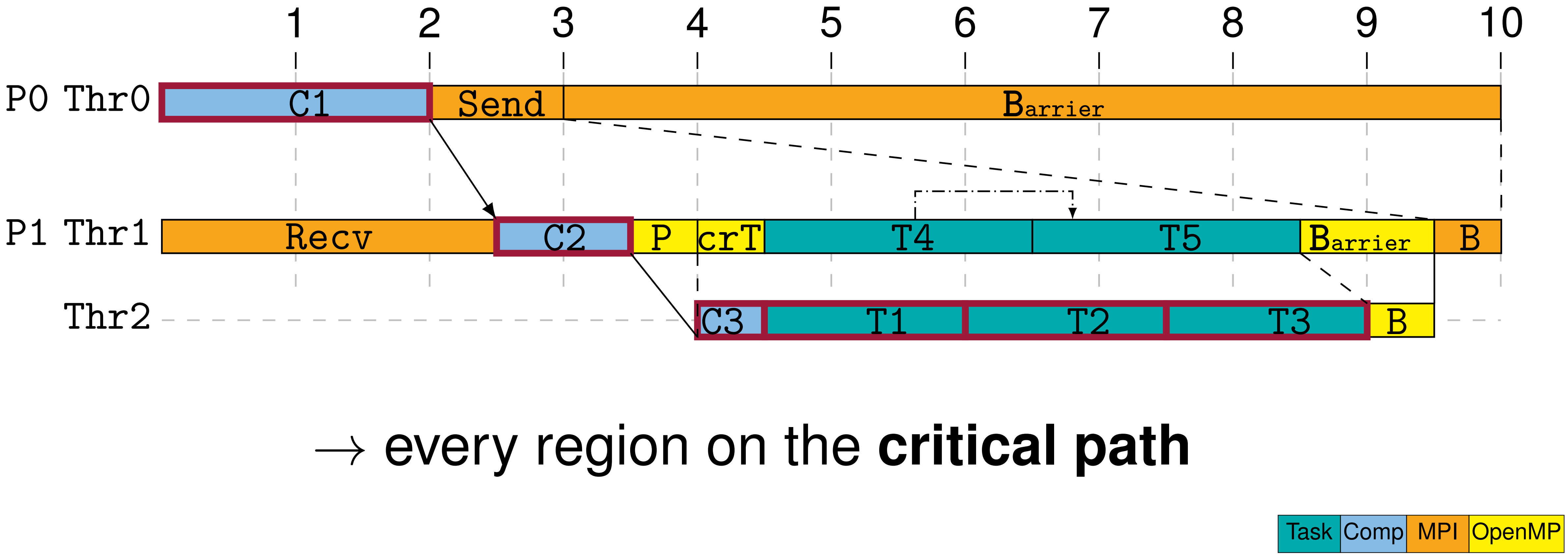
OTF-CPT: Application Insights Gained from Real-time Critical Path Analysis

Ben Thärigen, Joachim Jenke, Tobias Dollenbacher, Fabian Orland
RWTH Aachen University

September 19, 2024
15. International Parallel Tools Workshop Dresden 2024

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Who's responsible for the execution time?

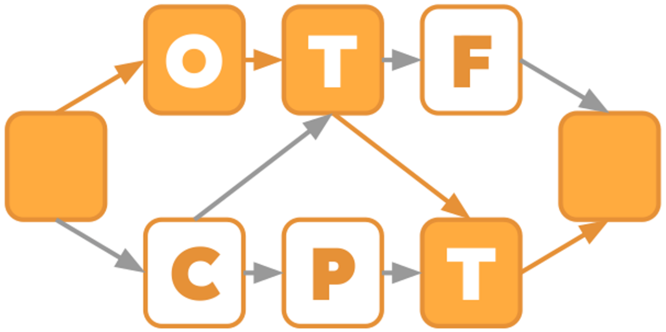


→ every region on the **critical path**

Motivation

- Critical Path (CP)
  - Determines execution time
  - Only fixed after execution
- Tools
  - Scalasca [1]
  - DIMEMAS [2]
  - OpenSpeedshop [3]

  → Post-mortem tools, need a trace



- What if we are content with a bit less information?
  - Path of CP needs post-mortem analysis
  - Length of the CP possible to obtain on-the-fly

  → On-the-fly critical path tool (OTF-CPT)

[1] https://scalasca.org/
[2] https://tools.bsc.es/dimemas
[3] https://openspeedshop.org/

Overview

What are critical paths?

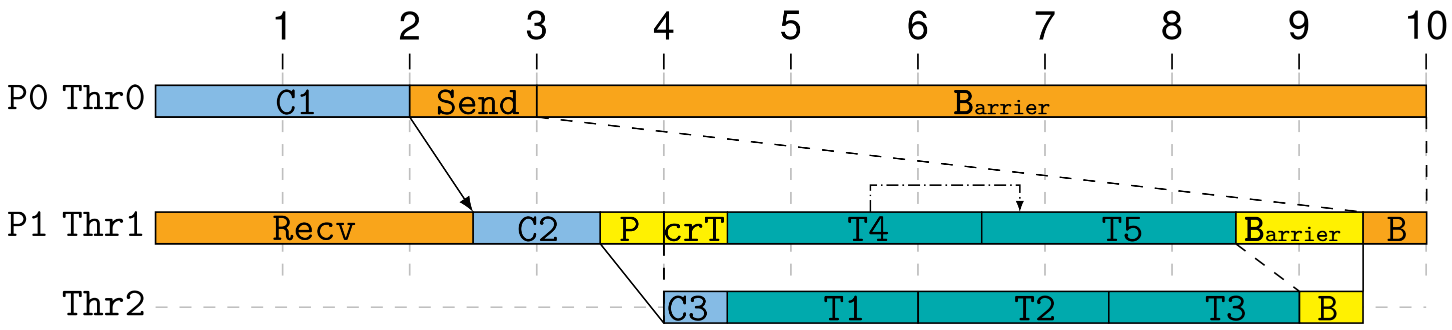
How does the OTF-CPT work?

