Effect Of Communications On Dynamic Allocation For Distributed Memory Execution Of Dense Tiled Linear Algebra Operations: Cholesky Factorization

Olivier Beaumont
Julien Langou, UC Denver
Willy Quach Northeastern University
Alena Shilova
Mathieu Vérité

OPAL Working Group
February 25th 2021

Presentation Layout

1 - Introduction

Cholesky factorization: $\mathbf{A} = \mathbf{L}\mathbf{L}^T$

- **A**: $n \times n$ symmetric definite positive matrix
  $N \times N$ tiled, each tile being of size b: $N = \frac{n}{b}$
- no tile compression $\Rightarrow$ homogeneous workload per tile
- P homogeneous computation resources (processors) in parallel
- distributed memory $\Rightarrow$ one-to-one communication on a shared medium (bus)
- objective function: minimize total *makespan*

Goals

- illustrate the **effect of data transfers** and **contention** over comm. medium
- understand the **performance** of **StarPU** dynamic runtime **schedulers**
- compare StarPU results with theoretical bounds
- do this on many different platforms (use of StarPU Simgrid)

2.1 - Work Related Bounds

Assuming **no communication cost**, derive a *makespan* lower bound from:

1. perfect balance of **total work** among processors: $\sum_{T \in \mathcal{T}} W_T / P$
2. work of all tasks on the **critical path** in the DAG: $\sum_{T \in \mathcal{CP}} W_T$
3. **ALAP schedule** without constraint on P

Combined *makespan* **lower bound**: $\max((1), (2), (3))$
($\Leftrightarrow$ *speedup* upper bound)

Scheduling *As Late As Possible* [1]

With unlimited number of resources, ALAP schedule of tasks is **optimal**.
Counting the number of **simultaneous tasks** of each type
$\Rightarrow$ provide *makespan* **lower (tighter) bound**.

2.1 - Work Related Bounds

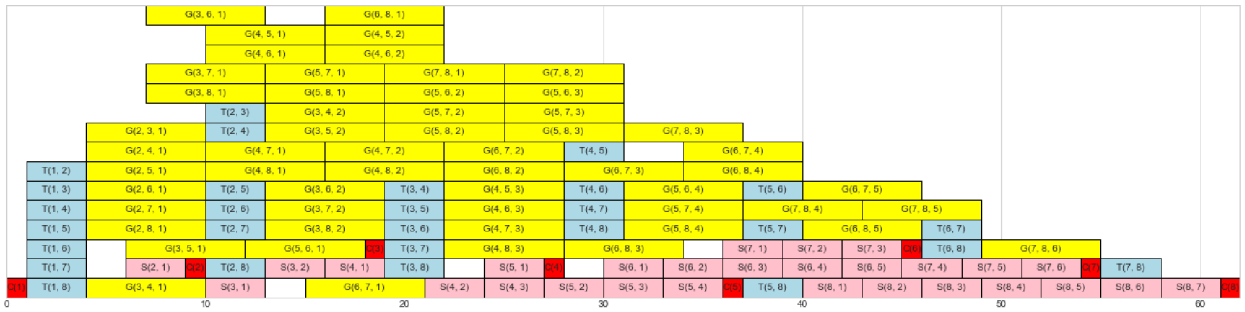


Figure: Gantt chart of Cholesky 8×8 ALAP execution (1,3,3,6 relative performance)

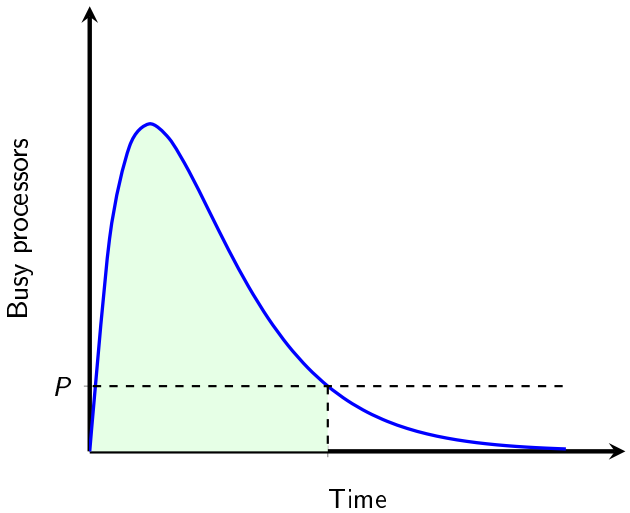


Figure: ALAP execution with unlimited resources

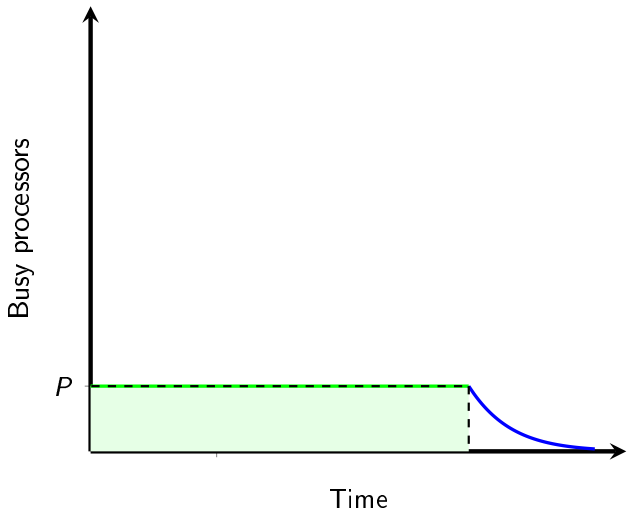


Figure: ALAP execution with P resources

2.1 - Work Related Bounds

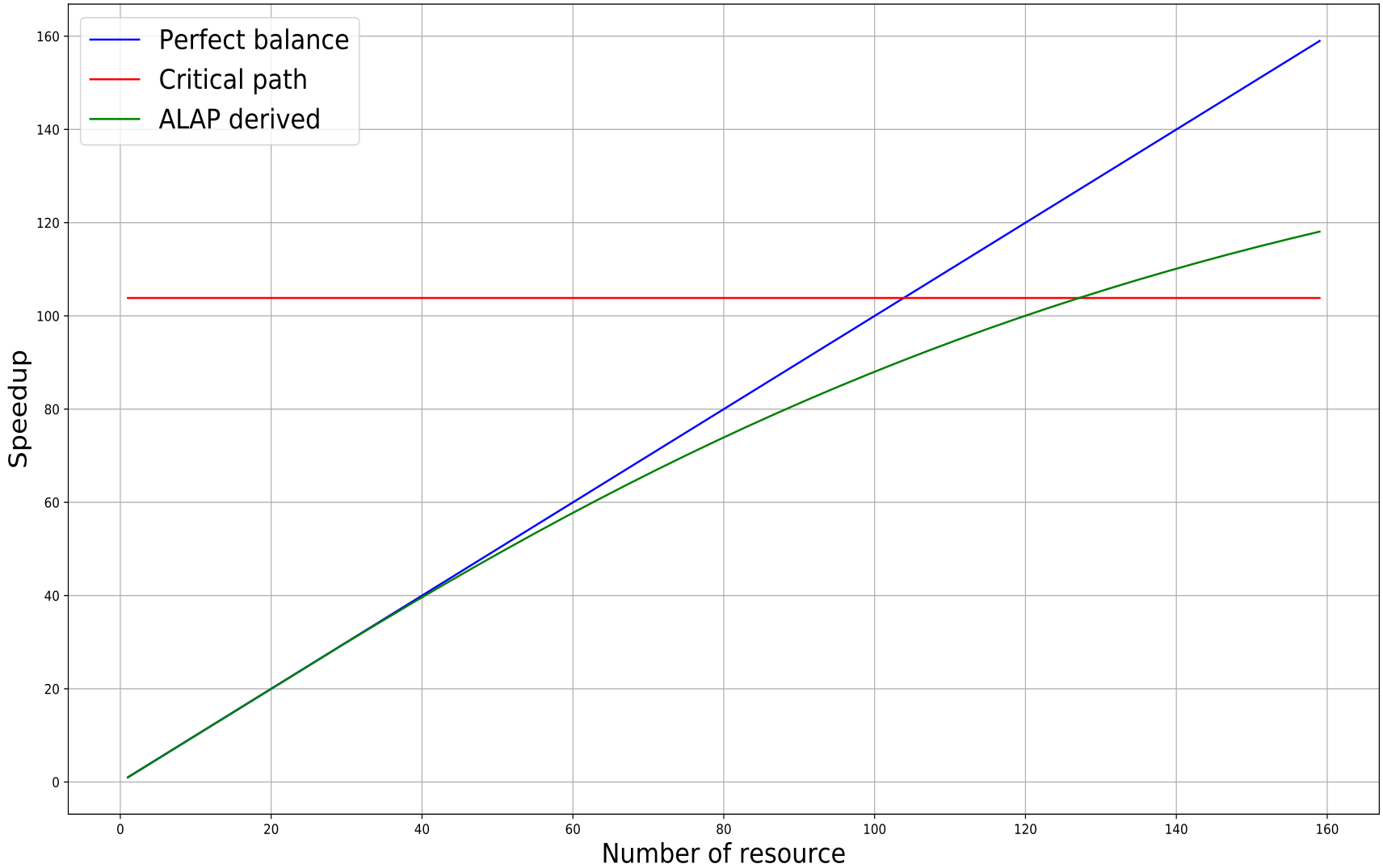


Figure: Theoretical bounds - CPU case ($N = 30$)

2.1 - Work Related Bounds

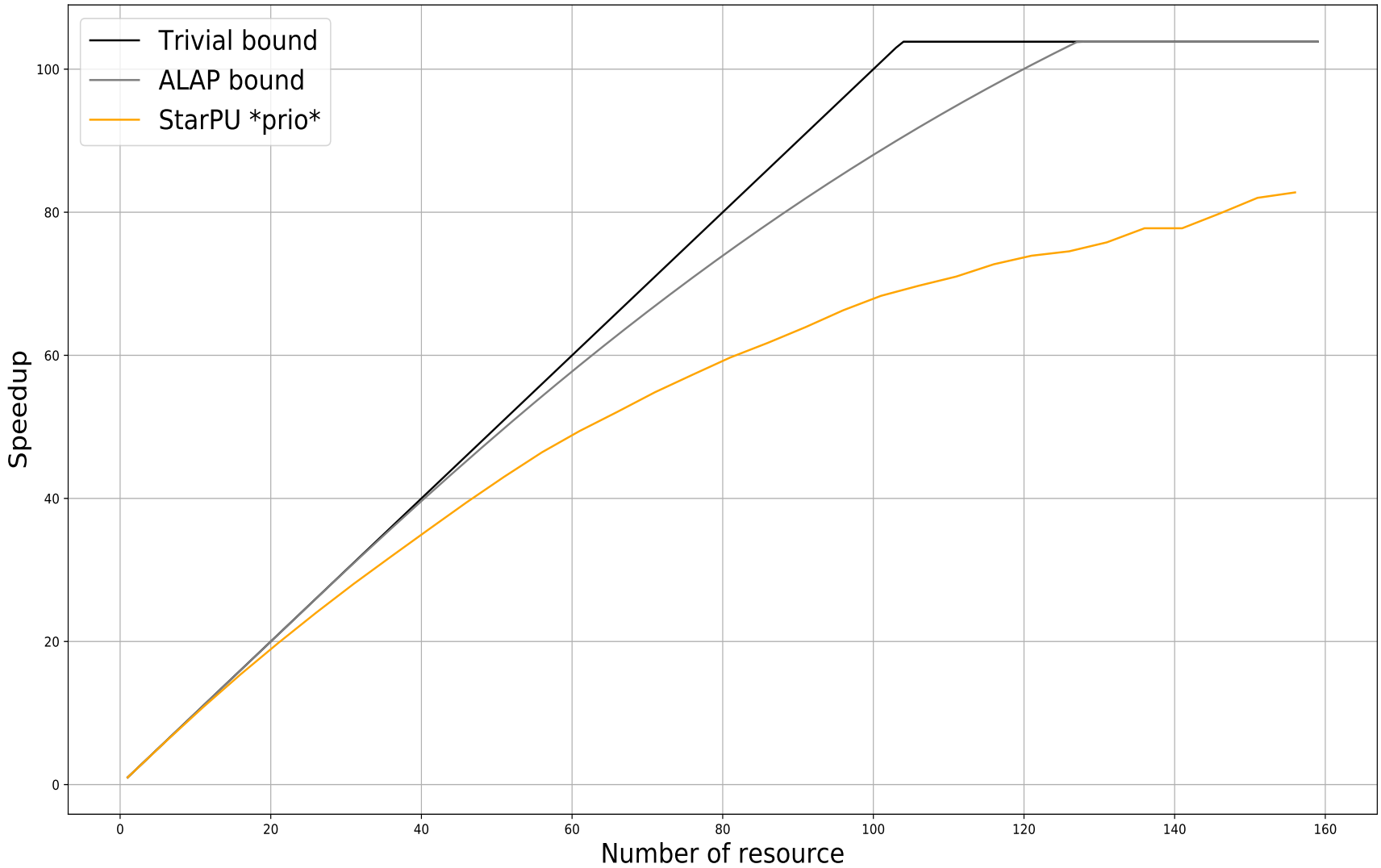


Figure: Results without communications - CPU case ($N = 30$)

