Computer Physics Communications 244 (2019) 180–186

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

# Parallel high-performance multi-beam multi-bunch simulations

Sondre Vik Furuseth [a,b,*], Xavier Buffat [a]

[a] European Organization for Nuclear Research (CERN), CH-1211 Geneva, Switzerland
[b] École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland

ARTICLE INFO

*Article history:*
Received 7 January 2019
Received in revised form 16 April 2019
Accepted 6 June 2019
Available online 17 June 2019

*Keywords:*
Causality
Beam–beam
Wakefield
Circular colliders

ABSTRACT

Coherent multi-bunch interactions can cause severe impacts on the beams in circular colliders. To understand the dynamics of such interactions, the accelerator physics community relies on high-performance tracking codes. Ensuring causality produces a severe bottleneck in simulations including both intra-beam and inter-beam interactions between the bunches in the beams. `COMBI` was developed to study such interactions. Its parallel algorithm greatly limits its efficiency if the number of bunches outnumbers the number of separate calculations per turn, or when the calculations vary in computational complexity. A new parallel algorithm, `COMBIp`, addresses the identified challenges with improved partitioning of the calculations and asynchronous communication between the bunches. The unavoidable bottleneck is now a limitation on the number of compute nodes that can be applied efficiently, instead of a limitation on the physics that can be simulated efficiently. The modifications have led to a great speedup from the old parallel algorithm, up to the number of bunches per beam.


## 1. Introduction

The dynamics of particle beams in circular colliders can be so involved that the most complex configurations must be evaluated by numerical simulations. Each beam consists of multiple bunches. The bunches are both affected by external electromagnetic fields produced by the machine, and by interactions with each other. There are: (i) independent, intra-bunch effects such as the forces from the various magnets; (ii) intra-beam, inter-bunch interactions such as the kicks from electromagnetic wakefields and electron clouds [1]; (iii) inter-beam, inter-bunch interactions close to the collision points, called beam–beam interactions [2]. The different effects modeled in simulations will be referred to as calculations.

There exists a wide library of simulation codes developed specifically for circular particle colliders, exploiting the parallel infrastructure of modern computers in different ways. How a code is parallelized depends on what the code is designed to study. `BeamBeam3D` is a parallel particle-in-cell code designed to model beam–beam interactions in three dimensions in detail [3]. Each bunch is represented by macroparticles, and six phase space coordinates represent each macroparticle. The parallelization is done with a particle-field decomposition, slicing each bunch longitudinally, and distributing the slices on separate processes. The impact of the slices of beam 1 on the slices of beam 2 is calculated in order. `PyHEADTAIL` studies the interplay of one beam with the machine through electromagnetic wakefields [4]. In the multi-bunch version, multiple bunches are allocated to each process. The same calculation is performed for all bunches simultaneously. `COMBI` was designed for beam–beam interactions, but can also study effects such as wakefields [5,6]. Each bunch is allocated to its own process, and all bunches are controlled by a master process. The bunches are distributed on a circular grid as in a collider, and synchronized at the end of each calculation to ensure causality.

An unbreakable rule of physics is causality. When simulating coherent multi-beam, multi-bunch effects, causality puts strong constraints on the order of the calculations. This will be discussed in the following. Based on the analysis of the challenges, a new parallel algorithm, which has been implemented in `COMBI`, will be presented. It was inspired by putting the calculations for each bunch in separate pipelines, and is referred to as `COMBIp`. The performance of `COMBIp` is then analyzed and compared to that of the old parallel algorithm.

## 2. Challenges of causality in simulations

A circular collider contains two beams going in opposite directions, as shown in Fig. 1. We will use this example with 8 bunches per beam repetitively in this paper. The locations must be traversed in order, meaning 19, 0, 1, ... for a bunch in beam 1 (B1), and 1, 0, 19, ... for a bunch in beam 2 (B2), to ensure causality. At each location there can be a calculation to be performed, or not. The different types of necessary calculations and their impact on ensuring causality will now be discussed.