Use case: POP metrics

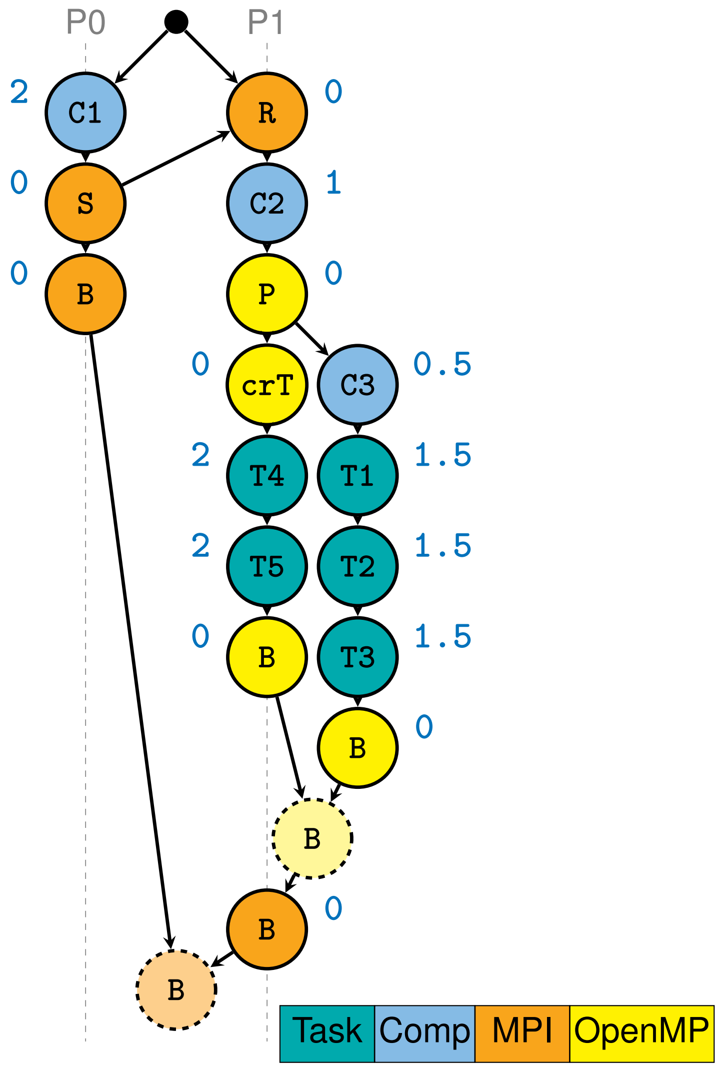
Use case: Tasking recommendations

Future steps and conclusions

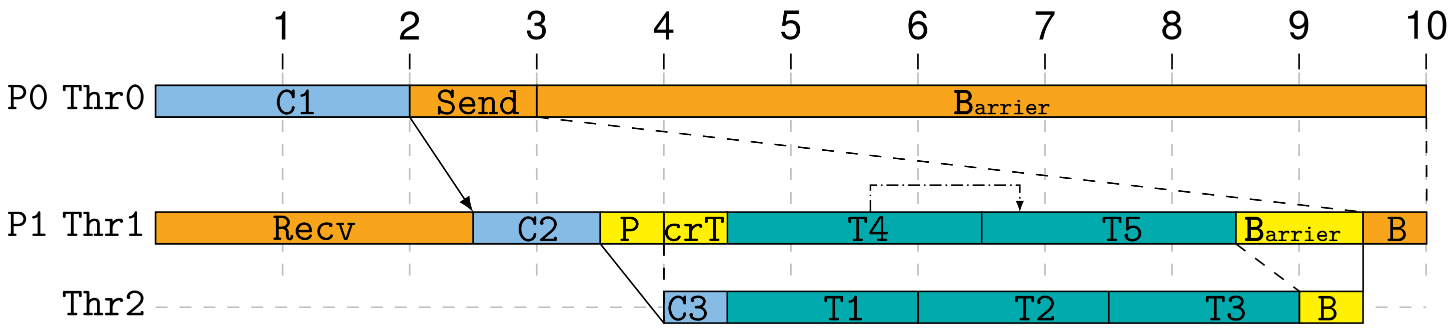
Program Activity Graphs



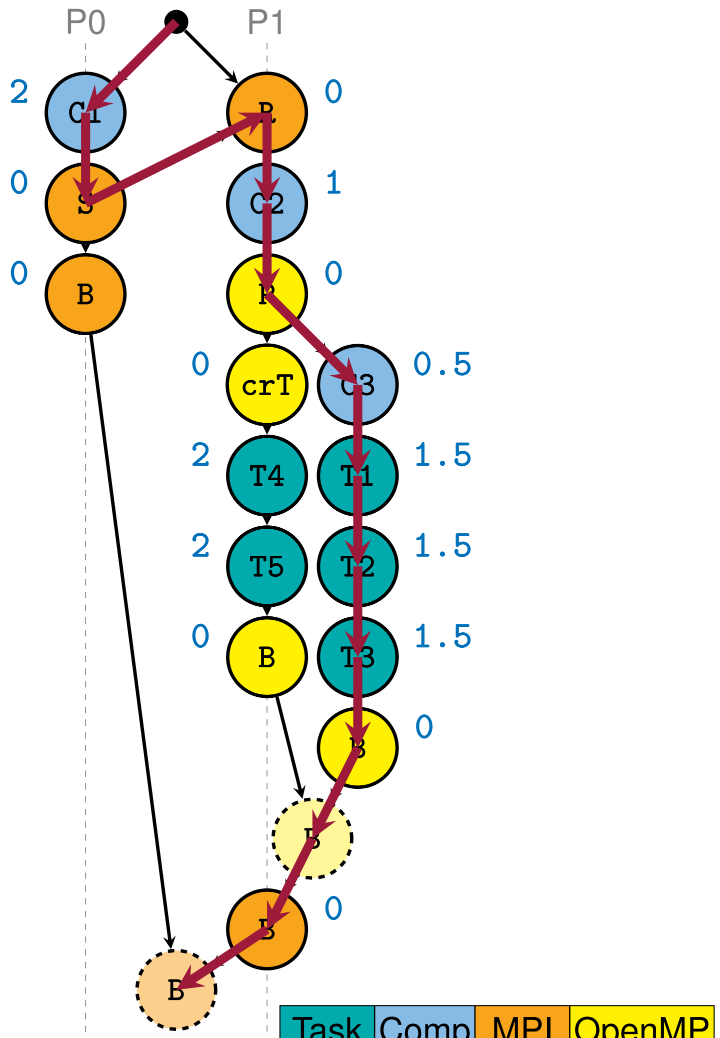
- **Program activity graph**
  - Nodes
    - Non-OpenMP/MPI code regions ('useful execution'), weight=execution time
    - OpenMP or MPI synchronization, weight=0
  - Edges
    - Happens on same thread after another
    - Happens before/after OpenMP/MPI sync point