2.1 - Work Related Bounds

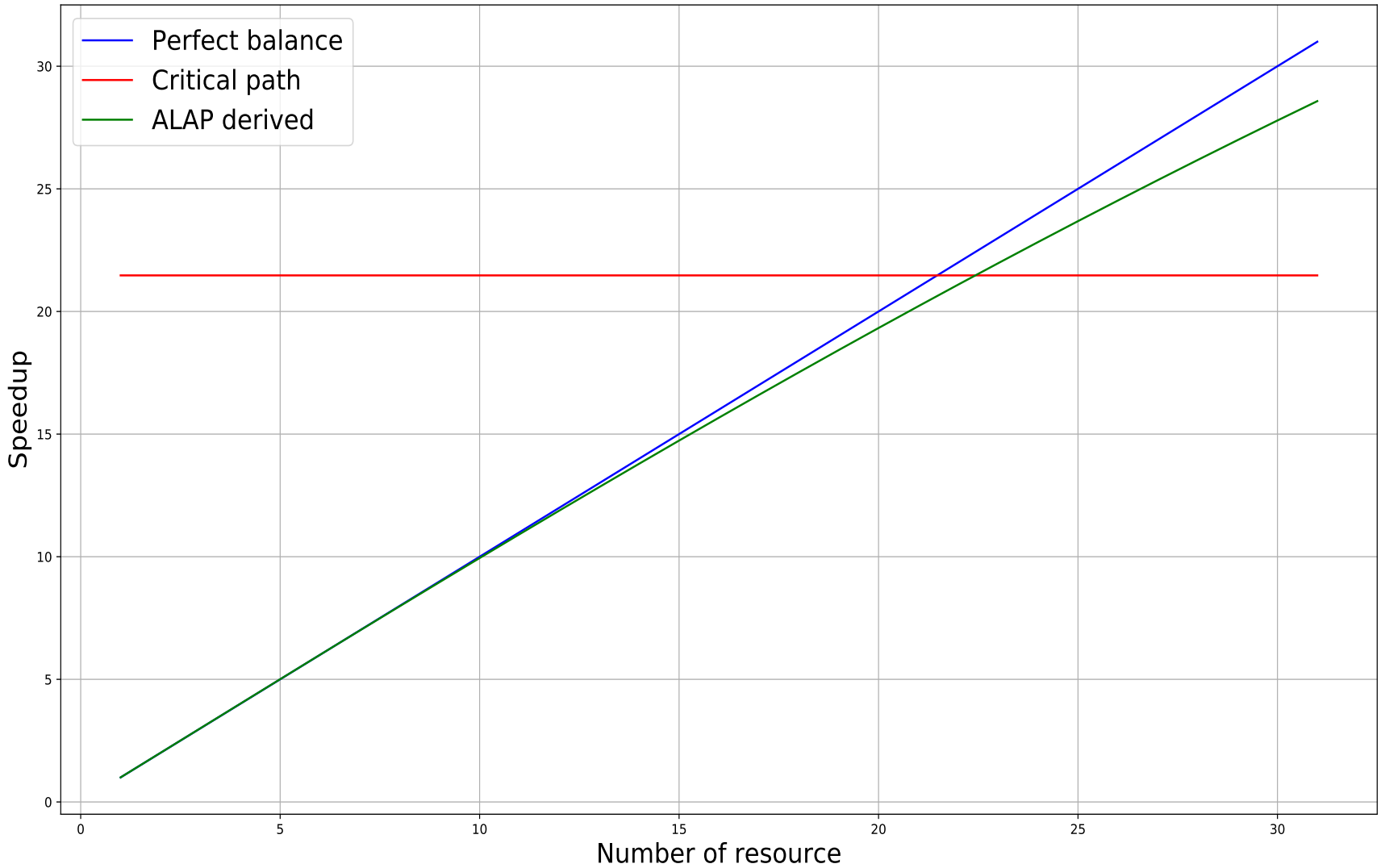


Figure: Theoretical bounds - GPU case ($N = 30$)

2.1 - Work Related Bounds

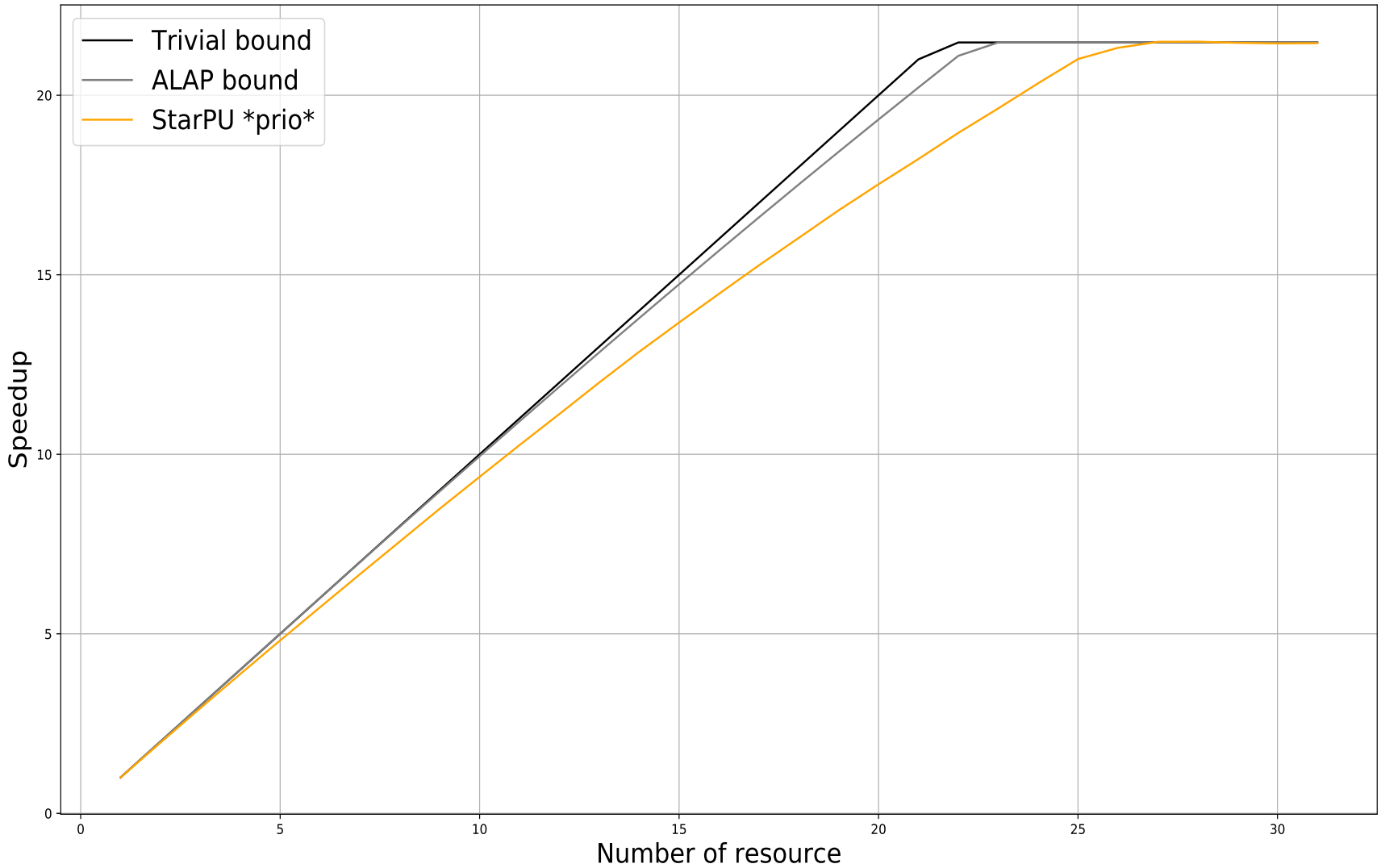


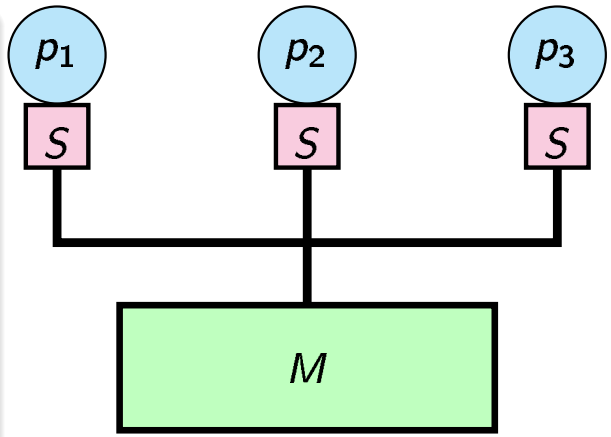
Figure: Results without communications - GPU case ($N = 30$)

2.2 - Communications Related Bounds

Adding **communications costs** $\Rightarrow$ general **problem is hard**
Related bounds exist.

Rastello bound [2]

- **two-level memory**: single slow unlimited memory M, fast individual "cache"of size S
- **store**: $S \rightarrow M$; **load**: $M \rightarrow S$
- **operation** can only be performed if **input data** in S
- **minimize** total number of **loads**
  (data movements in general)



Idea

- A **feasible schedule** = sequence of **store**, **load**, **calculation**
  $\rightarrow$ divide it in **sub-sequences** with exactly T loads (T parameter)
- Set of calculations Q ($\subset \mathrm{DAG}$) of any subsequence is such that: $|Q| = U$ by projections.
- In **[2]**: **automatic search** of all Q and calculation of U from the DAG.

$\Rightarrow$ Bound on the minimal number of data movements required: $T.\lfloor |\mathrm{DAG}|/U_{max} \rfloor$

2.2 - Communications Related Bounds

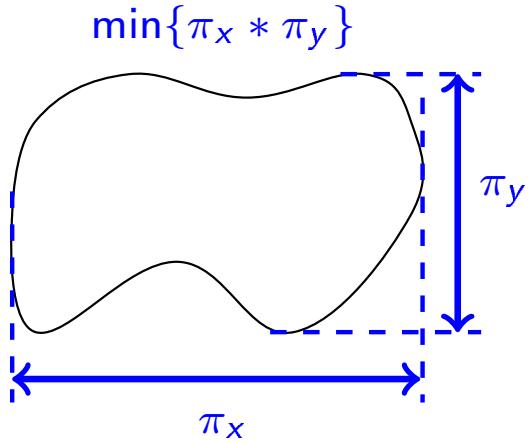
Adding **communications costs** $\Rightarrow$ general **problem is hard**
Related bounds exist.

Demmel bound [3]

- **distributed memory** of size S
- one-to-one communication
- **no contention**
- **minimize** the number of messages **along critical path**
  (simultaneous message = single count)

Idea

- Existing bound for matrix multiplication **[4]**:
  minimal projection of set of computation nodes
- Extend previous bounds **[4]** via a **reduction**:
  can perform a **matrix multiplication** using **cholesky factorization**



2.2 - Communications Related Bounds

Adding **communications costs** $\Rightarrow$ general **problem is hard**
Related bounds exist.

Limitations

Those bounds require that **fast memory** S is **limited**:

- in [**2**] the bound is **parametrized** by S
- in [**3**] the input matrix **A exactly fits in**: $S = \frac{n^2}{P}$

Considering typical order of magnitudes for CPU or GPU memory size:

- the bounds are only relevant for **very large problems**
  example: double precision using $S = 12\text{GB}$ and $P = 30 \Rightarrow n \geqslant 2.10^5$
- for smaller ones $\Rightarrow$ best solution: use **only one processor**

2.2 - Communications Related Bounds

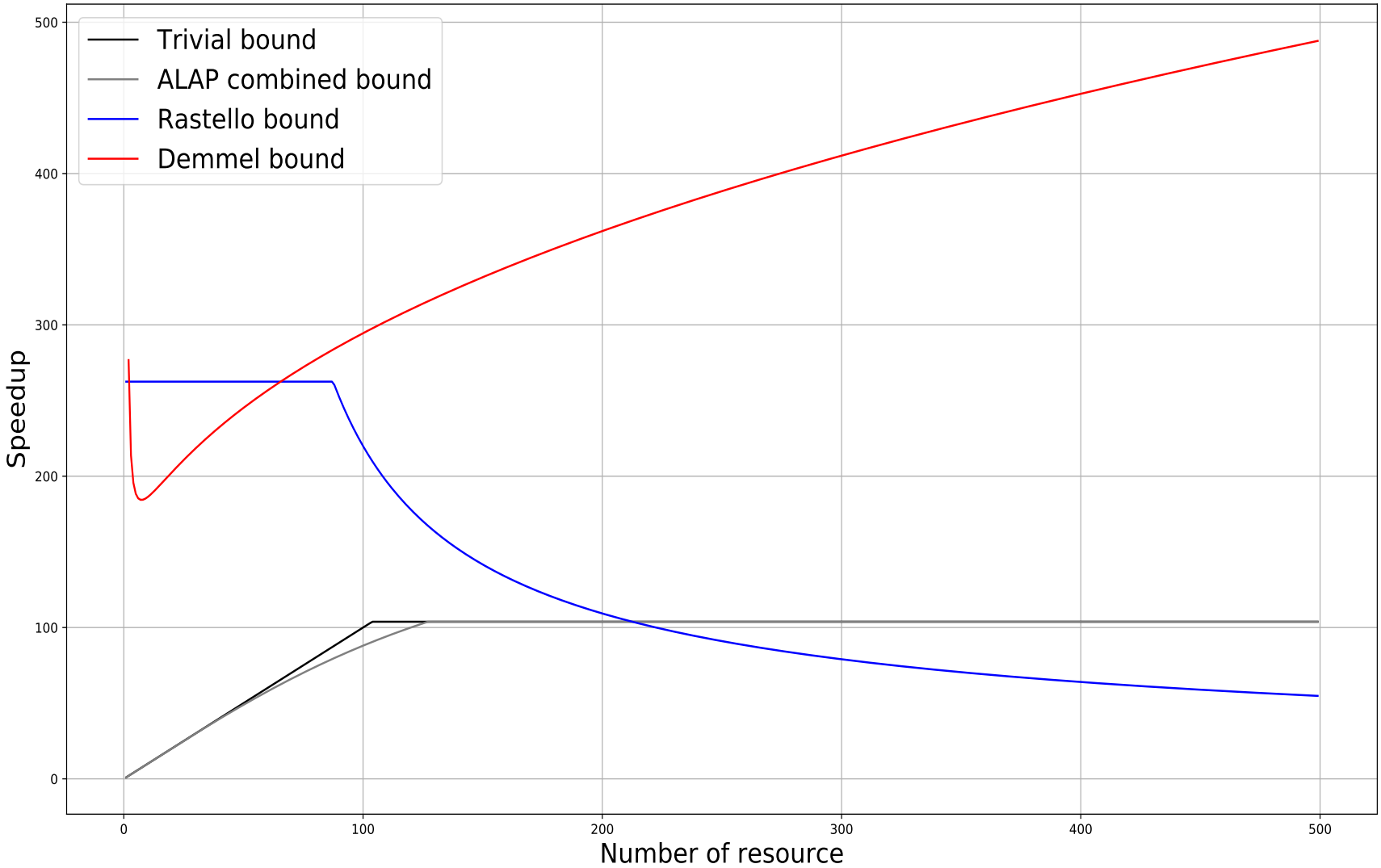


Figure: Communication bounds - CPU case ($N = 30$, $BW = 10GB/s$, $S = \frac{N^2}{P}$)

