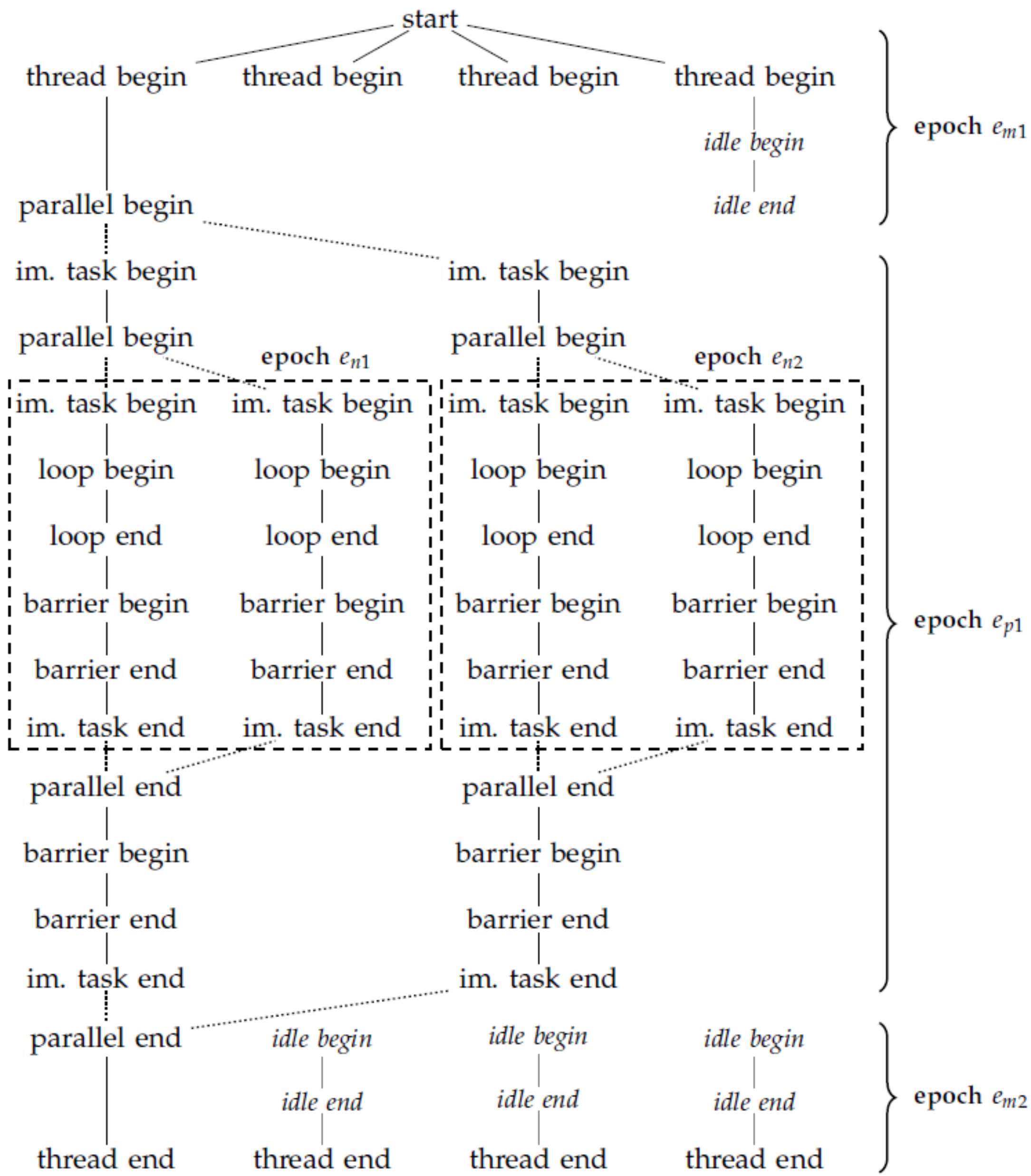


- First row: Input symbol
- Second Row: Stack operation pop(a) / push(*b*
- Third row(falls vorhanden): output symbiol.