Different Critical Paths [1]



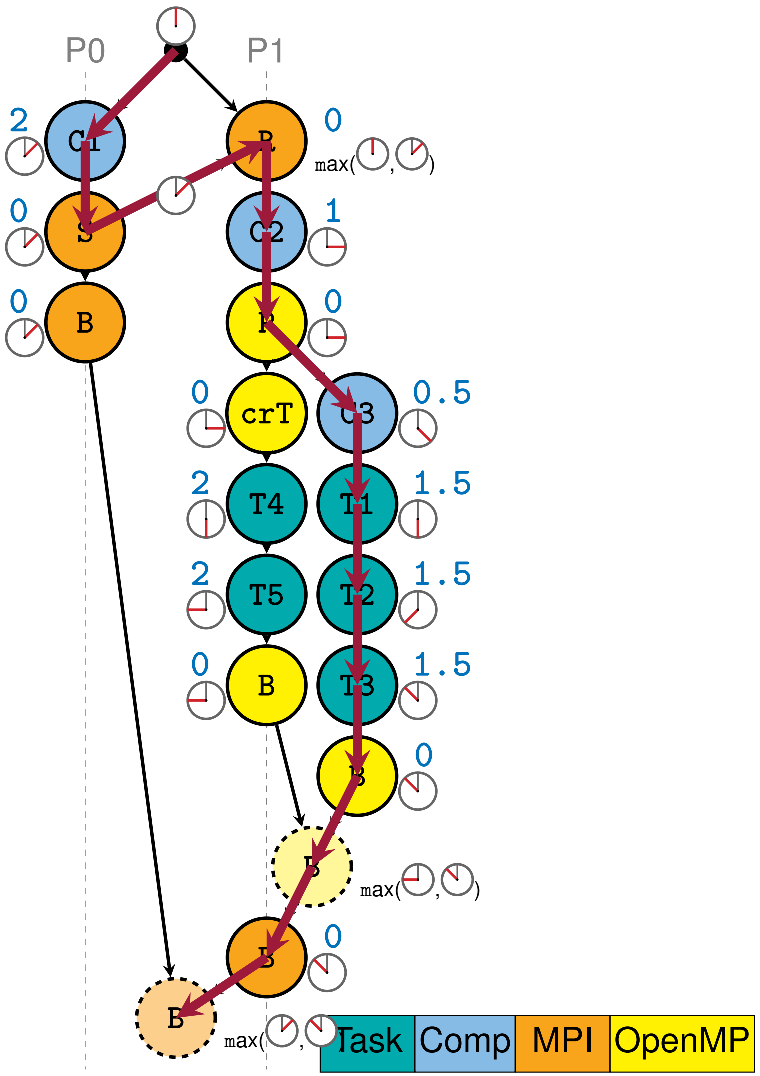
- **Critical Path (CP)**
  - Path with the highest sum of weights
- Considering different aspects results in multiple different CPs
  - **Scope:** Global, process-local, thread-local
  - **Paradigms considered:** Useful execution, Outside MPI, Outside OpenMP



[1] Protze et al. On-the-Fly Calculation of Model Factors for Multi-paradigm Applications. Euro-Par 2022. https://doi.org/10.1007/978-3-031-12597-3_5

OTF-CPT - How It Works

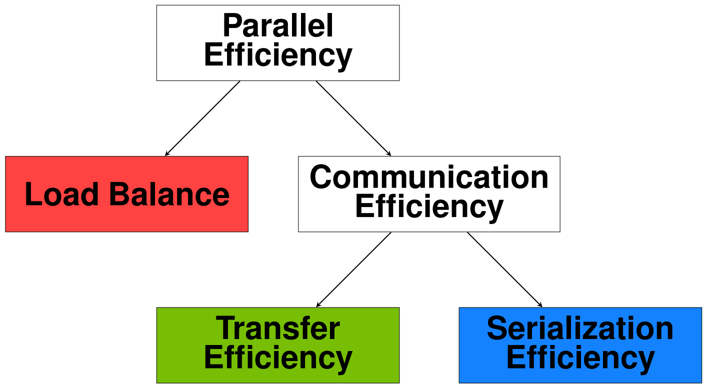
- Threadlocal clocks
  - Count the useful computation time
  - Maximum update at each synchronization point
    - Uses additional processlocal sync clocks
  - For intercepted MPI calls identical calls are made with the threadlocal clock
- Separate counters for different paths
  - Scope: Leave out process/thread synchronization
  - Paradigms: Start/stop measurement upon entering/leaving OpenMP/MPI



POP: Performance Model Factors [1]

- Hierarchically structured performance metrics
  → information on where to look more in-depth

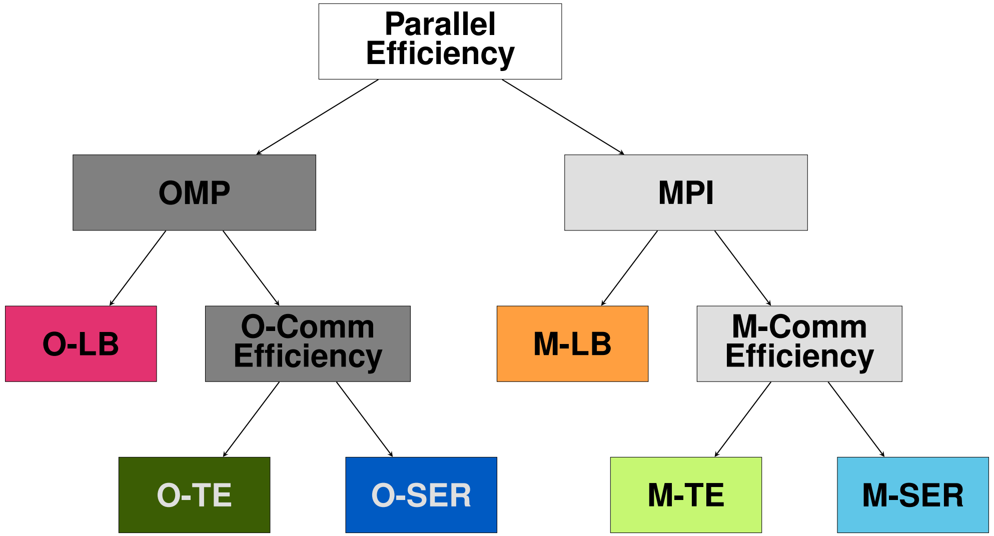
- Parallel Efficiency split into:
  - Load Balance (LB): Global imbalance across exec. units
  - Transfer Efficiency (TE): Efficiency loss due to network and memory transfer times, disappear on 'ideal' network
  - Serialization Efficiency (SER): Efficiency loss due to dependencies, causing alternating processes to wait