2.2 - Communications Related Bounds

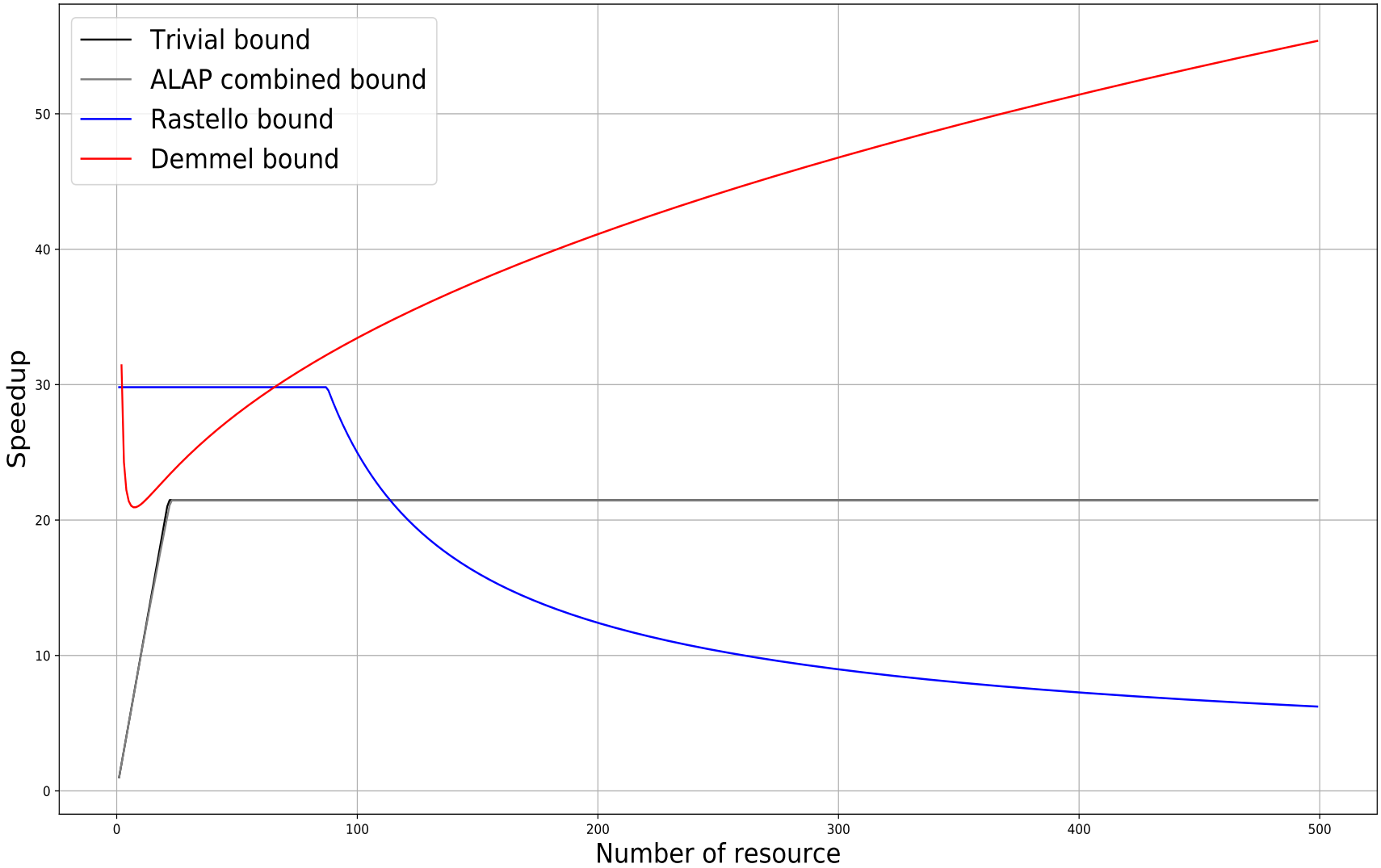


Figure: Communication bounds - GPU case ($N = 30$, $BW = 10GB/s$, $S = \frac{N^2}{P}$)

2.3 - *2D block cyclic* allocation

State-of-the-art method for dense tiled matrix:

- **good load balancing** for a given P ("fits" in N)
- expected to **limit data transfer** (parallelism being fixed along the execution): reduced number of processors in each **broadcast** (same row/column)

From a simple 2DBC allocation pattern ($tile \leftrightarrow P$):

1. maximum load among P processors
2. number of induced communications between processors

$\Rightarrow$ a "good" *makespan* lower estimation: max((1), (2))

Limitations:

- **not a bound**: only consider 2D allocations
- (1) and (2) **not necessarily feasible**

2.3 - *2D block cyclic* allocation

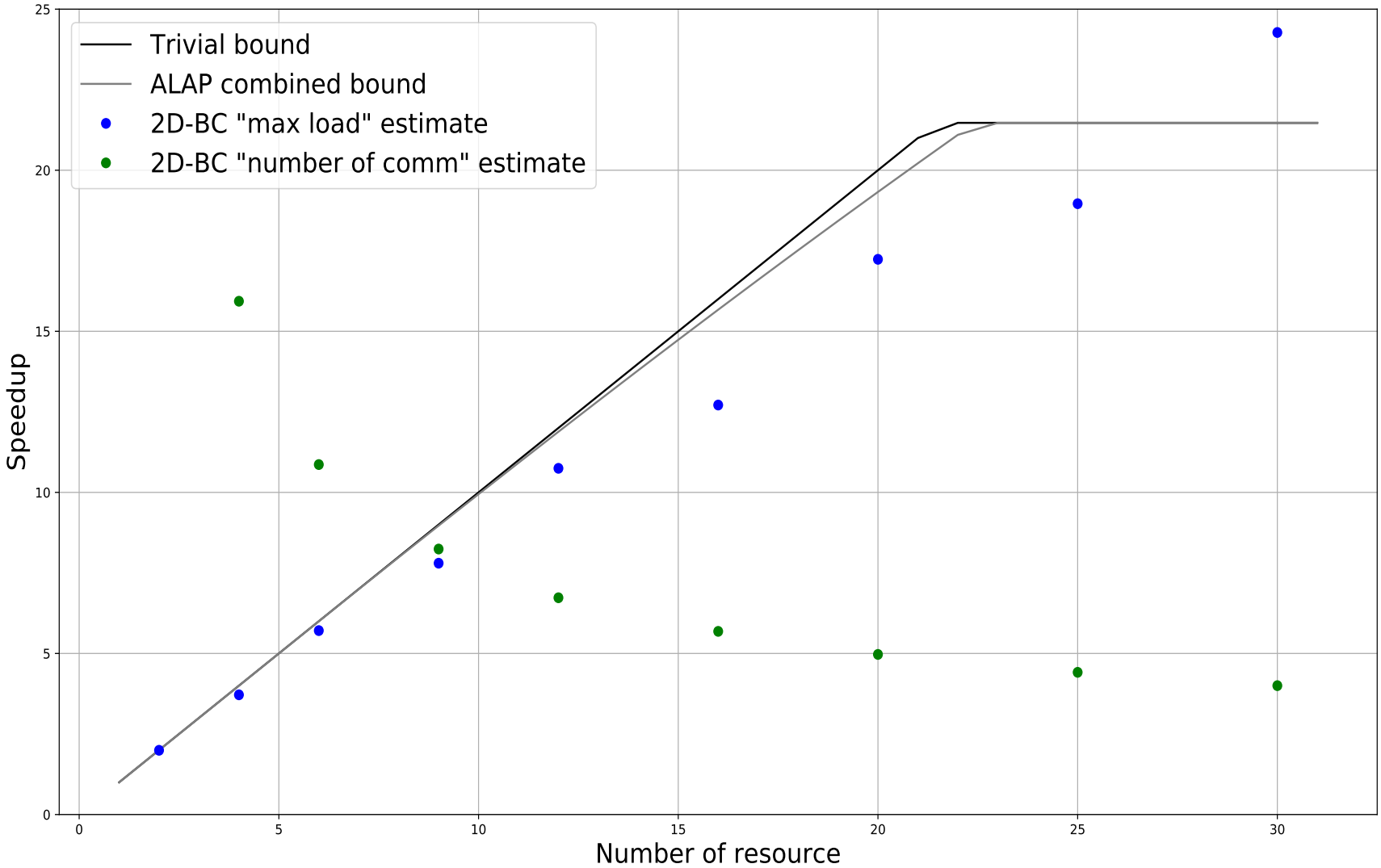


Figure: 2D block cyclic makespan estimate - GPU case ($N = 30$, $BW = 10GB/s$)

2.3 - *2D block cyclic* allocation

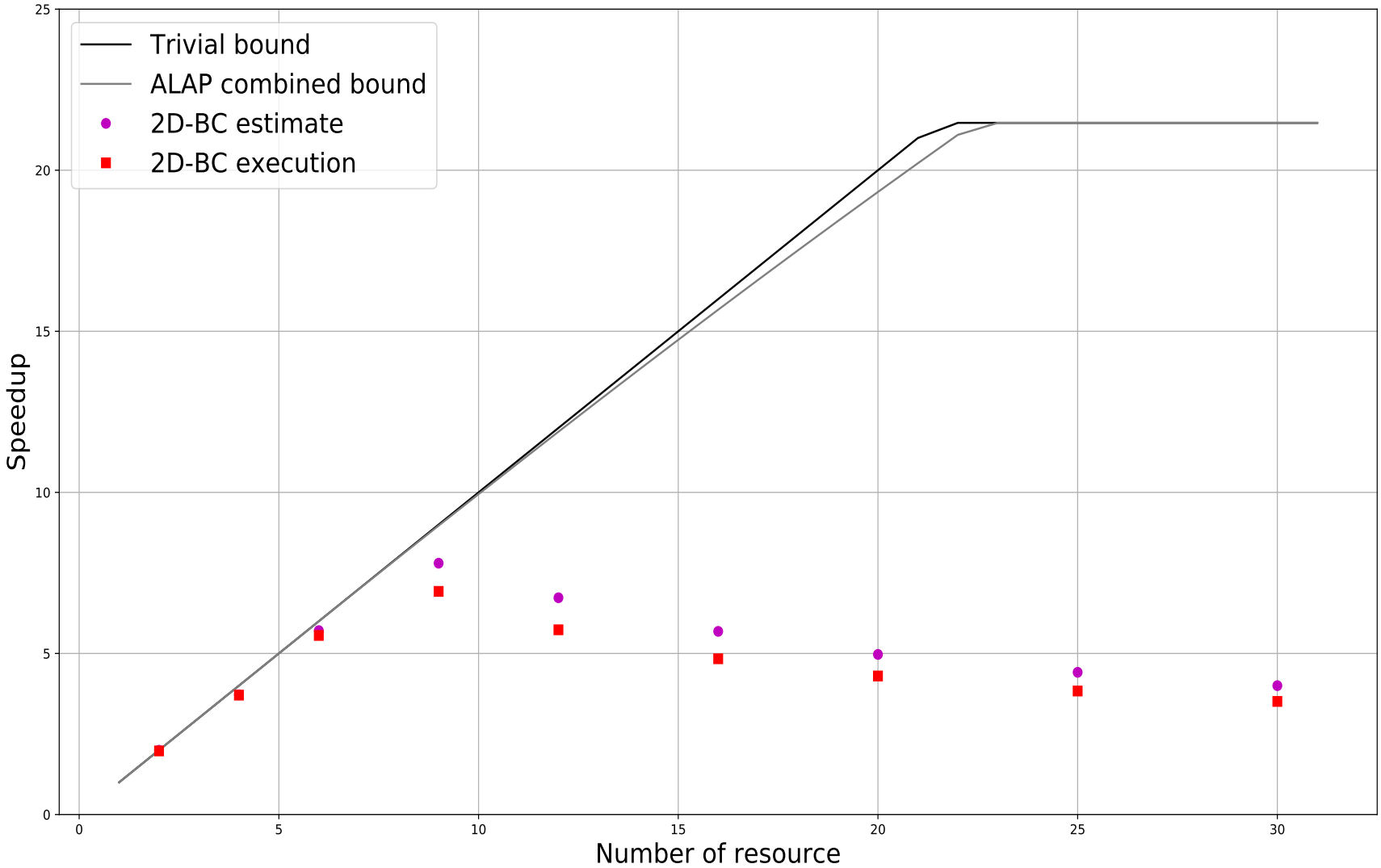


Figure: 2D block cyclic execution - GPU case ($N = 30$, $BW = 10GB/s$)

2.3 - *2D block cyclic* allocation

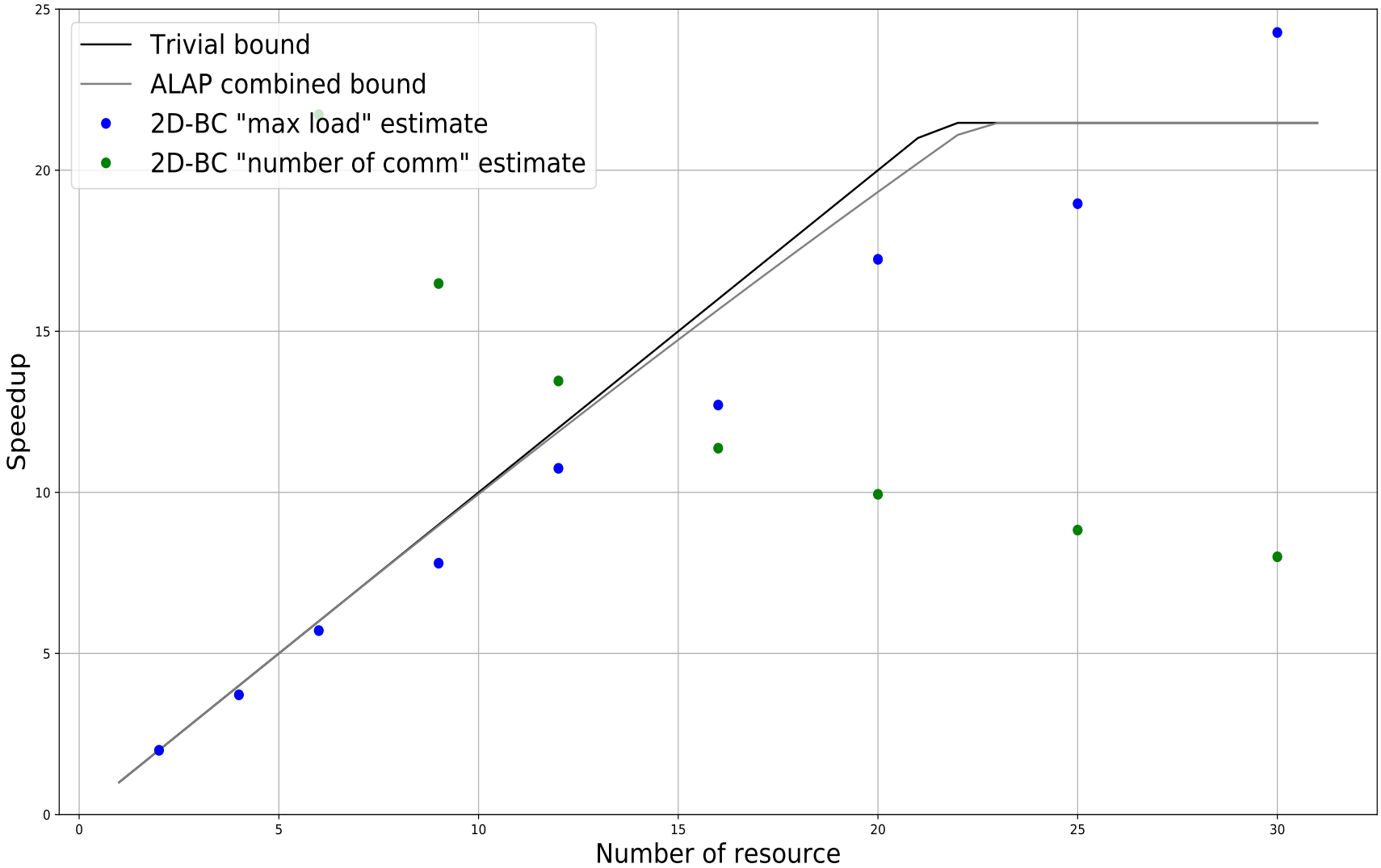


Figure: 2D block cyclic makespan estimate - GPU case ($N = 30$, $BW = 20GB/s$)