PDA calculation

- Example with nested parallelism:

```
#pragma omp parallel
{
  #pragma omp parallel for
  for(...)
  {
    foo(…);
  }
}
```

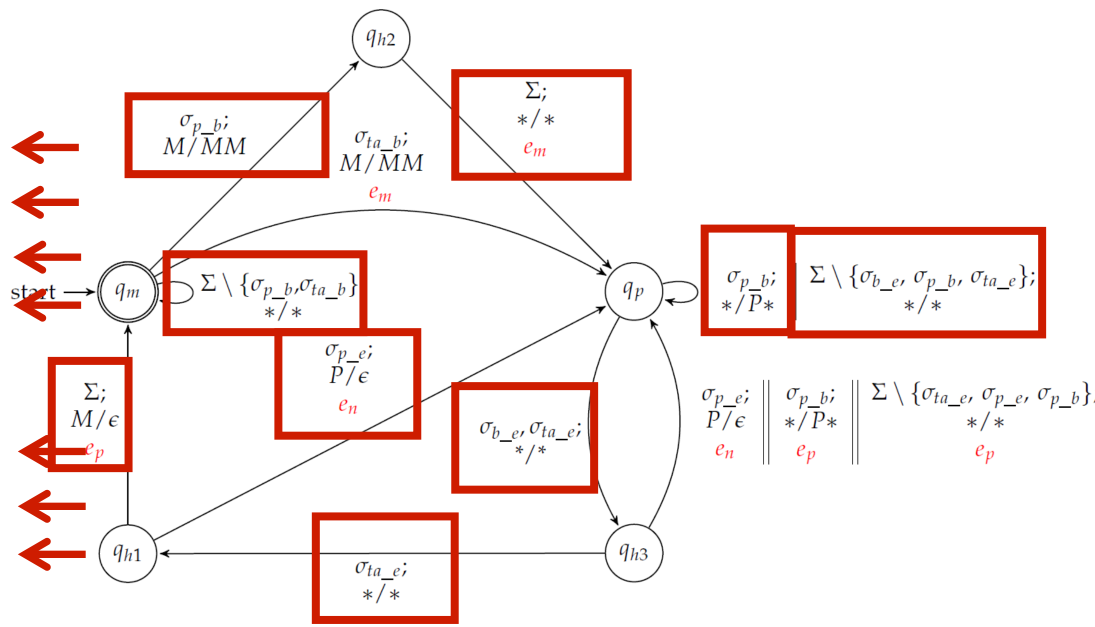


PDA Computation (2/2) (master thread only)

- Nested parallel example code:

```
#pragma omp parallel
{
  #pragma omp parallel for
  for(...)
  {
    foo(…);
  }
}
```