[1] https://pop-coe.eu/further-information/learning-material

POP: Performance Model Factors [1]

- Hierarchically structured performance metrics
  → information on where to look more in-depth
- Parallel Efficiency split into:
  - Load Balance (LB): Global imbalance across exec. units
  - Transfer Efficiency (TE): Efficiency loss due to network and memory transfer times, disappear on 'ideal' network
  - Serialization Efficiency (SER): Efficiency loss due to dependencies, causing alternating processes to wait
- Can be further split into separate metrics for OpenMP/MPI [2]
- OTF-CPT obtains parallel efficiency and all its submetrics



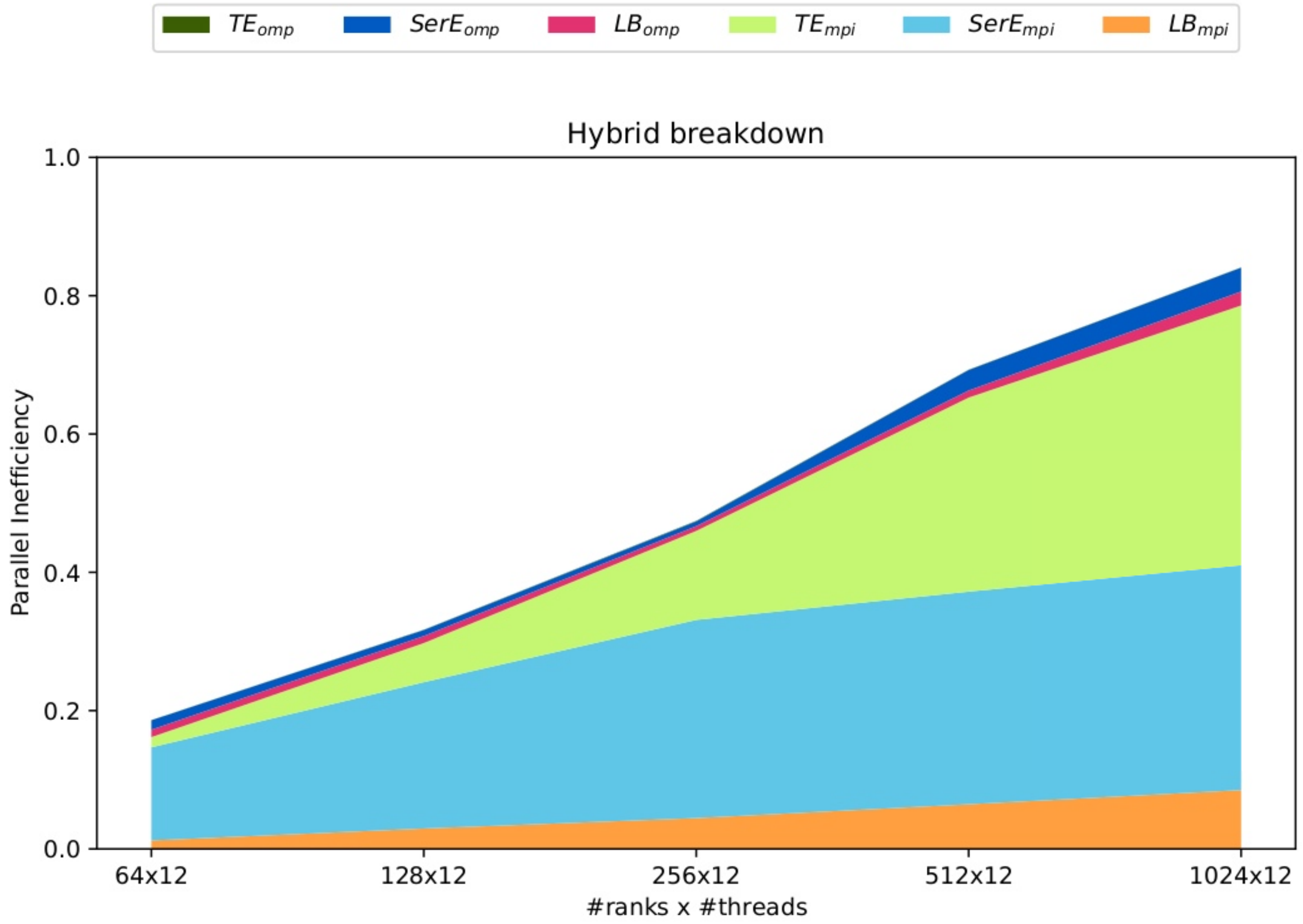
[1] https://pop-coe.eu/further-information/learning-material
[2] Protze et al. On-the-Fly Calculation of Model Factors for Multi-paradigm Applications. Euro-Par 2022. https://doi.org/10.1007/978-3-031-12597-3_5

POP Metrics for Concrete Code

- Measurement:
  - CLAIX-2023: 96 cores per node
  - Strong scaling
  - 12 threads/MPI rank

	64	128	256	512	1024
PE	81.4	68.3	52.6	30.7	15.9
MPI	83.5	69.7	53.5	32.5	17.3
MPI LB	98.7	96.9	95.3	93.0	90.3
MPI CE	84.6	71.9	56.1	34.9	19.1
MPI Ser	85.9	76.7	66.1	56.5	46.4
MPI Tra	98.5	93.8	84.9	61.8	41.2
OMP	97.5	97.9	98.4	94.7	92.1
Omp LB	98.9	98.9	99.3	98.9	97.7
Omp CE	98.6	99.0	99.1	95.7	94.3
Omp Ser	98.6	99.0	99.2	95.8	94.3
Omp Tra	100.0	100.0	99.9	99.9	100.0