* Corresponding author at: European Organization for Nuclear Research (CERN), CH-1211 Geneva, Switzerland.
*E-mail address:* sondre.vik.furuseth@cern.ch (S.V. Furuseth).

https://doi.org/10.1016/j.cpc.2019.06.006

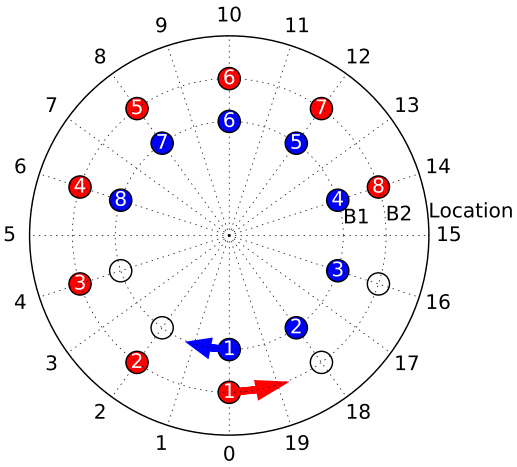



**Fig. 1.** A circular collider model where two beams (B1 in blue, B2 in red) move in opposite directions. Both beams have 8 bunches in a row followed by two empty slots. There are twice as many locations for calculations as bunch slots, to be able to model beam–beam interactions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

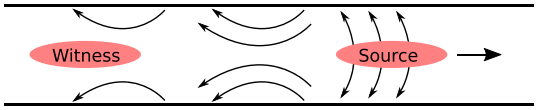
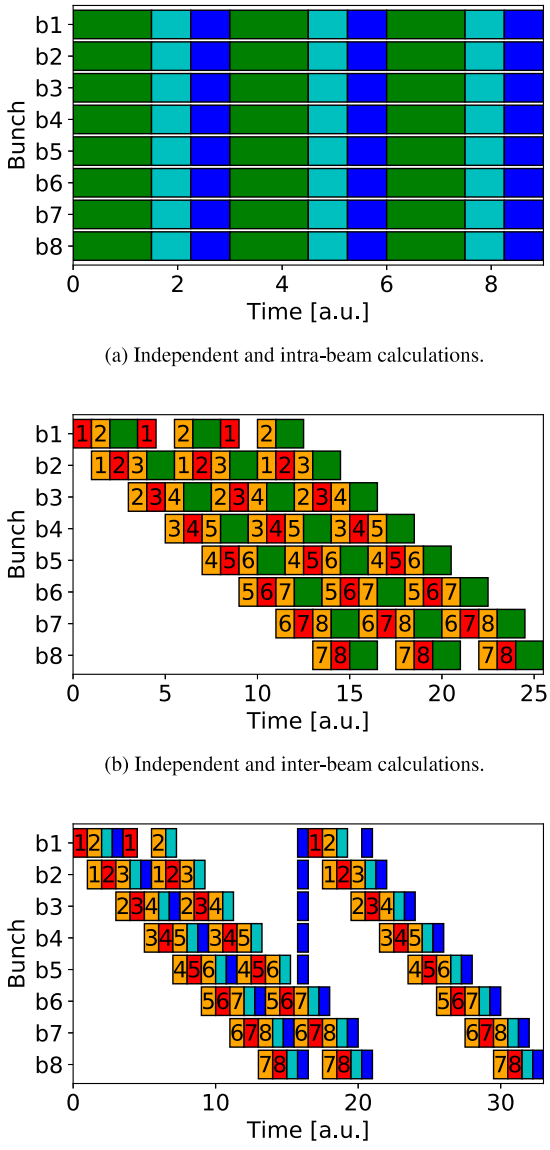



**Fig. 2.** Schematic of intra-beam calculations. A source charge traverses the machine and leaves some remnant mark on its surroundings. Witness charges, either further back in the same bunch or in other bunches, get affected by the modified surroundings. In the ultra-relativistic limit, a source charge cannot affect witness charges in front of itself. The effect is typically weaker the greater the distance is between the source and the witness.

Independent, intra-bunch calculations only depend on the affected bunch, and require no communication between the bunches. That is for example the bending from the magnets around the machine. It can also be the case for simplified models of inter-bunch calculations, such as the weak–strong model of beam–beam interactions [7]. These calculations have to be done in order for each bunch separately, to ensure causality. However, in a multi-bunch simulation, these calculations do not require any synchronization between the bunches.