start

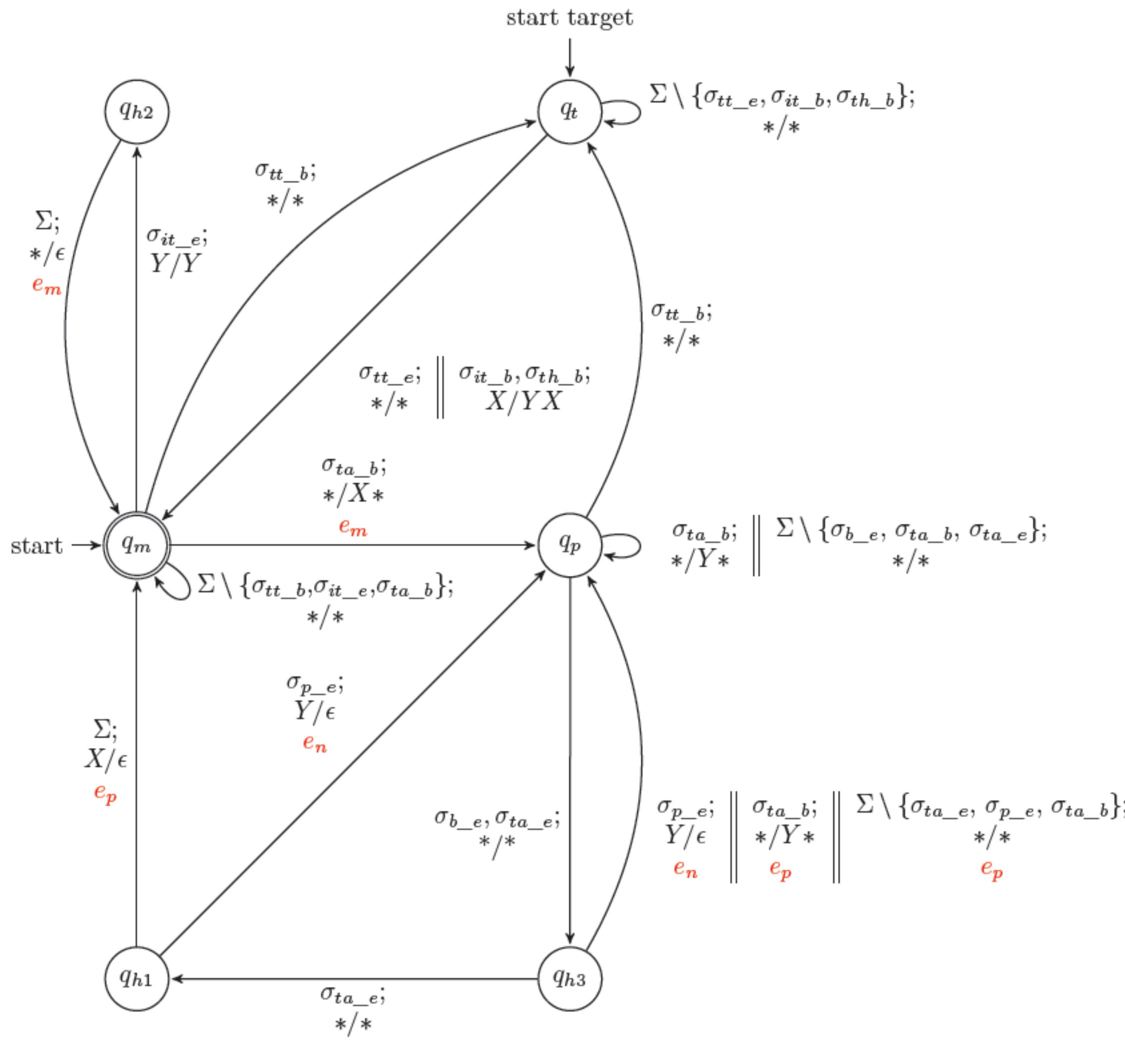


Γ	M	M	M	M M	M M	P M M	P M M	P M M	P M M	P M M	P M M	P M M	M M	M M	M M	M M	M	M
Q	q_m	q_m	q_m	q_{h2}	q_p	q_{h4}	q_p	q_p	q_p	q_p	q_{h3}	q_{h1}	q_p	q_p	q_{h3}	q_{h1}	q_m	q_m
λ					e_m								e_n				e_p	
Σ	σ_s	σ_{th_b}	σ_{p_b}	σ_{ta_b}	σ_{p_b}	σ_{ta_b}	σ_{l_b}	σ_{l_e}	σ_{b_b}	σ_{b_e}	σ_{ta_e}	σ_{p_e}	σ_{b_b}	σ_{b_e}	σ_{ta_e}	σ_{p_e}	σ_{th_e}	
	e_{m1}							e_{n1}	e_{p1}								e_{m2}	

start, thread begin, parallel begin, im. task begin, parallel begin, im. task begin, loop begin, loop end, barrier begin, barrier end, im. task end, parallel end, barrier begin, barrier end, im. task end, parallel end, thread end