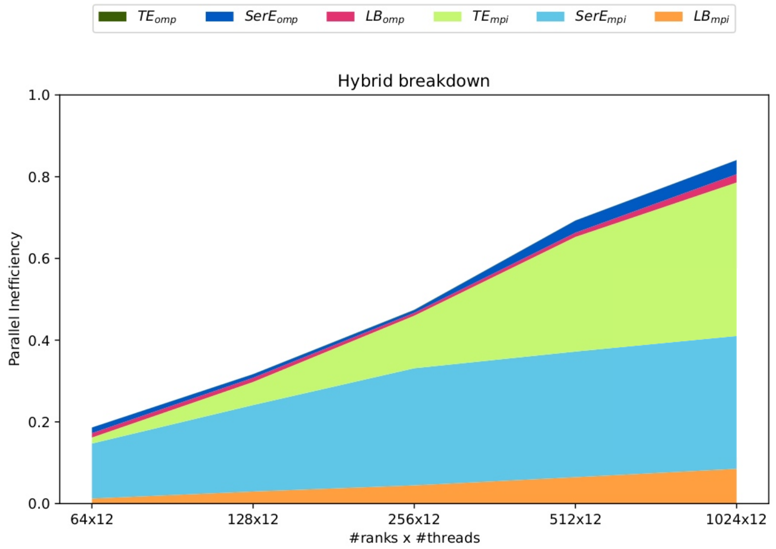
POP Metrics for Concrete Code



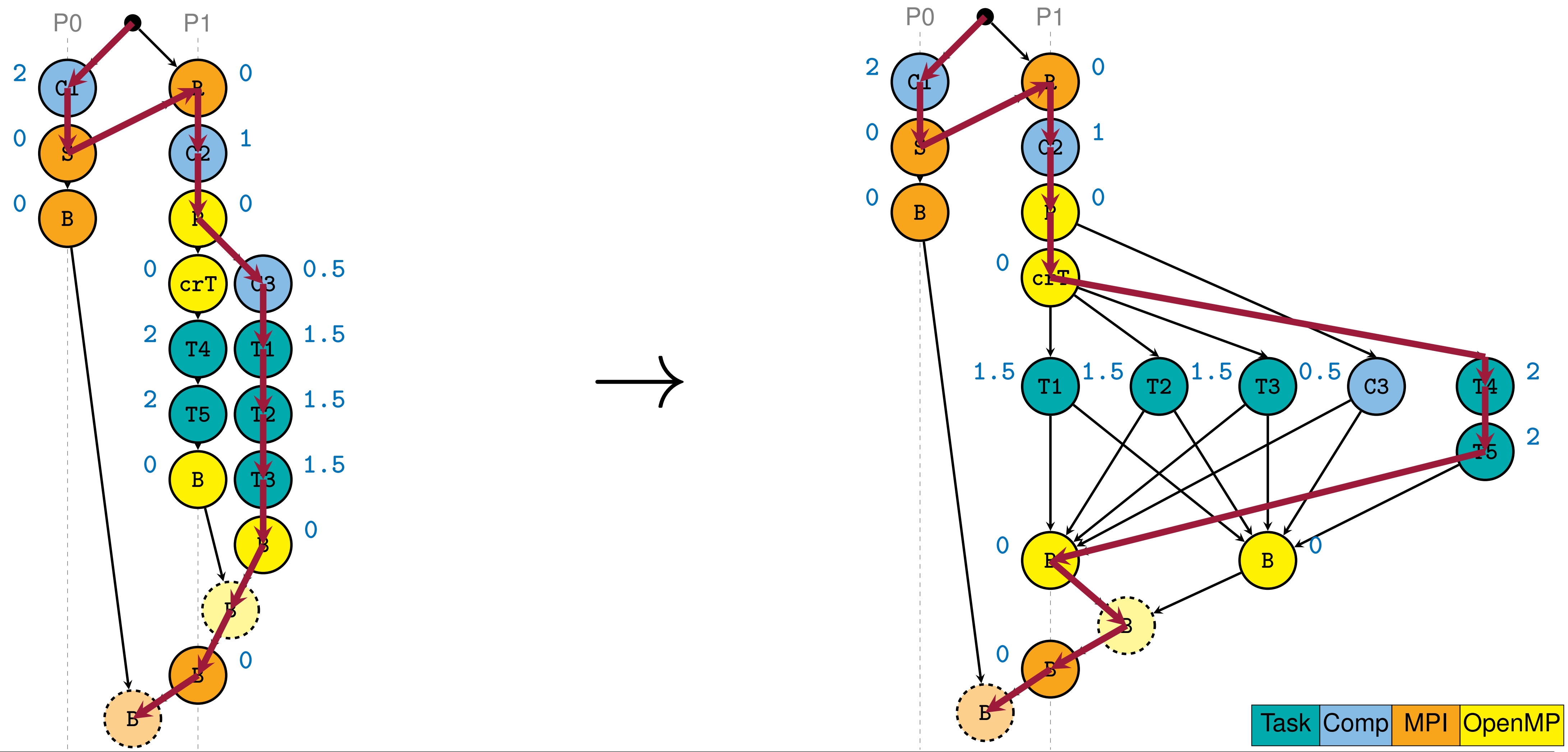
POP Metrics for Concrete Code

- Measurement:
  - CLAIX-2023: 96 cores per node
  - Strong scaling
  - 12 threads/MPI rank
- Results:
  - POP metrics show that problems are the MPI serialization and transfer efficiency
  - Further analysis of the trace identified `MPI_Alltoall` calls as problem
  - Solution: Overlapping of communication and computation

	64	128	256	512	1024
PE	81.4	68.3	52.6	30.7	15.9
MPI	83.5	69.7	53.5	32.5	17.3
MPI LB	98.7	96.9	95.3	93.0	90.3
MPI CE	84.6	71.9	56.1	34.9	19.1
MPI Ser	85.9	76.7	66.1	56.5	46.4
MPI Tra	98.5	93.8	84.9	61.8	41.2
OMP	97.5	97.9	98.4	94.7	92.1
Omp LB	98.9	98.9	99.3	98.9	97.7
Omp CE	98.6	99.0	99.1	95.7	94.3
Omp Ser	98.6	99.0	99.2	95.8	94.3
Omp Tra	100.0	100.0	99.9	99.9	100.0