Intra-beam, inter-bunch calculations, from now on referred to as intra-beam calculations, can be represented as in Fig. 2. From now on we discuss wakefields. How the inter-bunch dependencies affect the order of these calculations is visualized in Fig. 3(a). The wakefields produced by each bunch can be calculated simultaneously for all the bunches in the same beam. Then the wakefields must be communicated to the trailing bunches, before the kicks from the wakefields can be calculated. Hence, a bunch cannot overtake another bunch beyond this calculation. Without inter-beam calculations, the individual bunches can easily be parallelized, by performing each calculation simultaneously for every bunch. That is what is done in `PyHEADTAIL` [4].

Inter-beam calculations can be represented as in Fig. 4. In modern circular colliders, the two beams are kept separated except for close to the points where the beams are brought into collision. How the inter-bunch dependencies affect the order of these calculations is visualized in Fig. 3(b), assuming one long-range calculation on each side of a head-on calculation at location 0. Bunch $n$ has to calculate its interaction with bunch $n-1$, then $n$, then $n+1$ of the other beam. Since this is required for both beams, bunch $n-1$ has to finish its calculation with a bunch of the other beam, before bunch $n$ can start its calculation with the




(a) Independent and intra-beam calculations.

(b) Independent and inter-beam calculations.

(c) Intra-beam and inter-beam calculations.

**Fig. 3.** Gantt charts [8] of the most efficient flow of calculations for 8 bunches (b1–b8) of one beam, while still ensuring causality. The order of the calculations for each bunch is read left to right. If there is a white gap, it means that the next calculation cannot yet be initiated, because it requires input from another bunch. Green is independent calculations, Light blue is the calculation of that bunch's wakefields, Blue is the impact of other bunches' wakefields, Red is the head-on beam–beam calculation, and Orange is a long-range beam–beam calculation. The numbers on the beam–beam calculation blocks refer to which bunch of the other beam the calculation is with, assuming the head-on calculation is in location 0 in Fig. 1. The necessary calculations for 3 turns are displayed. The time of the different types of calculation is set artificially. These charts are only meant to show the inter-bunch dependencies. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

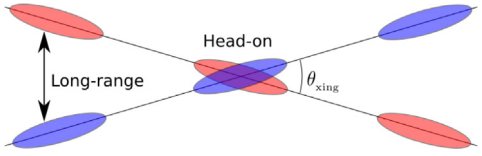
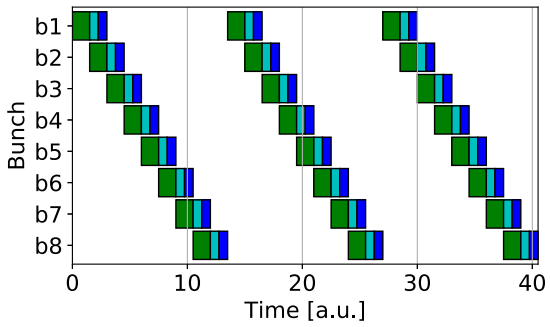
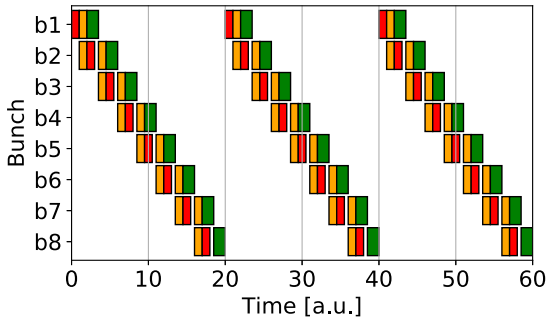


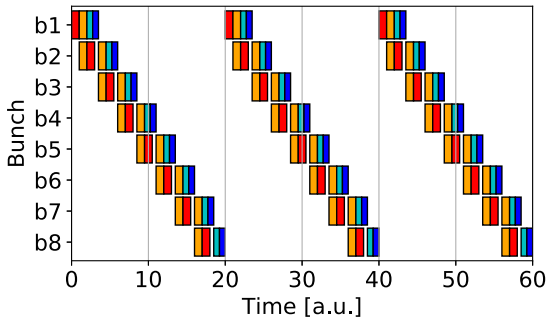

**Fig. 4.** Schematic of inter-beam calculations around a collision point in a circular collider. There are one head-on calculation in the center, and a number of long-range calculations on both sides.
*Source:* Courtesy of [7].