Complete PDA for OpenMP

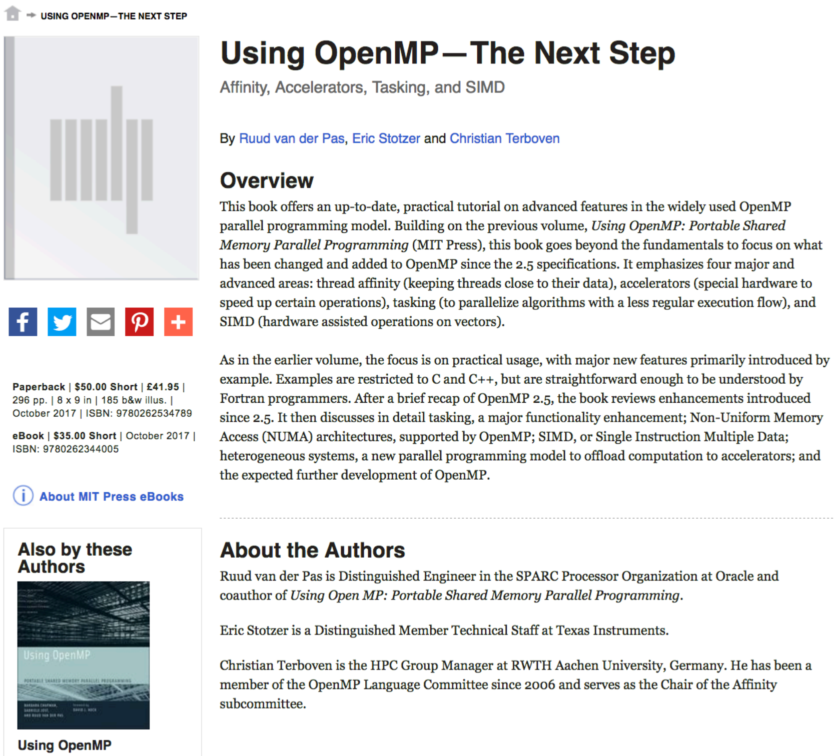
- Additional constructs:
  - Tasking
  - Target Offloading
- More information:
  Tim Cramer, “Analyzing Memory Accesses for Performance and Correctness of Parallel Programs”, PhD. Thesis, Aachen 2017



OpenMP vs MPI

$\cancel{4}$ 3 : 1

Multiparadigm programming and OpenMP



USING OPENMP—THE NEXT STEP

Using OpenMP—The Next Step

Affinity, Accelerators, Tasking, and SIMD

By Ruud van der Pas, Eric Stotzer and Christian Terboven

Overview

This book offers an up-to-date, practical tutorial on advanced features in the widely used OpenMP parallel programming model. Building on the previous volume, *Using OpenMP: Portable Shared Memory Parallel Programming* (MIT Press), this book goes beyond the fundamentals to focus on what has been changed and added to OpenMP since the 2.5 specifications. It emphasizes four major and advanced areas: thread affinity (keeping threads close to their data), accelerators (special hardware to speed up certain operations), tasking (to parallelize algorithms with a less regular execution flow), and SIMD (hardware assisted operations on vectors).

As in the earlier volume, the focus is on practical usage, with major new features primarily introduced by example. Examples are restricted to C and C++, but are straightforward enough to be understood by Fortran programmers. After a brief recap of OpenMP 2.5, the book reviews enhancements introduced since 2.5. It then discusses in detail tasking, a major functionality enhancement; Non-Uniform Memory Access (NUMA) architectures, supported by OpenMP; SIMD, or Single Instruction Multiple Data; heterogeneous systems, a new parallel programming model to offload computation to accelerators; and the expected further development of OpenMP.

Paperback | **$50.00 Short** | **£41.95** | 296 pp. | 8 x 9 in | 185 b&w illus. | October 2017 | ISBN: 9780262534789

eBook | **$35.00 Short** | October 2017 | ISBN: 9780262344005

About MIT Press eBooks

About the Authors

Ruud van der Pas is Distinguished Engineer in the SPARC Processor Organization at Oracle and coauthor of *Using Open MP: Portable Shared Memory Parallel Programming.*

Eric Stotzer is a Distinguished Member Technical Staff at Texas Instruments.

Christian Terboven is the HPC Group Manager at RWTH Aachen University, Germany. He has been a member of the OpenMP Language Committee since 2006 and serves as the Chair of the Affinity subcommittee.

Also by these Authors

Using OpenMP

- OpenMP now supports a lot of different paradigms:
  - Threading
  - Tasking
  - Offloading
  - …
- This is good, but complicated
- Possible rescue:

  We have to teach multiparadigm programming [1]

[1] Bjarn Stroustrup: “Multiparadigm programming is a fancy way of saying ``programming using more than one programming style, each to its best effect.” (Bjarn Stroustrup. FAQ)

Conclusion and outlook

- Both OpenMP and MPI are on track for Exascale
- The size and complexity of both standards are troublesome
- Multiparadigm programming is important to maintain/achieve productivity
- Programmers productivity should get more attention when developing programming models and standards

In direct comparison of the productivity of OpenMP vs. MPI:

OpenMP is the clear winner!!

.. and MPI is its best friend

Vielen Dank
für Ihre Aufmerksamkeit