2.3 - *2D block cyclic* allocation

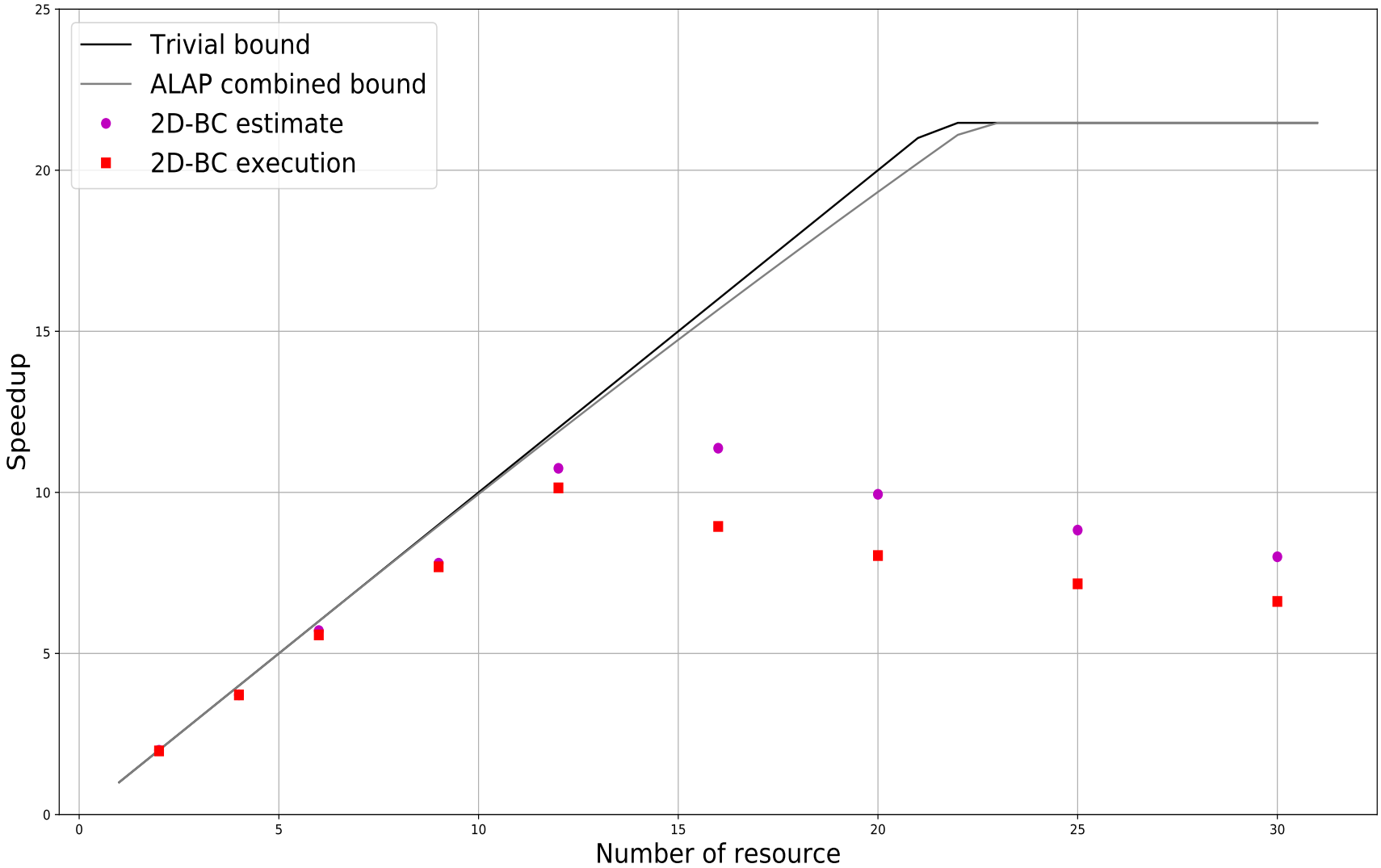


Figure: 2D block cyclic execution - GPU case ($N = 30$, $BW = 20GB/s$)

2.3 - *2D block cyclic* allocation

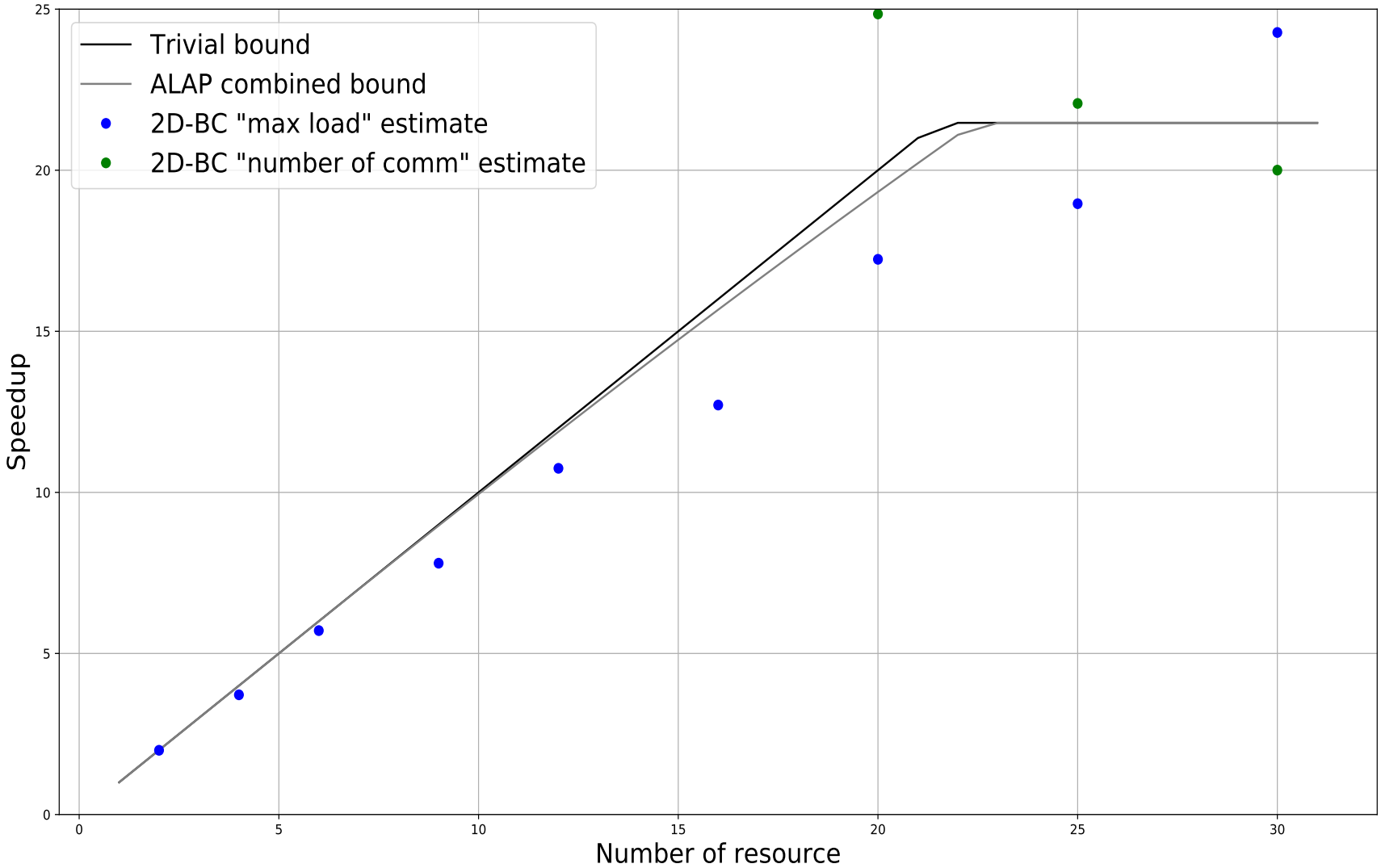


Figure: 2D block cyclic makespan estimate - GPU case ($N = 30$, $BW = 50GB/s$)

2.3 - *2D block cyclic* allocation

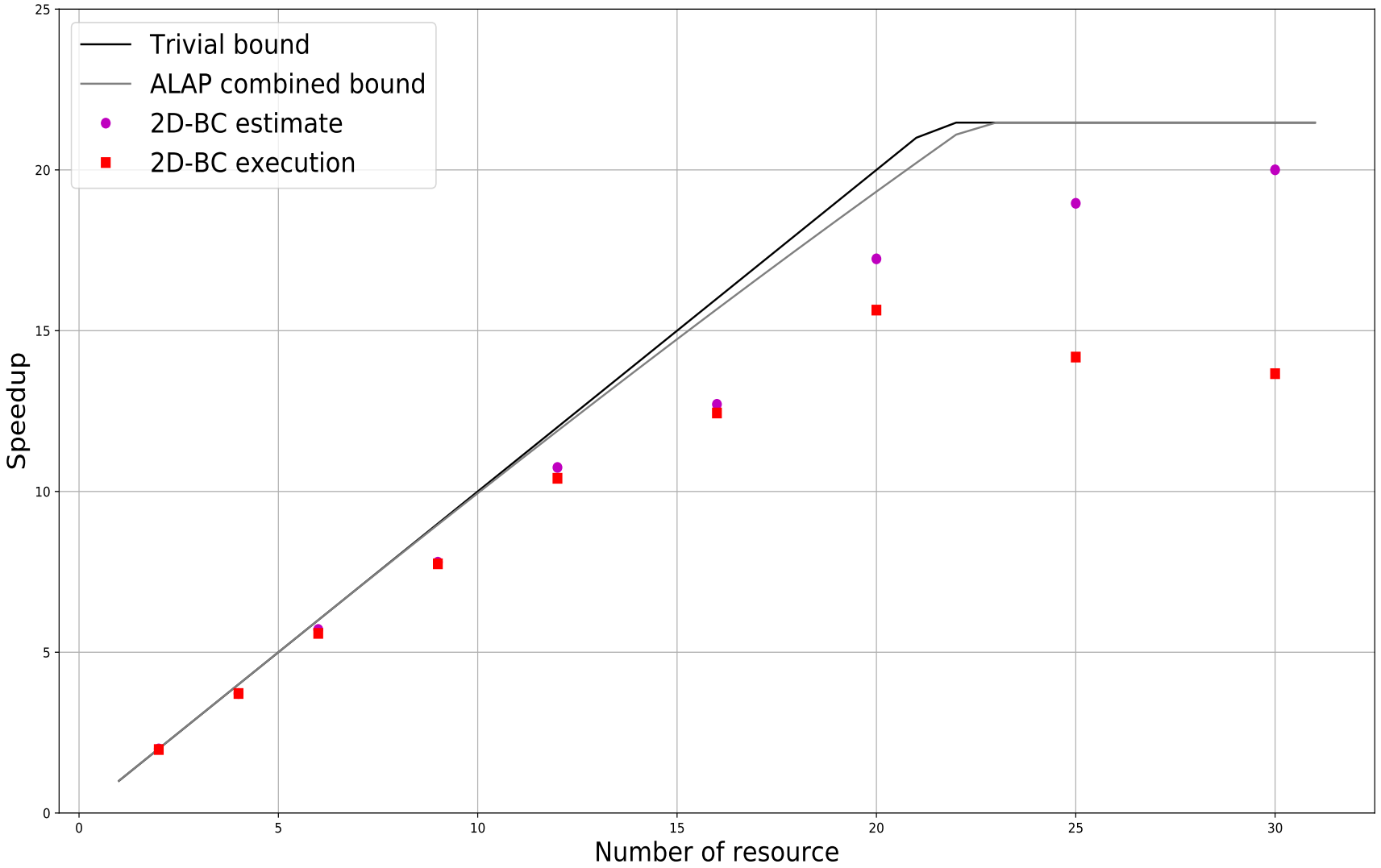


Figure: 2D block cyclic execution - GPU case ($N = 30$, $BW = 50GB/s$)

3.1 - Experimental Environment

Simulated execution using: **Chameleon** & **StarPU** & **SimGrid**

Priorities (Chameleon)

Limited number of priority levels: actually 11 available $\{-5, ..., 5\}$.
$\Rightarrow$ Gather tasks in equal size intervals of local critical path.

For each task T:
$\ell(T)$ = longest path from T to the last `POTRF` in the DAG ($0 \leqslant \ell(T) \leqslant \mathcal{CP}$).

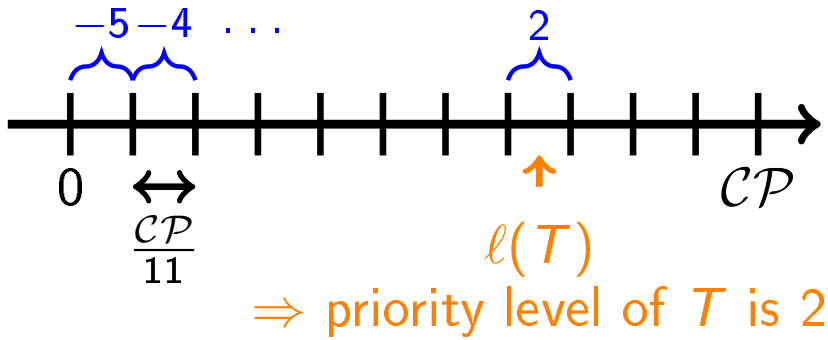


Figure: Priority level calculation

3.1 - Experimental Environment

Simulated execution using: **Chameleon** & **StarPU** & **SimGrid**

Performance model and execution (StarPU)

2 types of **identical computation resource** : 4-core CPU and GPU.
(`STARPU_PERF_MODEL_HOMOGENEOUS_CPU/CUDA = 1`)

Kernel	GEMM	POTRF	SYRK	TRSM
Execution time (ms)	20/2	3/12	10/1	10/3

Table: Performance values for CPU and GPU

No cost for "handling" tasks: remove (almost) all `STARPU_SIMGRID_..._COST`.
Remove tasks pipelining: `STARPU_CUDA_PIPELINE = 1`.