(a) Independent and intra-beam calculations in COMBI.




(b) Independent and inter-beam calculations in COMBI.




(c) Intra-beam and inter-beam calculations in COMBI.

**Fig. 5.** Gantt charts [8] of the flow of calculations for 8 bunches (b1–b8) of one beam, also including the synchronization after each calculation in COMBI. Otherwise, the charts contain exactly the same calculations as in Fig. 3, with the same explanations of the colors. The numbers on the beam–beam calculation blocks have been omitted here for readability. The calculation of a bunch's wakefields and the calculation of the impact of other bunches' wakefields, is a single calculation in COMBI.

same bunch. Without intra-beam calculations, or filling of every slot in the collider model, the calculations can still be parallelized efficiently, by letting the bunches do different calculations simultaneously. That was the inspiration of the old algorithm in COMBI.

The bottleneck, preventing these parallel multi-beam, multi-bunch simulations from being efficient, arises when one includes both intra-beam and inter-beam calculations in the same simulation, as visualized in Fig. 3(c). The intra-beam calculations prefer the bunches to do the same calculation in parallel, while the inter-beam calculations prefer the bunches to do different calculations in parallel. In result, there is a sizeable amount of white space corresponding to time when no calculation can be done for a given bunch. The reason in this model is that the kick from wakefields on bunch 1 (b1) in turn 2, cannot be calculated before bunch 8 has been there in turn 1 to produce its wakefield.

## 3. Parallel algorithm

To combat the inefficiencies of COMBI and the inherent bottleneck of multi-beam, multi-bunch simulations, we have developed a new parallel algorithm. It has been implemented in the already existing and well-used code COMBI, meaning that the physics is modeled exactly as before.

### 3.1. COMBI

COMBI is implemented with a hybrid OpenMP-MPI parallelization. The MPI parallelization employs a master–worker algorithm, with one master process overall, and one worker process per bunch [9]. Each calculation is parallelized internally in each worker process with OpenMP [10]. The algorithm is shown in detail in Algorithm 1. The bunches of each beam are fixed to a circular grid as in Fig. 1, and rotated synchronously to their next locations in the collider model. The master process tells all workers what the bunches have to do, whether it is a calculation or nothing. After the workers are done, they send a completion confirmation each to the master. Therefore, all workers have to wait for the slowest calculation to finish, before the bunch grids are rotated to their next locations.

Considering the examples for the most efficient flows of calculations possible in Fig. 3, the equivalent Gantt charts for the worker processes in COMBI will include more white gaps between the calculations, visualized in Fig. 5. This algorithm was developed with beam–beam calculations in mind. It limits the efficiency especially when the number of bunches outnumbers the number of calculations, or when the calculations are of varying numerical complexity. The wall time per turn, assuming that the number of bunches is larger than the number of calculations, is expected to be in the order of $\max_i\{t_i\} \cdot \max_j\{N_{bj}\} \cdot (N_{b1} + N_{b2})/$

**Algorithm 1** COMBI

```
1:  procedure MASTER(rank)
2:    PARSE input files
3:    SEND bunch details to Workers
4:    CREATE collider
5:    // list of calculations at location
6:    for turn in numberOfTurns do
7:      for step in numberOfLocations do
8:        Rotate bunch grids one step
9:        SEND calculations to Workers
10:       RECEIVE confirmation from Workers
11:     end for
12:   end for
13:   SEND Abort to Workers
14: end procedure
15:
16: procedure WORKER(rank)
17:   RECEIVE bunch details from Master
18:   CREATE bunch
19:   while TRUE do
20:     RECEIVE message from Master
21:     if message is calculation then
22:       Perform calculation
23:       SEND confirmation to Master
24:     else if message is Abort then
25:       BREAK
26:     end if
27:   end while
28: end procedure
```

$N_{\text{cores}}$, where $t_i$ is the wall time of the calculation at location $i$ with 1 core, $N_{bj}$ is the number of bunches in beam $j$, and $N_{\text{cores}}$ is the total number of cores. $N_{\text{cores}}$ is assumed moderate. This formula is meant for comparison with the new algorithm, not for predicting the actual wall time.