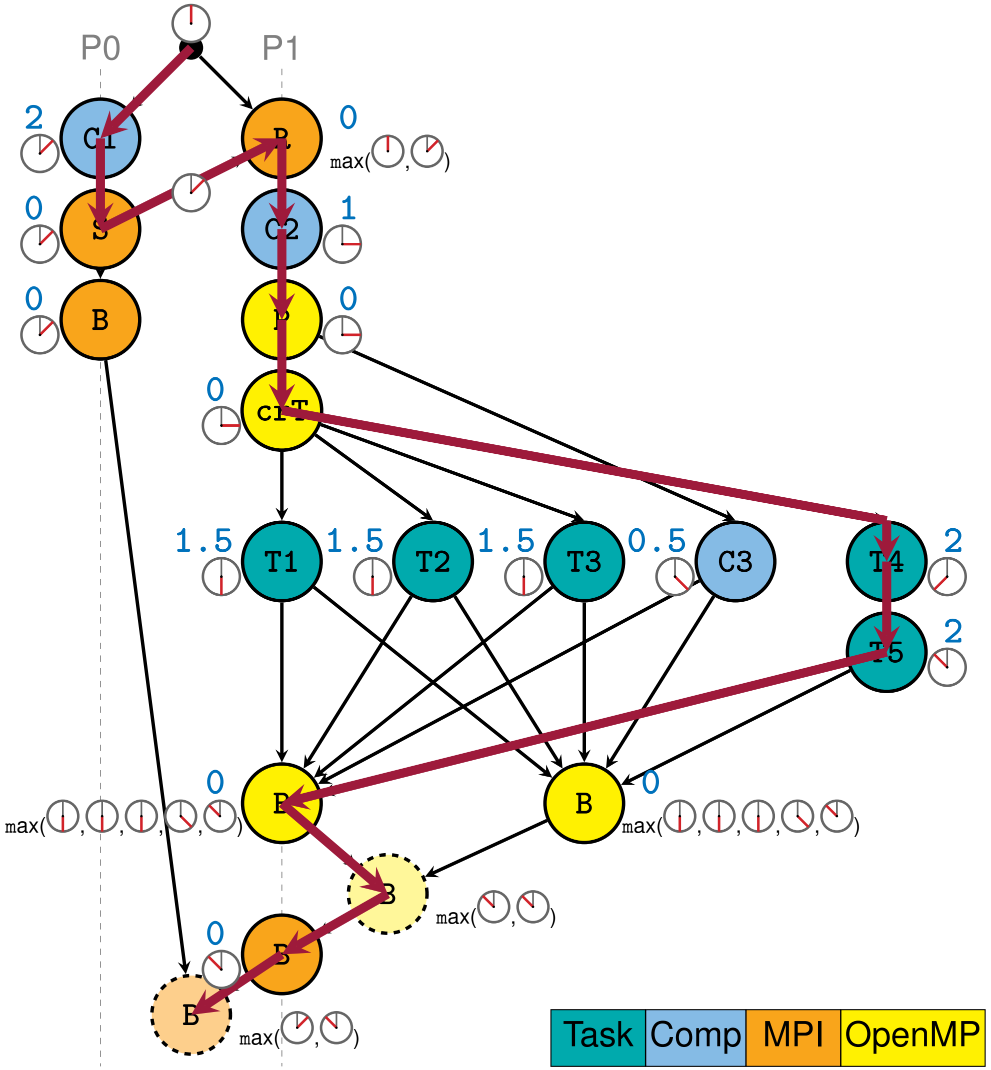


Task-Centric Critical Path



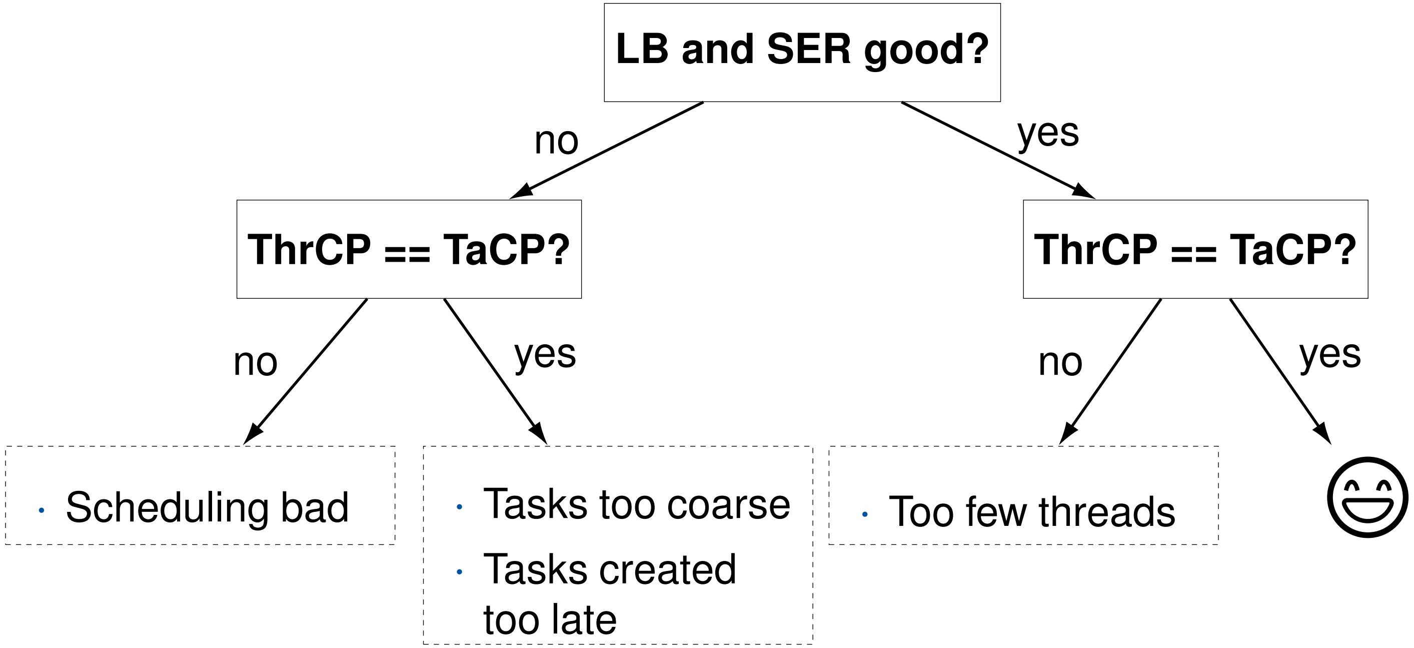
Task-Centric CP – Detection

- How to track task-based critical path?
- 1 clock for each task
  - Gets created when task gets created
  - Only counts time (is active) whenever its corresponding task is scheduled
  - If it is active, it follows the same starting/stopping and synchronization rules as the threadlocal clock