3.1 - Experimental Environment

Simulated execution using: **Chameleon** & **StarPU** & **SimGrid**

Communication model (SimGrid)

Topology: (<link ... >)

- **single shared bus** (sharing_policy="SHARED")
- limited bandwidth (bandwidth=...GBps)
- no latency (latency=0us)

3.2 - Dynamic Allocation

StarPU scheduling policies

Single queue & late allocation decision: `eager`, `prio`

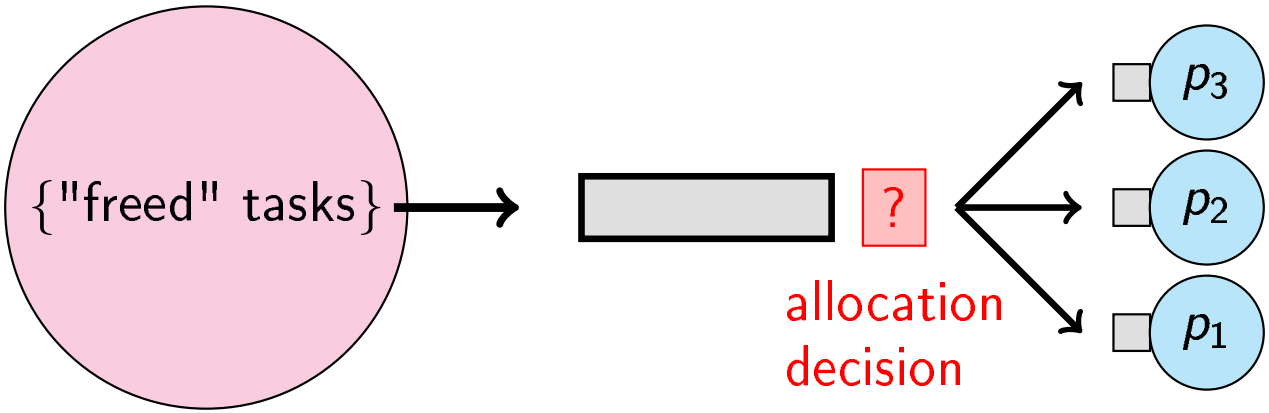


Figure: Single queue schedulers principle ($P = 3$)

3.2 - Dynamic Allocation

StarPU scheduling policies

One queue per processor & immediate allocation decision:
Heterogeneous Earliest Finish Time strategies $\rightarrow$ `dmdaX` schedulers

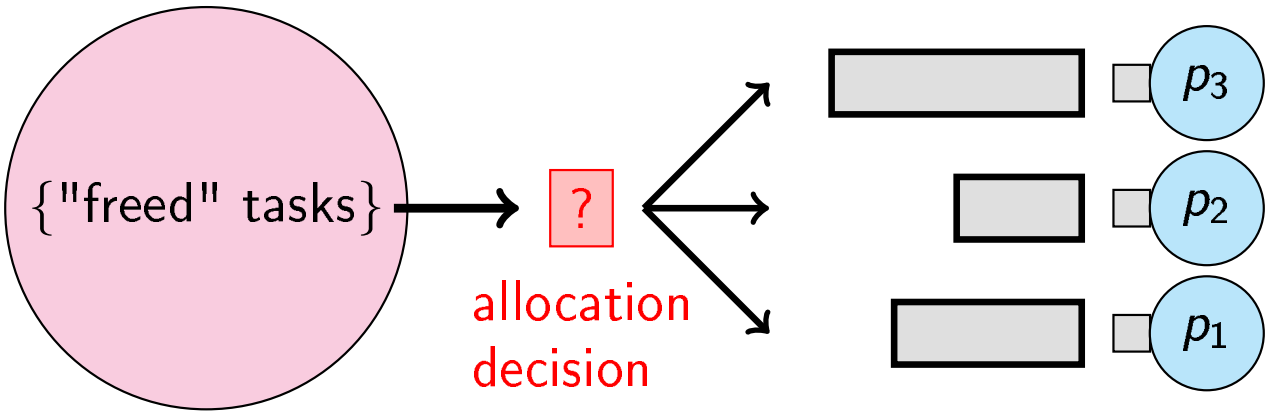


Figure: Individual queues schedulers principle ($P = 3$)

3.2 - Dynamic Allocation

Observations

- Tight communication constraints: `eager` and `prio` shows poor performance.
- No communication constraints. `prio` $\approx$ `dmdaX` and almost optimal.

3.2 - Dynamic Allocation

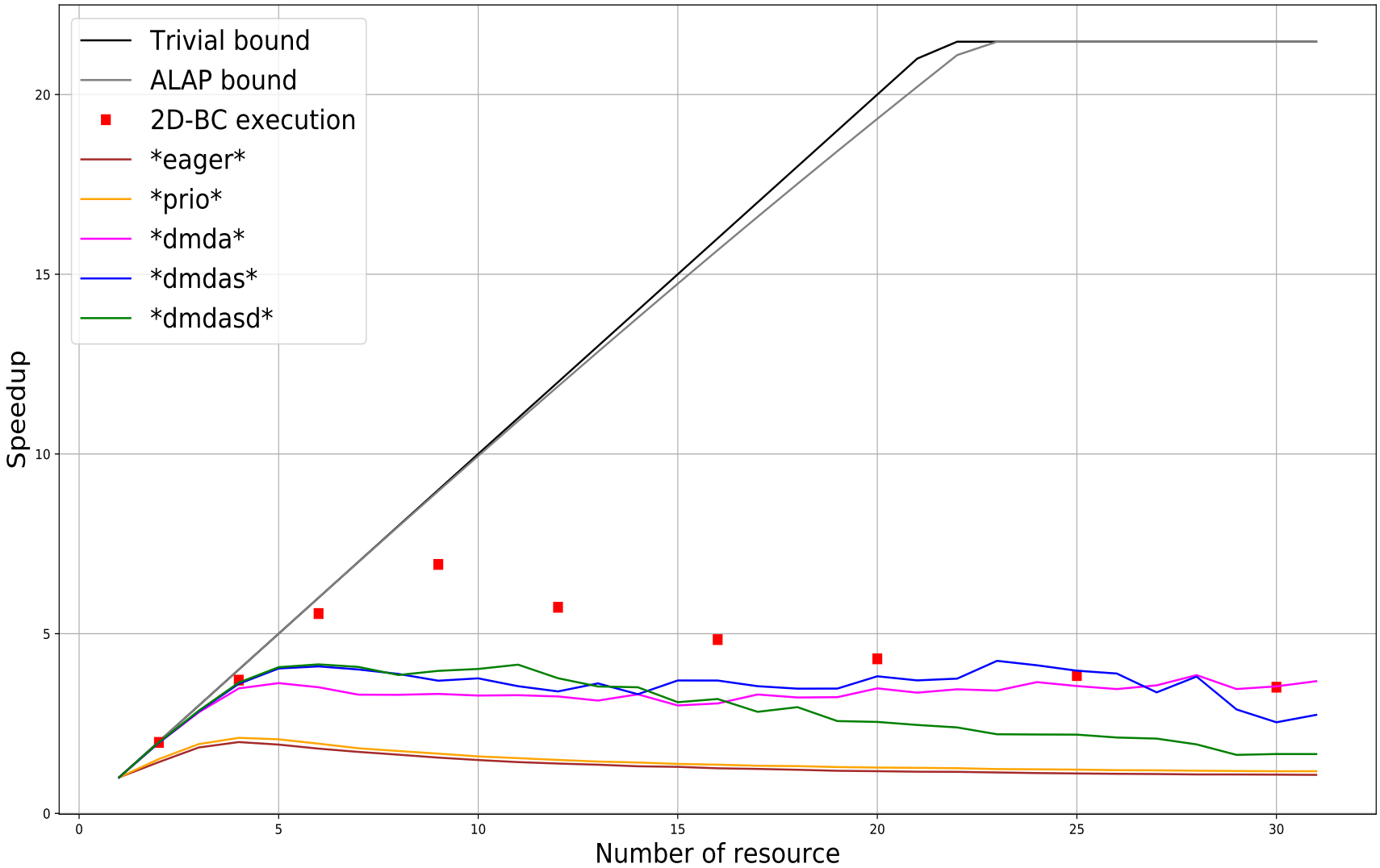


Figure: Default StarPU schedulers performance - GPU case ($N = 30$, $BW = 10GB/s$)

3.2 - Dynamic Allocation

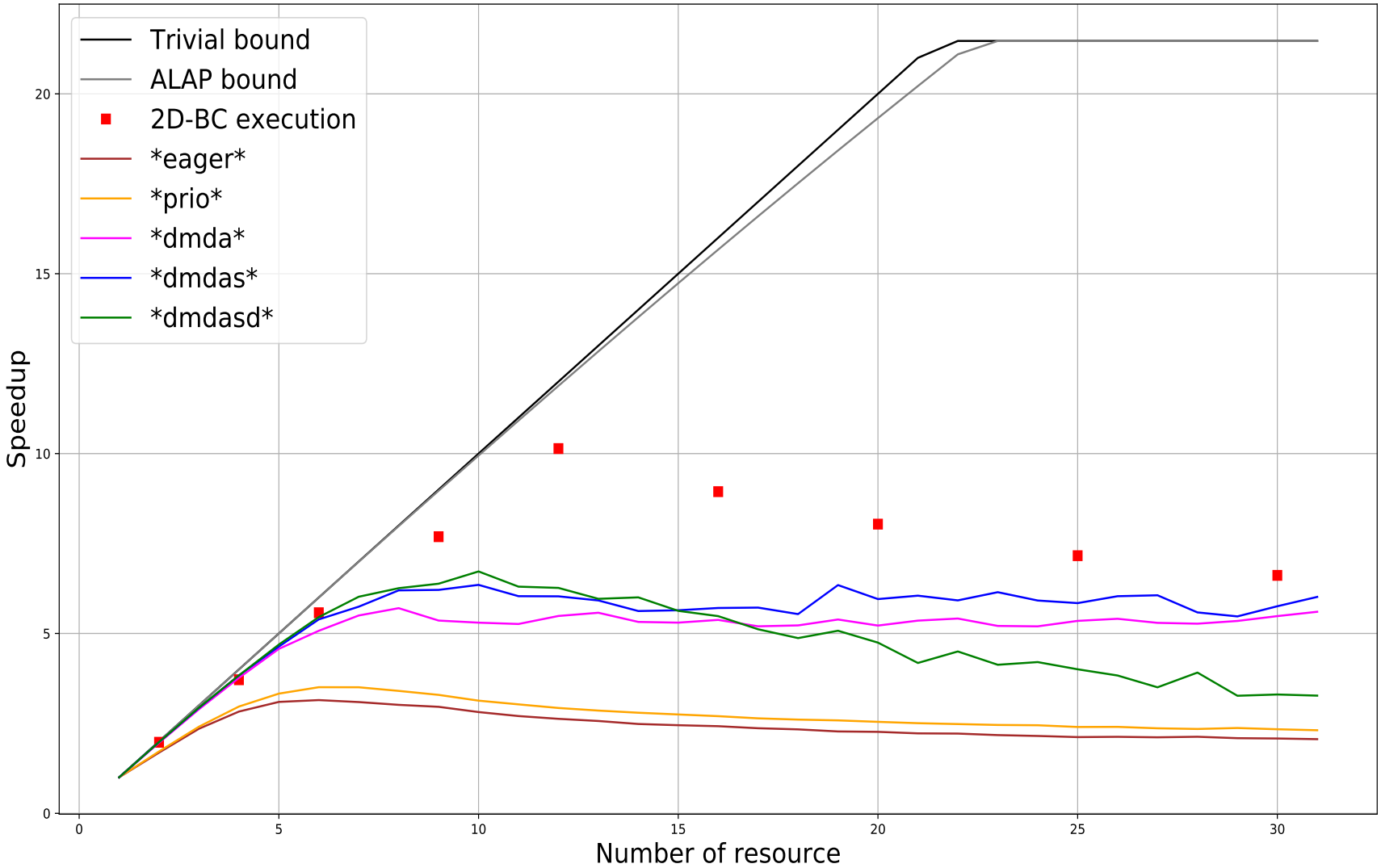


Figure: Default StarPU schedulers performance - GPU case ($N = 30$, $BW = 20GB/s$)

3.2 - Dynamic Allocation

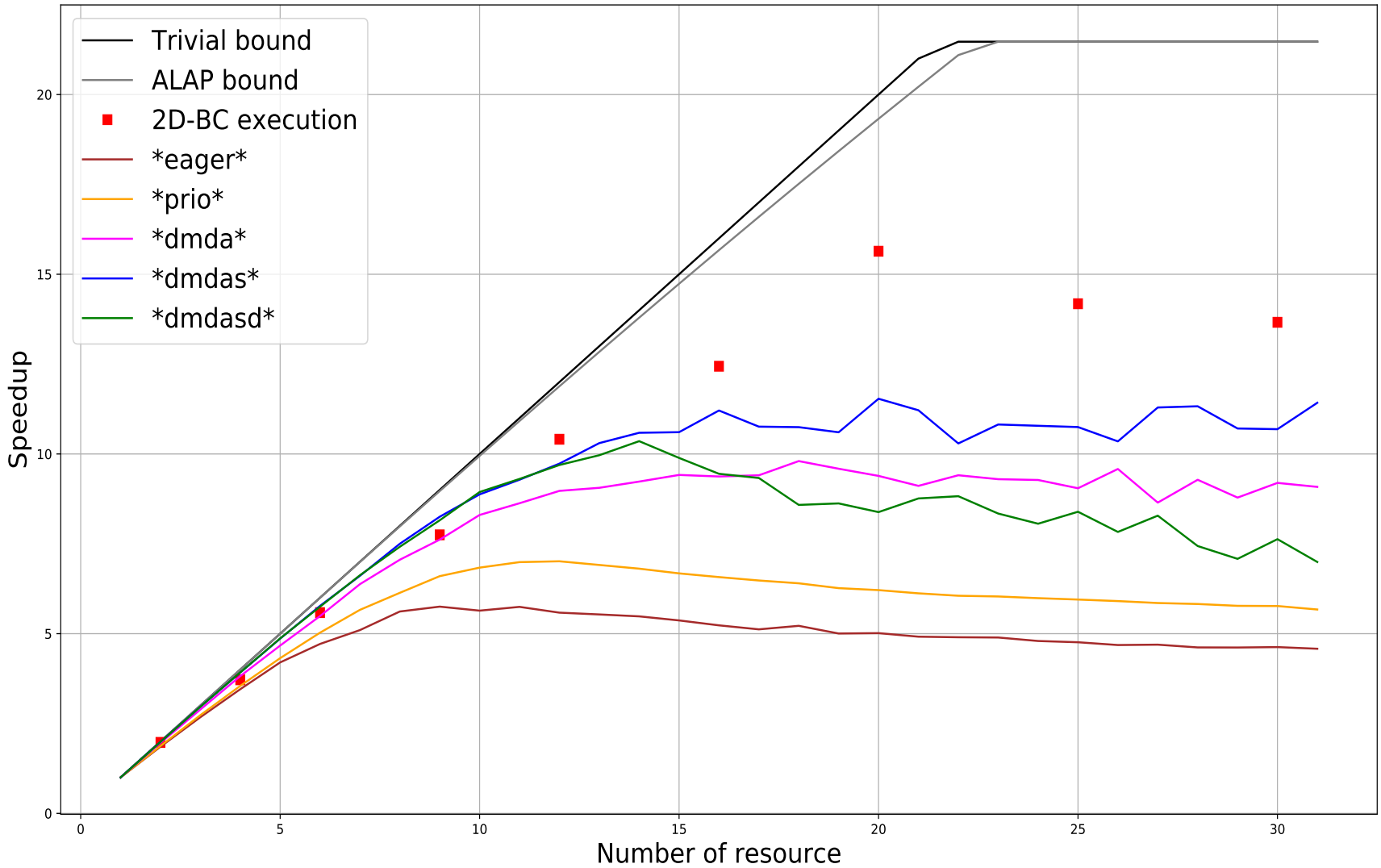


Figure: Default StarPU schedulers performance - GPU case ($N = 30$, $BW = 50GB/s$)