### 3.2. COMBIp

The new parallelization algorithm, `COMBIp`, is detailed in Algorithm 2. The key advances from the original algorithm, are: (i) the bunches are autonomous; (ii) all calculations for each bunch are put in separate pipelines; (iii) the communication is asynchronous; (iv) there can be multiple bunches per process. The processes are still parallelized with `MPI`. The communication between the bunches is handled with `MPI` as well, even if the bunches are on the same process. Additional memory is allocated for the messages, to prevent the memory from being overwritten before it is received. The size of this additional memory is negligible compared to the memory required to store the phase space coordinates of the macro-particles that constitute the bunches. Each calculation handled by the processes is still parallelized with `OpenMP`. Hence, each process can maximally exploit the cores on one full compute node.

Since the bunches are autonomous, the need for the master process is gone. Therefore, the communication from and to the master is no longer needed. Since the bunches' calculations are in separate pipelines, the bunches are no longer waiting after each calculation for all of them to finish. Since the communication is asynchronous, the processes do not have to stall while waiting for other processes to respond, freeing up the processing power. To achieve this, the calculations that require communication with other bunches have been split in two, first sending and then receiving the required information. This separation is especially useful for the intra-beam calculations, where each bunch must communicate with all other bunches in the same beam. Due to the first 3 advances, the Gantt charts for the processes are the most efficient ones for the bunches shown in Fig. 3.

It is still of utmost importance to ensure that causality is preserved. Because the calculations are in separate pipelines, the independent calculations are automatically performed in the correct order. Mistakes can only arise from the calculations that require communication between the bunches. When a bunch arrives at a location where the calculation requires communication, the bunch knows which bunch(es) it is supposed to communicate with. First, it will test if the previous message(s), using the assigned memory buffer, has been received with `MPI_Test`. If yes, it will non-blockingly send the new message(s) with `MPI_Isend` to the bunch(es) it is interacting with. Then, the process allows the other bunches on the process to do their next calculations. Next, the bunch will non-blockingly check with `MPI_Iprobe` if the message(s) from the bunch(es) it is interacting with has been sent. If yes, it will start receiving the message(s) with `MPI_Recv`. Because of this split of the calculation in two, first sending and then receiving, the bunch can and will wait after it has sent its message(s), until it can receive the required message(s) to perform the calculation. The bunch cannot progress to the next calculation in the pipeline before it has completed this calculation. Thus, the minimum required synchronization is ensured, and causality is preserved, due to how the inter-bunch communication is implemented.

The main goal of this work was to make efficient simulations including both intra-beam and inter-beam calculations. It was displayed in Fig. 3(c) that in such simulations, individual bunches would have to wait due to causality, even if the synchronization was as efficient as possible. However, when bunch 1 has to wait in this example, bunch 5 can perform its calculations. Since the new algorithm allows for having multiple bunches per process, a process does not have to stall when a bunch has to wait. This last key advance, thus allows the simulation efficiency to go beyond the most efficient flow of calculations shown in Fig. 3. The wall time per turn is expected to be in the order of $\sum_i t_i \cdot (N_{b1} + N_{b2})/N_{\text{cores}}$. Hence, it can be up to $\max_j\{N_{bj}\}$ times faster than the old algorithm.

**Algorithm 2** `COMBIp`

```
1:  procedure PIPELINE(rank)
2:    PARSE input files
3:    CREATE bunches for this rank
4:    SET bunch.step to 0
5:    CREATE bunch.pipeline
6:    // all calculations for one bunch
7:    while all bunches are not done do
8:      for bunch in bunches do
9:        if bunch is done then
10:         CONTINUE
11:       end if
12:       SET calculation to pipeline[step]
13:       if ready for calculation then
14:         Perform calculation
15:         INCREMENT step
16:         Update whether the bunch is done
17:       end if
18:     end for
19:   end while
20: end procedure
```