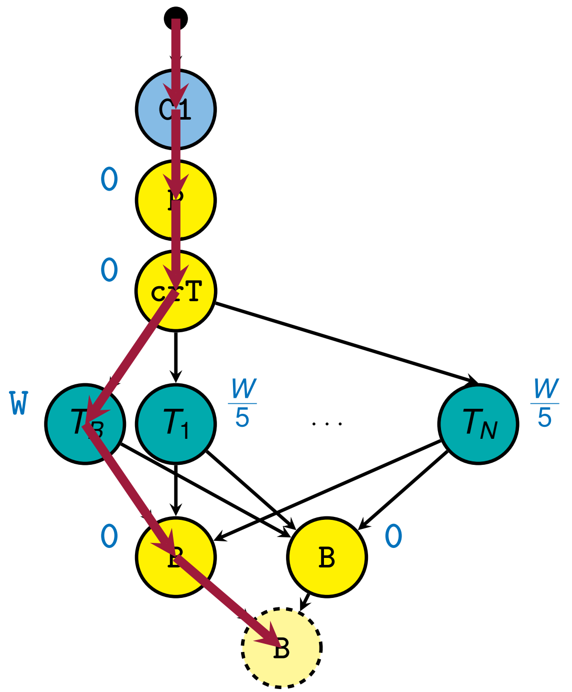
Tasking Suggestions

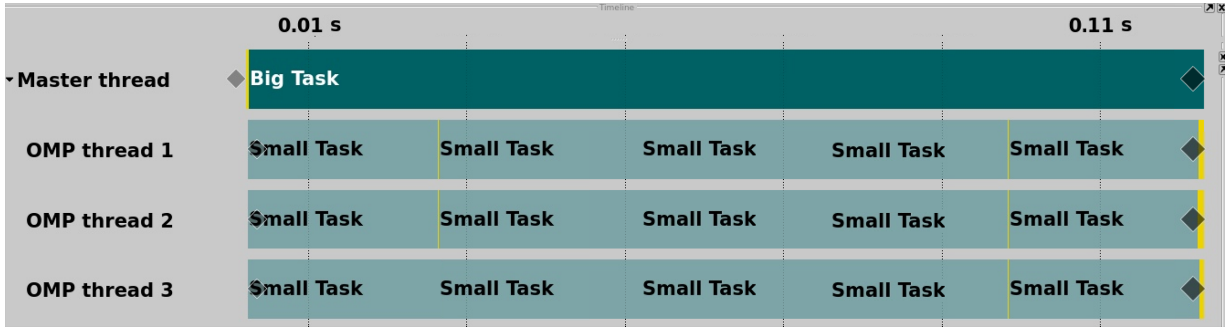
- Task-centric CP (TaCP) gives ‘ideal’ runtime
  (in regards to tasks)
  - perfect scheduling
  - ∞ threads
- Consider difference to thread-centric CP (ThrCP)
- Combine with LB/SER to get idea of inefficiency causes



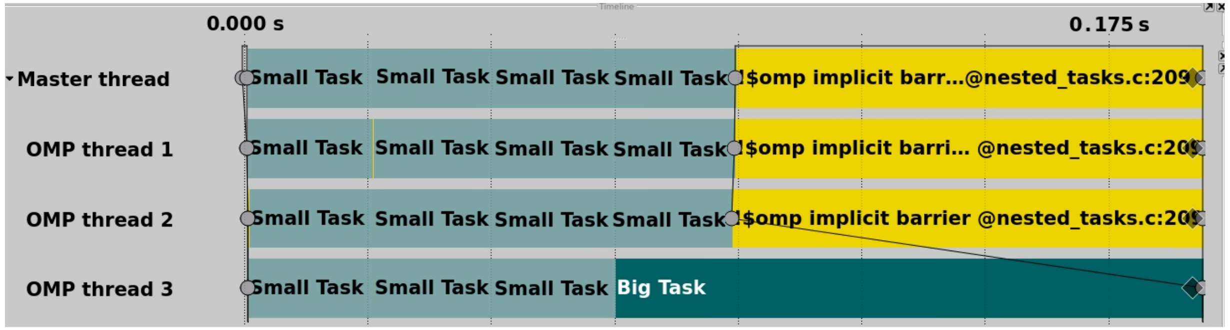
Tasking Test Code

- 1 big task, many small tasks
- Use task priorities to change behavior





Trace for good scheduling (4 threads, 15 small tasks)

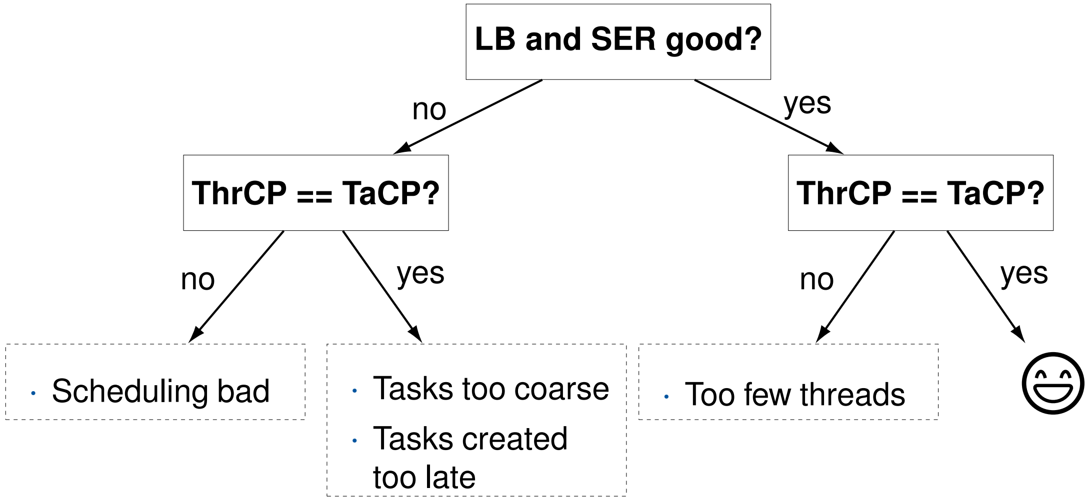


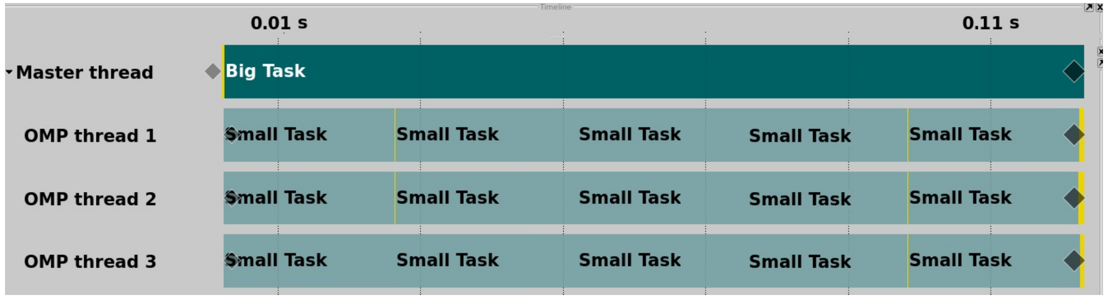
Trace for bad scheduling (4 threads, 15 small tasks)