3.2 - Dynamic Allocation

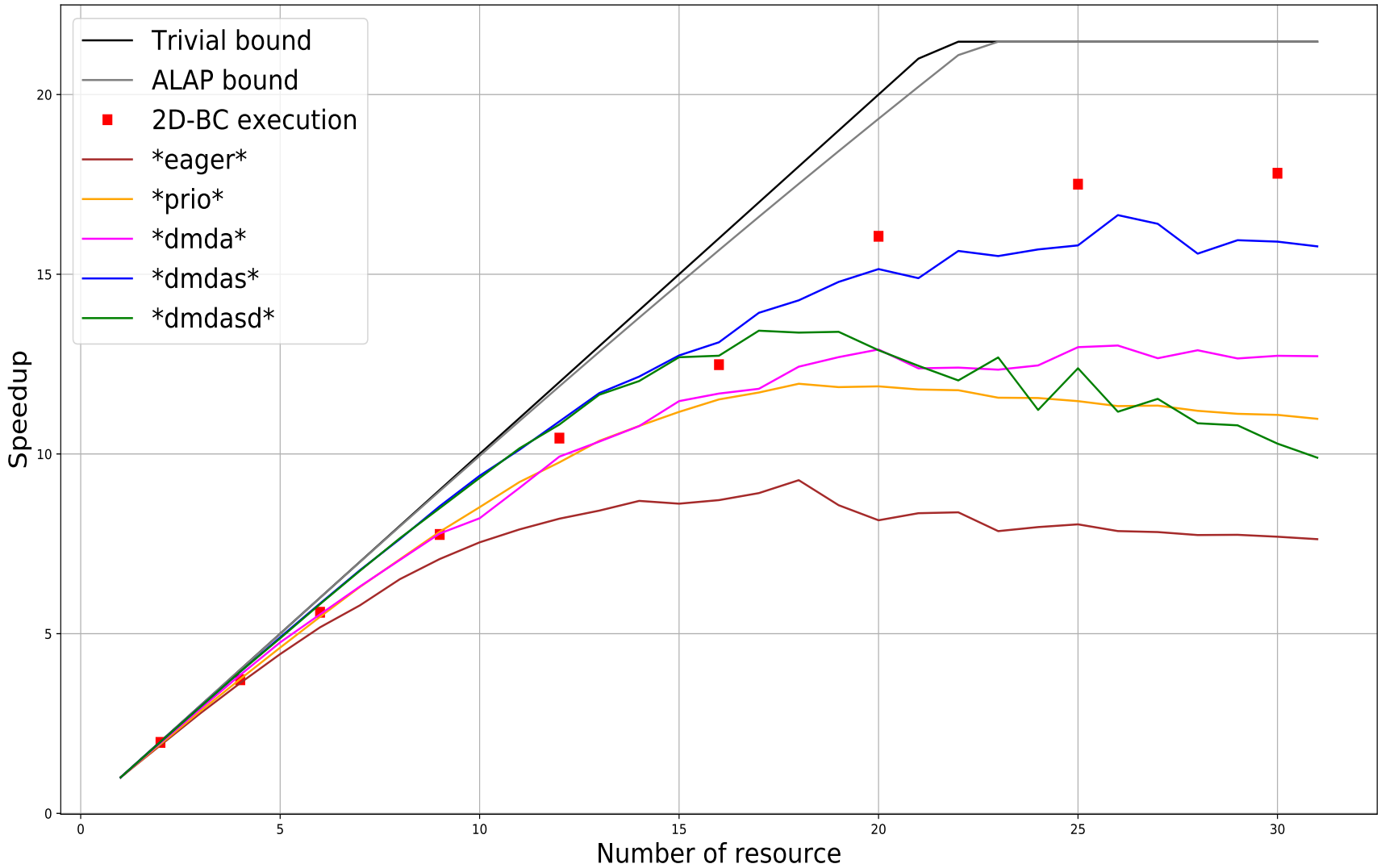


Figure: Default StarPU schedulers performance - GPU case ($N = 30$, $BW = 100GB/s$)

3.2 - Dynamic Allocation

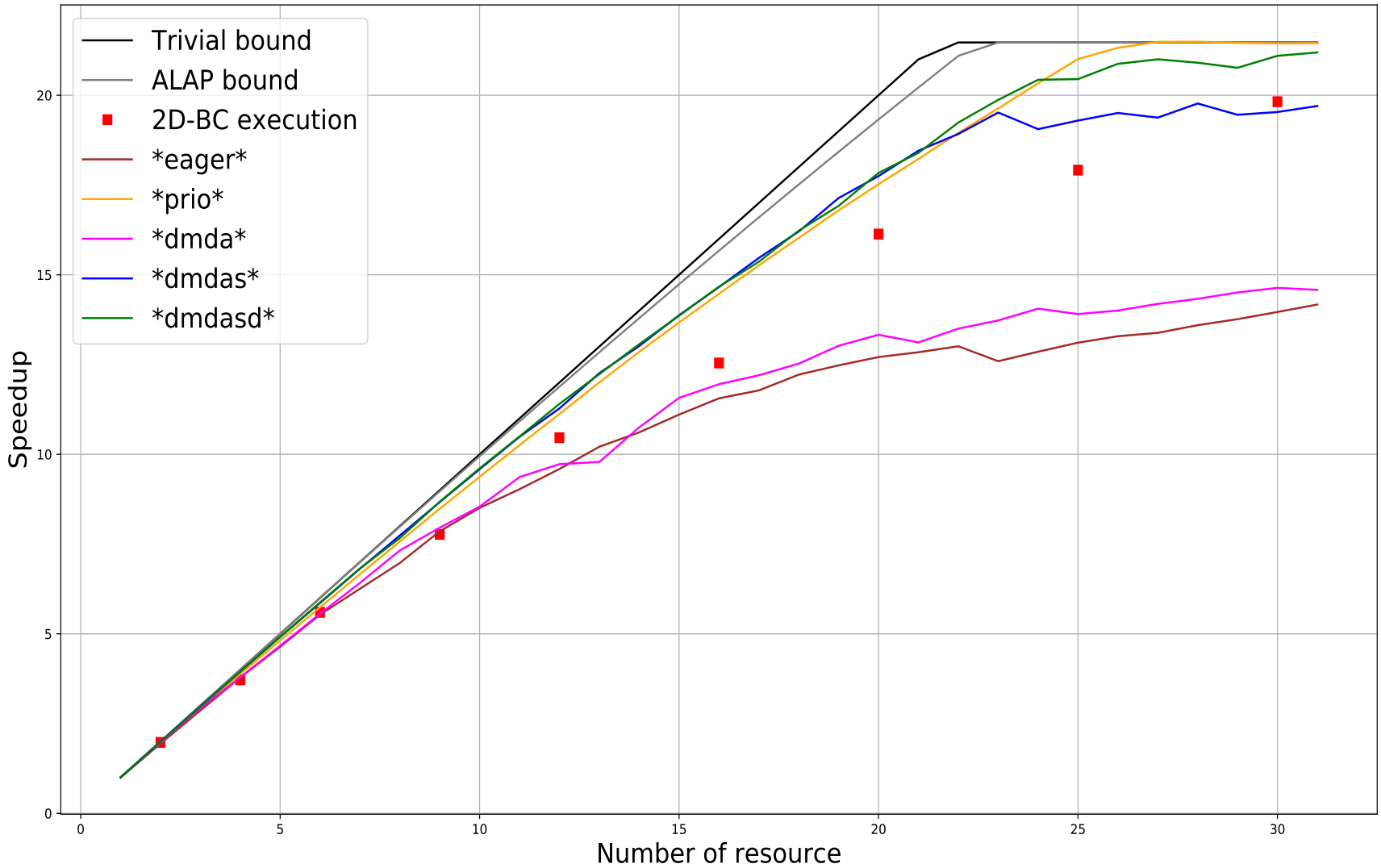


Figure: Default StarPU schedulers performance - GPU case ($N = 30$, $BW = 1000 GB/s$)

3.2 - Dynamic Allocation

Observations

- Tight communication constraints: `eager` and `prio` shows poor performance.
- No communication constraints. `prio` $\approx$ `dmdaX` and almost optimal.

- `dmda`: load of each queue >> data transfert cost: **balancing oriented**
  good load balancing but no priority $\Rightarrow$ poor scheduling
- `dmdasd`: data transfer cost critical: **data locality oriented**
  with priorities $\Rightarrow$ good scheduling but bad load balancing

`dmdas`: **trade-off** strategy $\rightarrow$ shows the best overall results.

3.2 - Dynamic Allocation

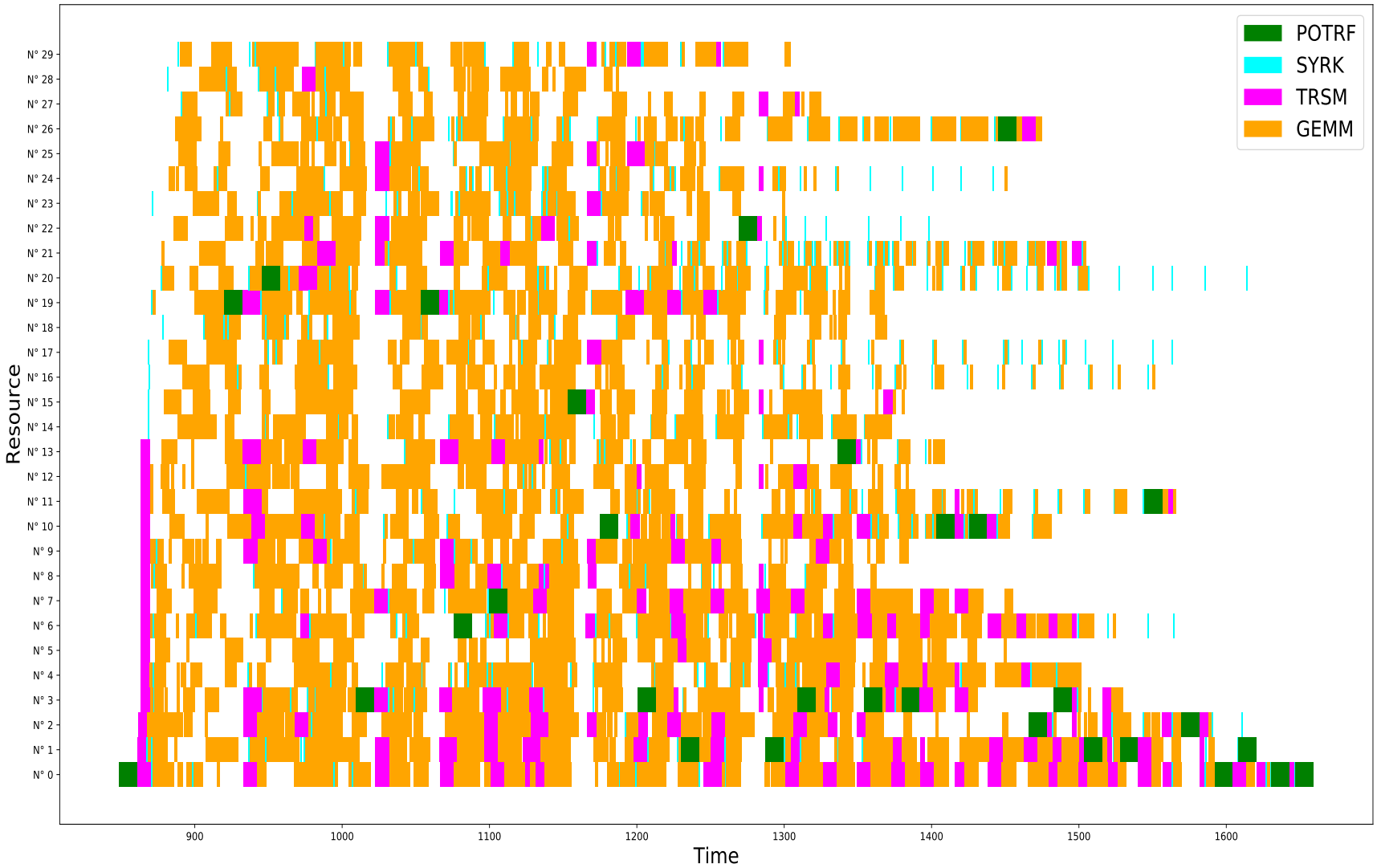


Figure: Gantt diagram of *dmda* execution - GPU case ($N = 30$, $BW = 100GB/s$, $P = 30$)

3.2 - Dynamic Allocation

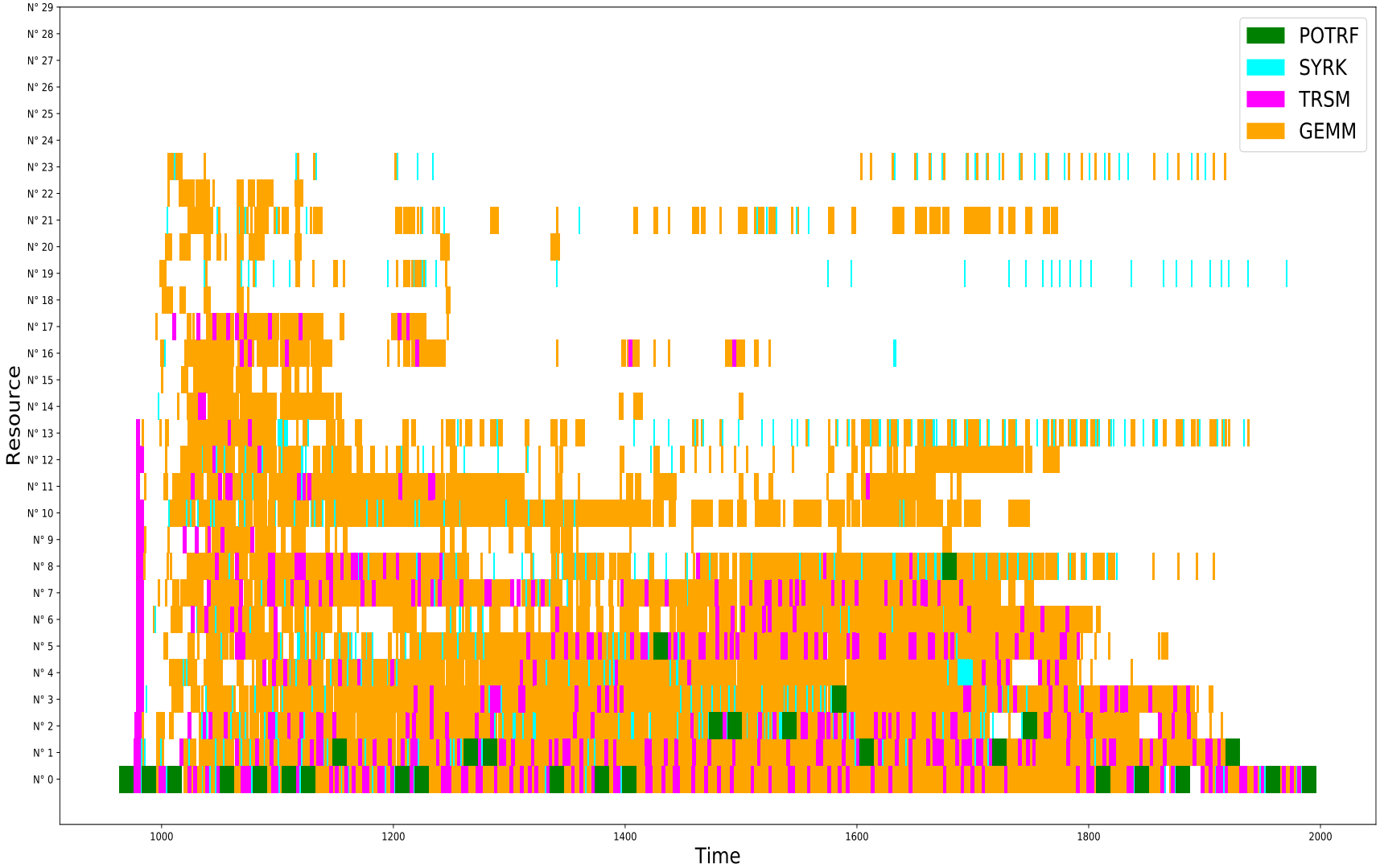


Figure: Gantt diagram of *dmdasd* execution - GPU case ($N = 30$, $BW = 100GB/s$, $P = 30$)