## 4. Timing results

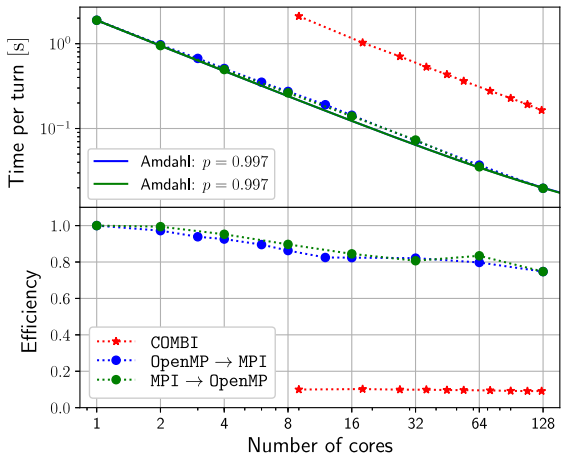
The parallel algorithms have been tested in detail for their performance in relevant configurations. All simulations have been run on the Deneb cluster at EPFL, with nodes containing 2 Ivy Bridge processors running at 2.6 GHz, with 8 cores each [11]. All simulations have been run with $10^6$ particles per bunch, for 100 turns. They were run 4 times, whereupon the average wall time was calculated. The wall time per turn and efficiency will be presented. The efficiency is a measure of how well a parallel algorithm exploits additional computing resources [12], and is here defined as

$$\text{Efficiency} = \frac{t_{\text{ref}}}{t_{\text{par}}} \cdot \frac{N_{\text{cores,ref}}/s_{\text{ref}}}{N_{\text{cores,par}}/s_{\text{par}}}, \tag{1}$$

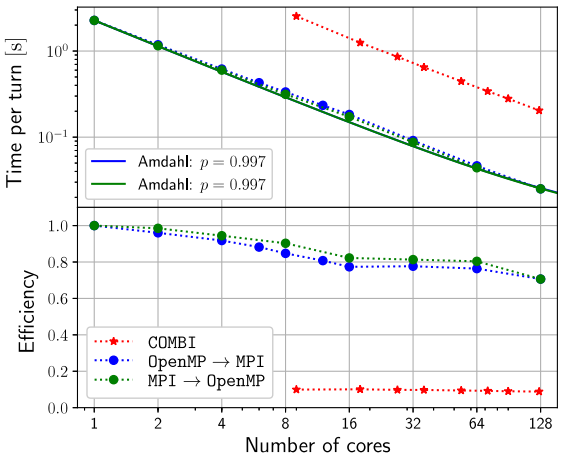
where $t$, $N_{\text{cores}}$ and $s$ are the wall time, number of cores and problem size of a simulation. The subscripts ref and par correspond to a reference simulation and the parallel simulation for which we want to know the efficiency. Typically $N_{\text{cores,ref}} = 1$, but the more general definition in Eq. (1) is more suitable for this study. For a perfectly scaling algorithm, the efficiency is 1. The speedup from the reference simulation to a parallel simulation will also be referred to, defined as

$$\text{Speedup} = \frac{t_{\text{ref}}}{t_{\text{par}}}. \tag{2}$$

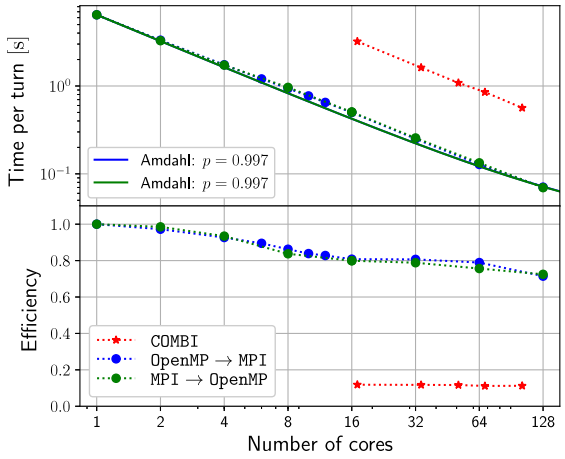
The timings have been measured for models like the ones presented in Fig. 3. The models consisted of various combinations of the following: (i) linear phase advance including chromaticity, independent for each bunch ("Ind"); (ii) a section with one head-on calculation and one long-range calculation on each side ("BB"); (iii) a wakefield calculation ("Wake"). The relative wall times of the different types of calculations, neglecting the communication, are presented in Table 1. These values are meant to show the ratio between the different calculations, not to be compared to the integrated wall time per turn that will be presented.