Test Code - Good Scheduling

- Setup:
  - 1 big task (0.1s), 115 small tasks (0.02s)
  - Enough work for up to 24 threads
  - High priority for big task

Tnum	LB	ThrCP	TaCP	Diff	Diff %
2	100.0	1.466	0.124	1.342	91.6
4	99.9	0.735	0.124	0.611	83.2
6	99.9	0.481	0.121	0.360	74.8
8	99.8	0.368	0.120	0.247	67.3
12	99.8	0.241	0.121	0.120	49.8
16	93.7	0.196	0.123	0.073	37.2
24	99.6	0.124	0.124	1.20E-05	0.0
32	80.7	0.120	0.120	3.00E-06	0.0
48	47.0	0.146	0.146	4.00E-06	0.0
64	35.9	0.153	0.153	4.00E-06	0.0
96	25.2	0.153	0.153	2.00E-06	0.0



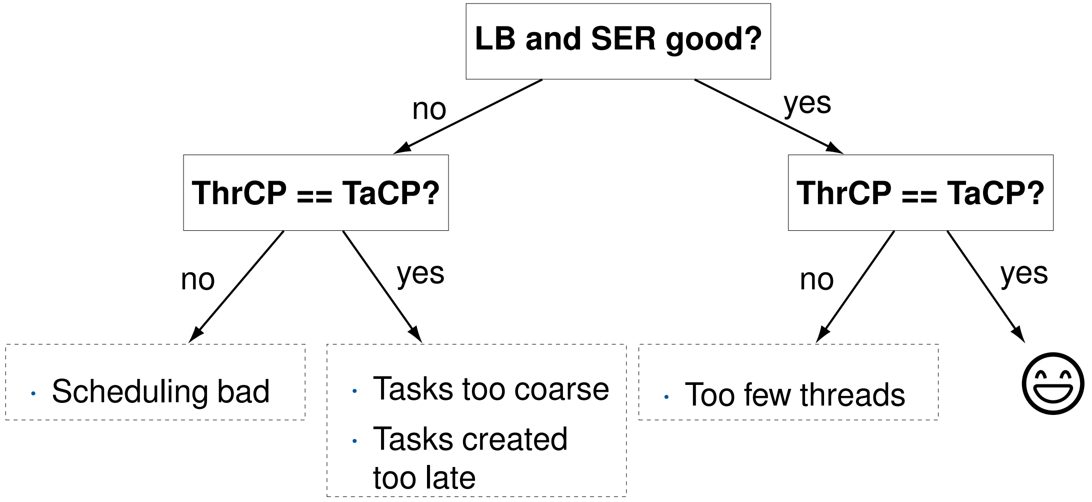


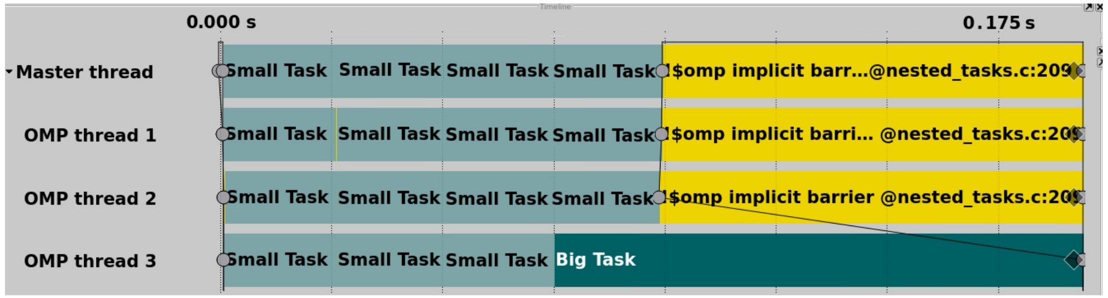
Trace for good scheduling (4 threads, 15 small tasks)

Test Code - Bad Scheduling

- Setup:
  - 1 big task (0.1s), 115 small tasks (0.02s)
  - Enough work for up to 24 threads
  - High priority for small tasks

Tnum	LB	ThrCP	TaCP	Diff	Diff %
2	96.8	1.513	0.119	1.393	92.1
4	90.8	0.794	0.120	0.674	84.8
6	83.4	0.576	0.120	0.456	79.2
8	79.0	0.456	0.120	0.336	73.6
12	71.4	0.343	0.123	0.220	64.2
16	62.6	0.288	0.121	0.168	58.2
24	56.0	0.215	0.119	0.096	44.7
32	46.9	0.193	0.121	0.072	37.3
48	34.9	0.204	0.146	0.058	28.4
64	32.1	0.162	0.131	0.030	18.7
96	21.2	0.180	0.151	0.029	16.0

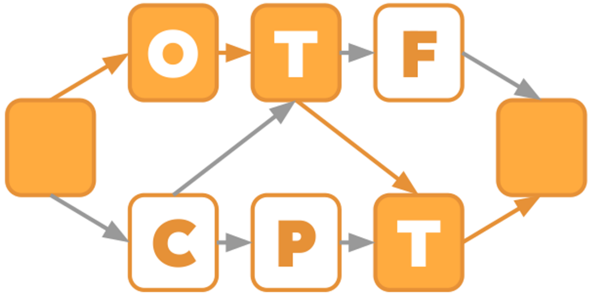




Trace for bad scheduling (4 threads, 15 small tasks)

Conclusions

- Critical Path (CP): longest path in the graph
- On-the-fly measurement of CP using OTF-CPT
  - Threadlocal clocks for each thread
  - Exchange at synchronization points
- Low/No overhead POP metric calculation using OTF-CPT
  - Information on where to look next
- Task-centric analysis
  - Helps to identify problems related to tasking



https://github.com/RWTH-HPC/OTF-CPT