3.2 - Dynamic Allocation

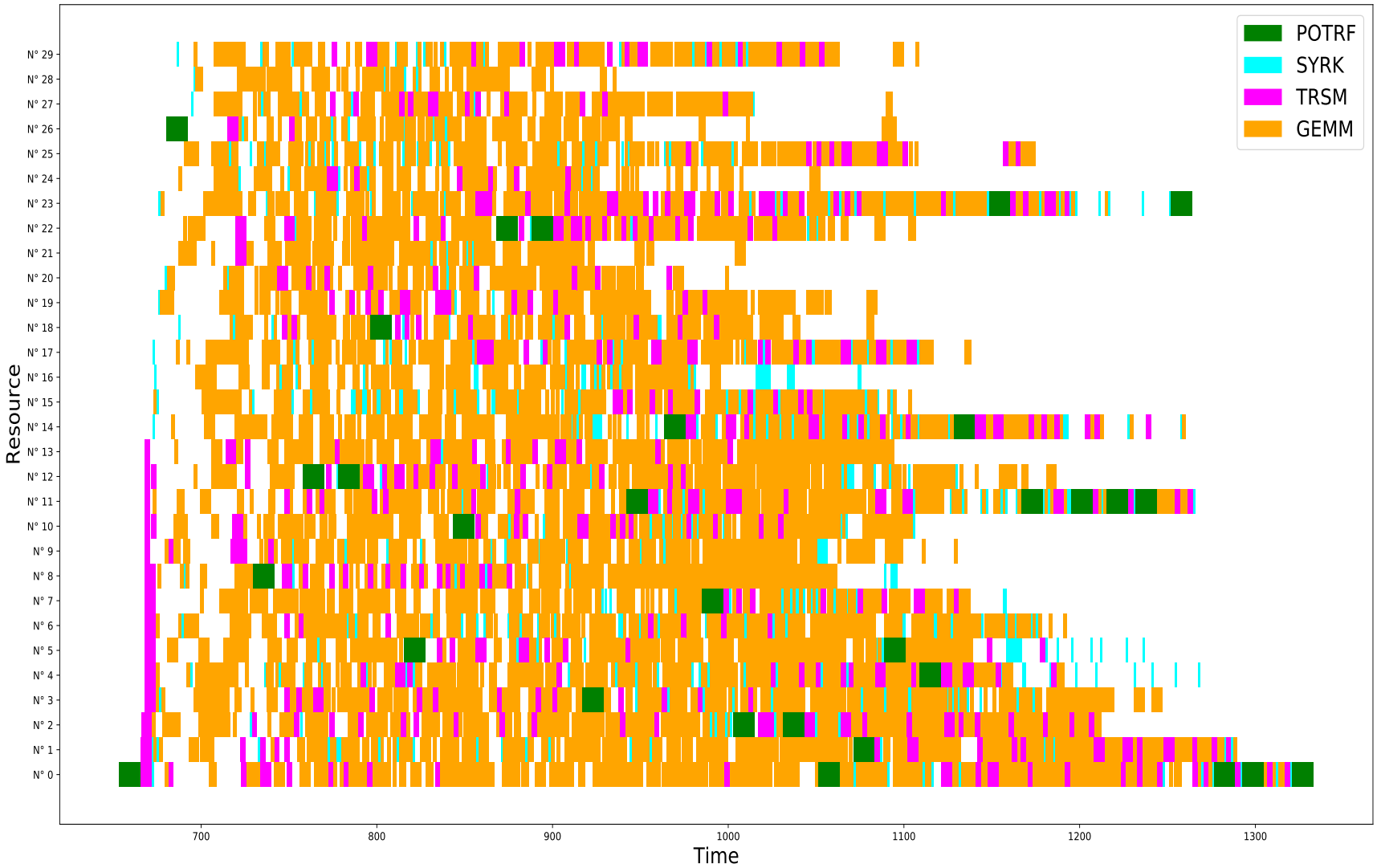


Figure: Gantt diagram of *dmdas* execution - GPU case ($N = 30$, $BW = 100GB/s$, $P = 30$)

3.2 - Dynamic Allocation

Observations

- Tight communication constraints: `eager` and `prio` shows poor performance.
- No communication constraints. `prio` $\approx$ `dmdaX` and almost optimal.

- `dmda`: load of each queue >> data transfert cost: **balancing oriented**
  good load balancing but no priority $\Rightarrow$ poor scheduling
- `dmdasd`: data transfer cost critical: **data locality oriented**
  with priorities $\Rightarrow$ good scheduling but bad load balancing

`dmdas`: **trade-off** strategy $\rightarrow$ shows the best overall results.

- Giving unconditional **priority to tasks on** $\mathcal{CP}$ with `dmdasd`: less prioritised tasks constantly postponed $\Rightarrow$ **bottleneck** at the end of execution.
- Default **data transfer estimation very pessimistic**
  $\Rightarrow$ `dmdasd` keep tasks on the same processors.

3.2 - Dynamic Allocation

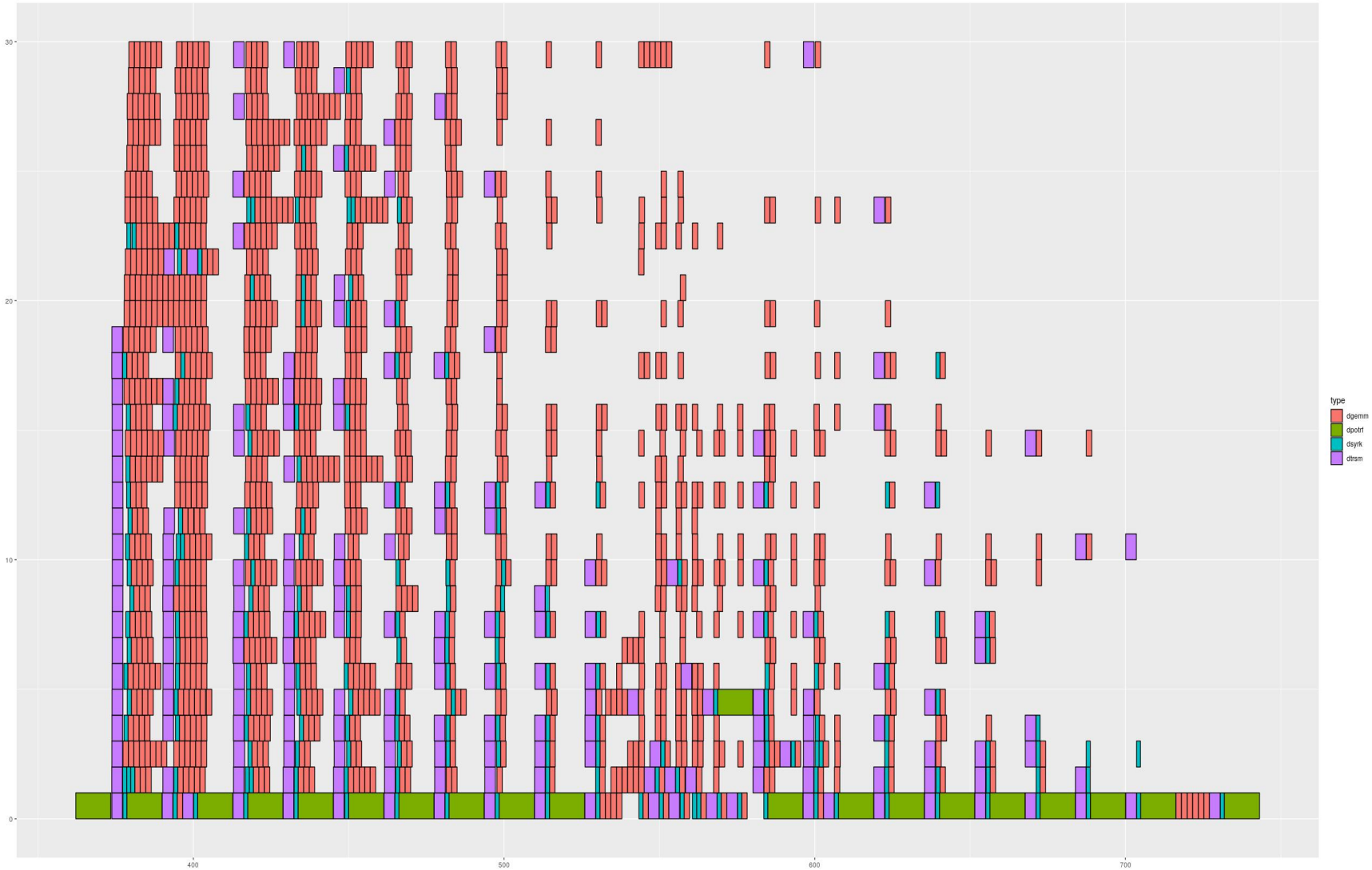


Figure: Gantt diagram of *dmdasd* execution using per-type priorities - GPU case ($N = 20$, $BW = 1000GB/s$, $P = 30$)

3.2 - Dynamic Allocation

Tweaking default schedulers

- carefully select **priorities**
- better **estimation of data transfer cost**:
  - "cheating" with bandwidth seen by StarPU
  - using number of current data transfers

$\Rightarrow$ prevent degradation of `dmdasd` performance when P increases

$\Rightarrow$ does not improve much more than default `dmdas`

3.2 - Dynamic Allocation

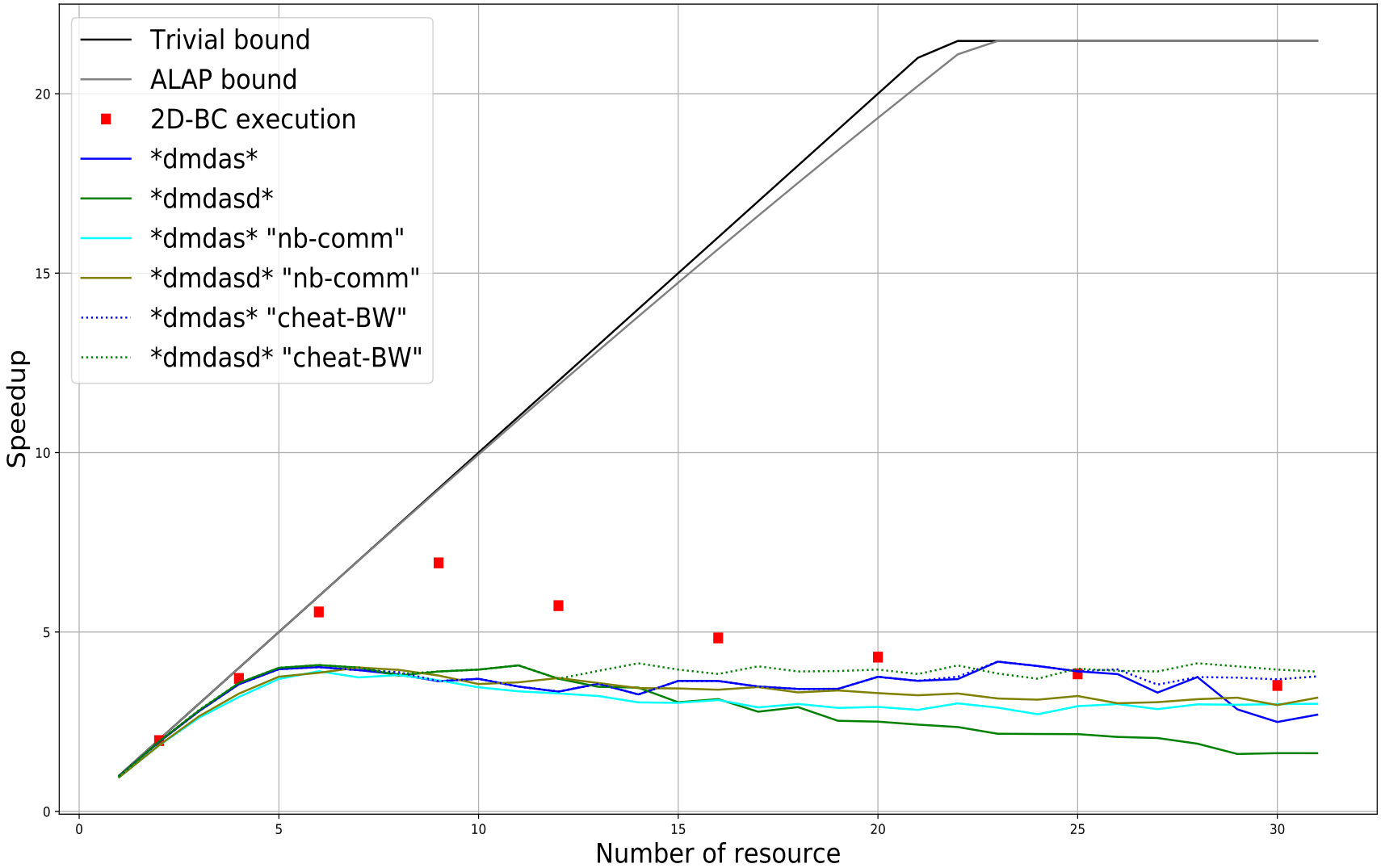


Figure: Modified schedulers performance - GPU case ($N = 30$, $BW = 10GB/s$)