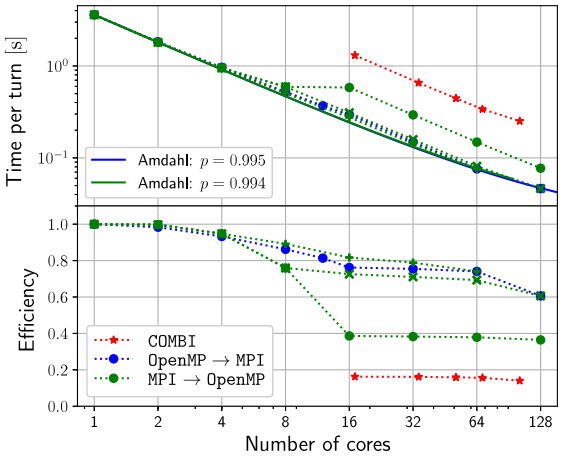

(a) 1 beam ($N_{b1} = 8$). 1 independent calculation.




(b) 1 beam ($N_{b1} = 8$). 1 wakefield calculation and 1 independent calculation, as in Fig. 3a.




(c) 2 beams ($N_{b1} = 8 = N_{b2}$). 3 beam-beam calculations and 1 independent calculation, as in Fig. 3b.




(d) 2 beams ($N_{b1} = 8 = N_{b2}$). 1 wakefield calculation and 3 beam-beam calculations, as in Fig. 3c.

**Fig. 6.** Strong scaling of `COMBI` and `COMBIp` for different collider models, including independent, beam–beam and wakefield calculations. In (d) "`MPI` → `OpenMP`" goes to 4 (+), 8 (x) and 16 (•) `MPI` processes, before it scales further with multiple threads per process.

**Table 1**
Relative time of different calculations in the scalings.

| Calculation | Time [$t_{BB}$] |
|---|---|
| Independent | 9.6 |
| Beam–beam[a] | 1 |
| Wakefield | 0.84 |
| Bunch moments[b] | 1.64 |

[a]Head-on and long-range interactions are calculated with the same function, only varying in the introduced offset.
[b]Calculated prior to sending the required information for beam–beam and wakefield calculations, and to be stored after each turn.

### 4.1. Strong scaling

The strong scaling of an algorithm is how the wall time varies with the number of cores for a fixed total problem size. That is, $s_{\text{ref}} = s_{\text{par}}$ in Eq. (1) [12]. The reference simulation was calculated with `COMBIp` with 1 core. In this case, the speedup of a perfectly scaling algorithm would be equal to $N_{\text{cores,par}}$. For the following simulations, the strong scaling will be presented for three schemes, the first with the old algorithm and the last two with the new algorithm: (i) `COMBI`: Start with $(N_{b1} + N_{b2}) + 1$ processes of 1 thread, then increase the number of threads; (ii) `OpenMP`→`MPI`: First scale a single process up to 16 threads, then increase the number of processes; (iii) `MPI`→`OpenMP`: First scale up to $(N_{b1} + N_{b2})$ processes, then add multiple threads per process. One exception will be noted in the text. All simulations have 8 bunches per beam. Hyper-threading has been forced off to get reliable scaling data. The reduction of wall time is fitted to Amdahl's law, and the parallel part $p$ is given in the legends for the new algorithm [12]. The expected and actual scaling depend on the required synchronization of the calculations in the collider model, as discussed in Section 2.

Consider first one beam requiring a single independent calculation per turn. The new algorithm with one core is expected to finish slightly faster than the old algorithm with $N_{b1} + 1$ cores. This is because the master–worker algorithm requires communication for the synchronization and work organization, and only one worker is computing at a time. The scaling is presented in Fig. 6(a). In this case, `COMBI` using 9 cores achieves a speedup of 0.89 (efficiency of 0.099) from the serial version of `COMBIp`, approximately as expected. The results are rather independent of how you increase the number of cores with the new algorithm, both schemes show that the simulation has a parallel part of

99.7%. Because this collider model requires neither communication nor synchronization between the bunches, this is assumed to be the best performance achievable by the COMBIp-algorithm.