3.2 - Dynamic Allocation

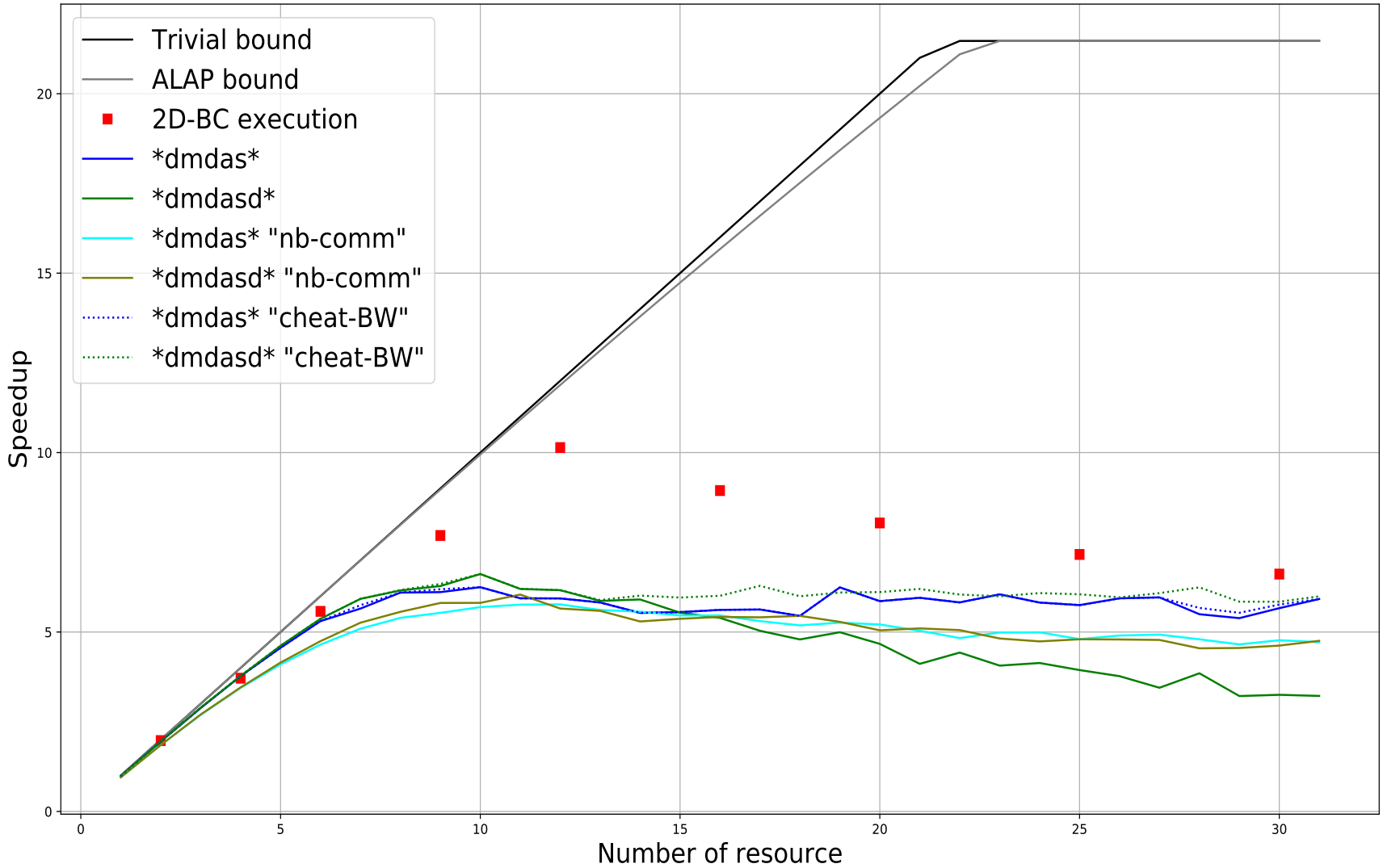


Figure: Modified schedulers performance - GPU case ($N = 30$, $BW = 20GB/s$)

3.2 - Dynamic Allocation

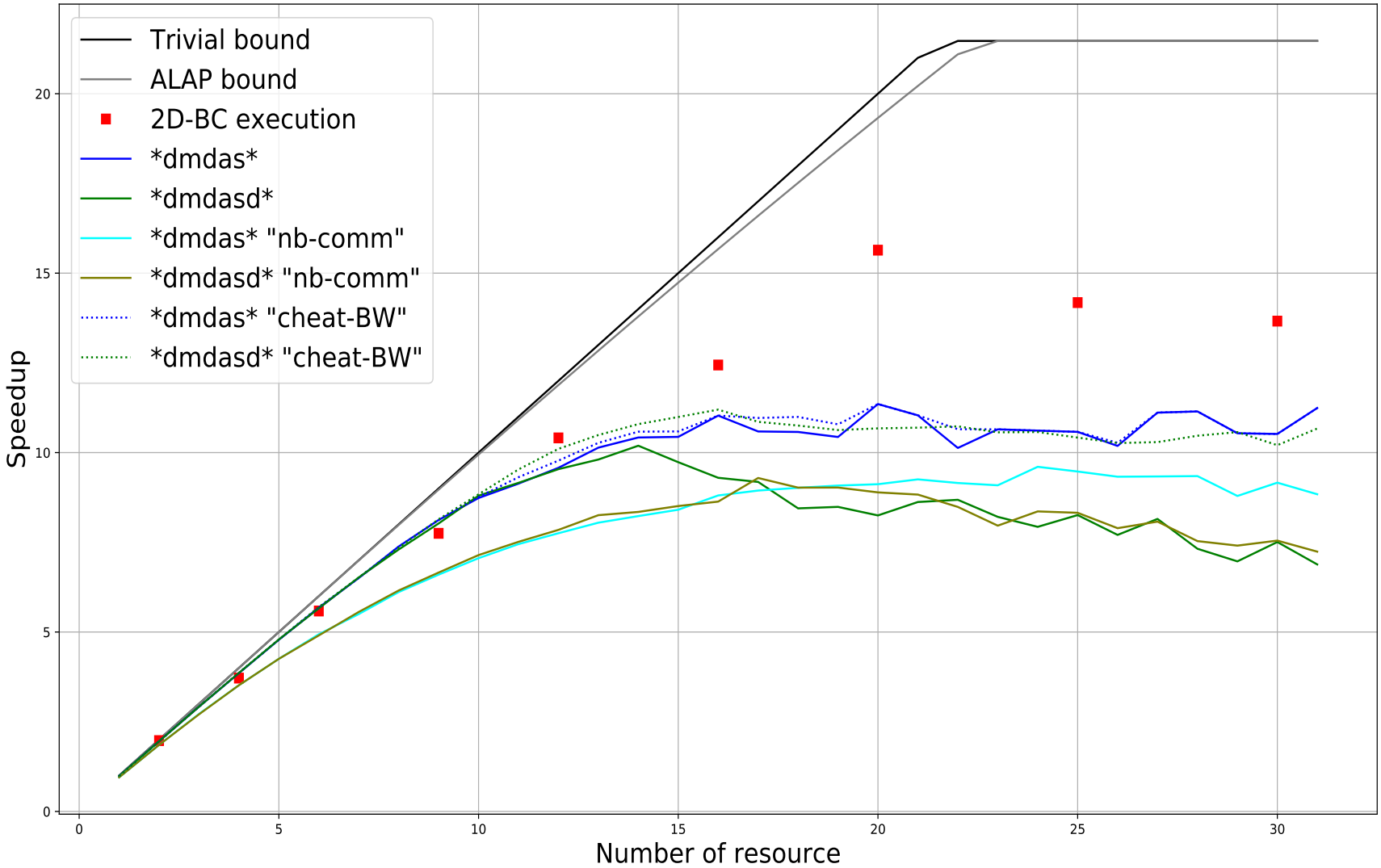


Figure: Modified schedulers performance - GPU case ($N = 30$, $BW = 50GB/s$)

3.2 - Dynamic Allocation

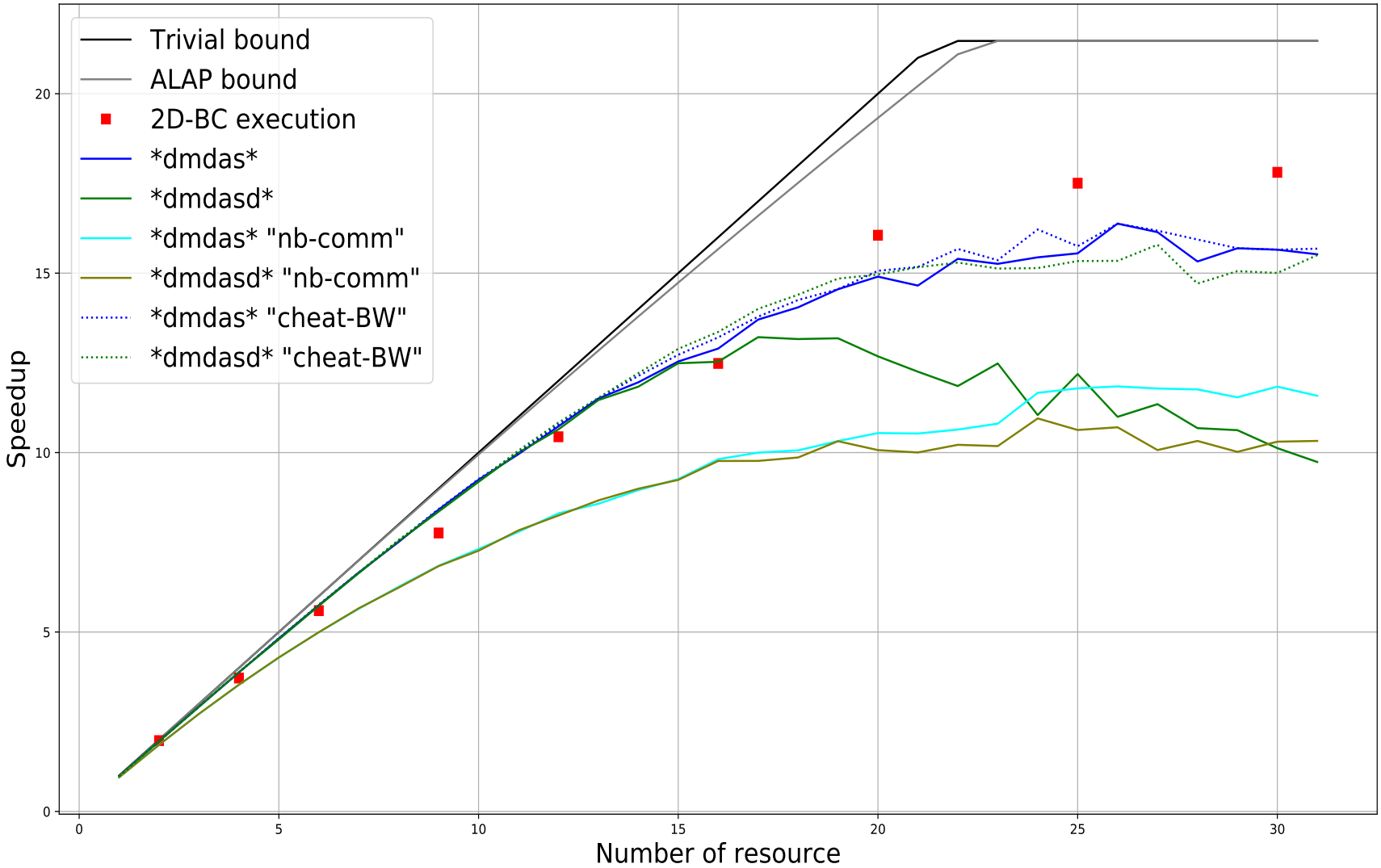


Figure: Modified schedulers performance - GPU case ($N = 30$, $BW = 100GB/s$)

3.2 - Dynamic Allocation

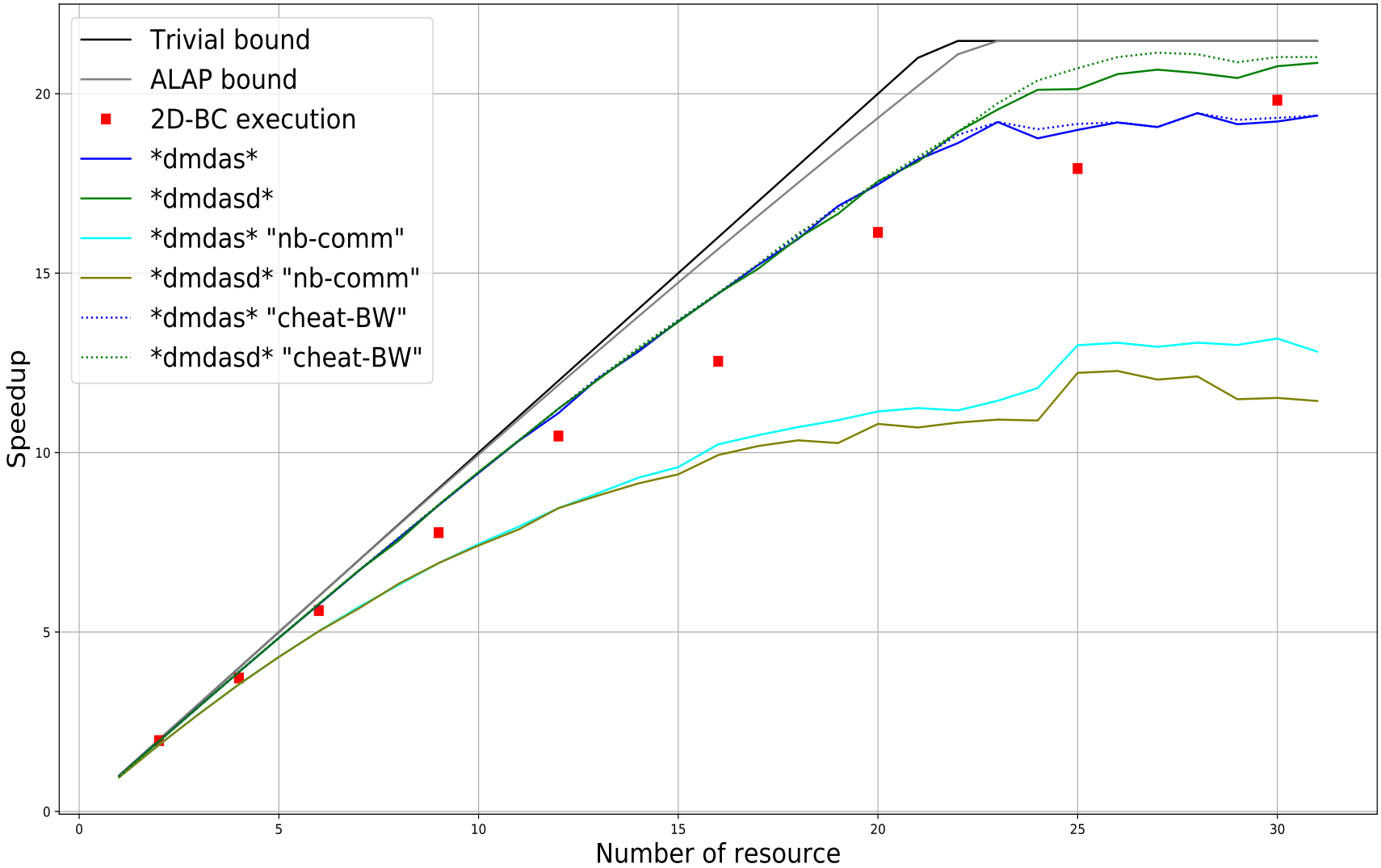


Figure: Modified schedulers performance - GPU case ($N = 30$, $BW = 1000 GB/s$)

4 - Future Work

General conclusion

Communication bounds are unsatisfactory $\rightarrow$ to combined with work progress.
BUT: not trivial...

Perspectives

- **generalize/automatize** ALAP bound method to other kernels
- applicability to **heterogeneous** tiles and/or resources
- **guide exploration** of more efficient **static allocation** inspired by dynamic ones

4 - Future Work

References

- **[1]** Beaumont, Langou, Quach, Shilova - *A Makespan Lower Bound for the Tiled Cholesky Factorization based on ALAP Schedule* - EuroPar 2020
- **[2]** Olivry, Langou, Pouchet, Sadayappan, Rastello *Automated Derivation of Parametric Data Movement Lower Bounds for Affine Programs* - 2019
- **[3]** Ballard, Demmel, Holtz, Schwartz - *Communication-Optimal Parallel And Sequential Cholesky Decomposition* - SPAA 2009
- **[4]** Irony, Toledo, Tiskin - *Communication lower bounds for distributed-memory matrix multiplication* - JPDC 64 Nov. 2004
- **[5]** Hong, Kung - *I/O Complexity: The Red-Blue Pebble Game* - Symposium on Theory of Computing 1981

Thank you for your attention

Questions?