For simulations with one wakefield calculation and one independent calculation per turn, the scaling is presented in Fig. 6(b). It is slightly better to increase the number of processes before the number of threads. That could be because these simulations require a substantial amount of MPI communication, including organization of the incoming wakefield kicks, which is currently performed by only one thread per process. COMBI using 9 cores achieves again a speedup of 0.89 (efficiency of 0.099) from the serial version of COMBIp. The efficiency of the new algorithm for this collider model is close to the best performance found above, as expected.

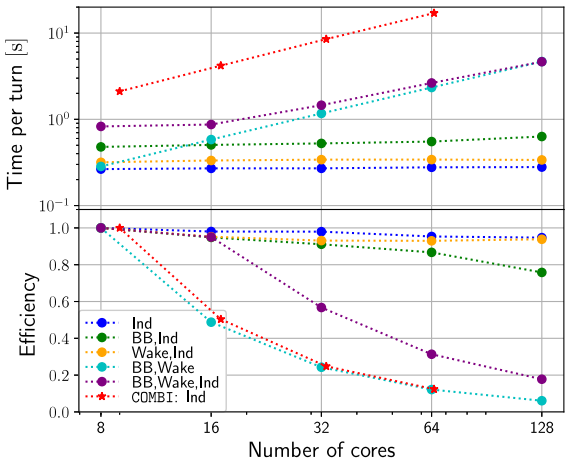
For simulations with one head-on calculation and one long-range calculation on each side, in addition to one independent calculation, the scaling is as in Fig. 6(c). COMBI requires a minimum of 17 cores, with which it achieves a speedup of 2.0 (efficiency of 0.12) from the serial version of COMBIp. COMBIp scales equally well for the two schemes, independently of how the cores are added. The efficiency of the new algorithm for this collider model is close to the best performance found above, as expected.

For simulations with three beam–beam interactions, as above, and one wakefield calculation, the scaling is as in Fig. 6(d). The MPI→OpenMP scaling of this configuration was performed in three ways, by initially going to 4, 8 and 16 MPI processes, before adding more threads per process. This was done to emphasize the bottleneck introduced in Section 2, which limits the performance in simulations with both intra-beam and inter-beam calculations, if there is only 1 bunch per MPI process. The 16 bunches were distributed evenly over the MPI processes. With only 1 bunch per process, the efficiency dropped to 0.4 with only 16 cores. There is a clear improvement by having multiple bunches on the same process. For this collider model, it is sufficient to have 4 bunches per MPI process. By allowing for bunches to share a process, the reduction of performance due to the bottleneck has been avoided.

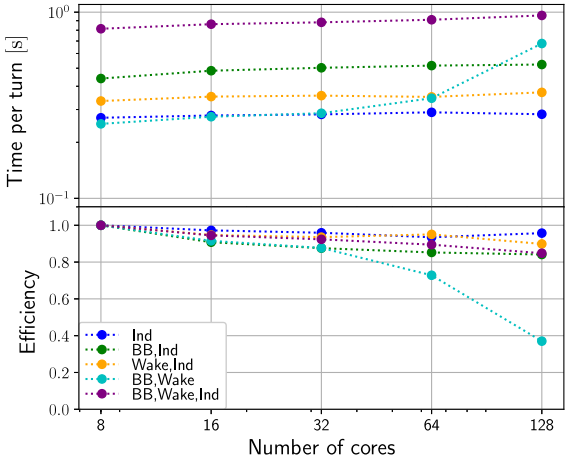
### 4.2. Weak scaling

The weak scaling of an algorithm is how the wall time varies with the number of cores for a fixed problem size per core [12]. Hence, the efficiency is equal to the speedup given by Eq. (2). To test the weak scaling, the bunch sizes were kept constant, while the total number of bunches were equal to the number of cores, subtracted 1 for the master process. It is typically easier to achieve a good weak scaling than strong scaling, hence the names. However, in multi-beam, multi-bunch simulations it can be more challenging to keep a high efficiency as the number of bunches increases, as we will see. The collider models are the same as in Fig. 6, plus one as in Fig. 6(d) also including two independent calculations per turn. The models with beam–beam calculations have $N_{\text{cores}}/2$ bunches per beam, while the others have $N_{\text{cores}}$ bunches in the first beam only. As the beam–beam calculations do not make sense for one bunch, the scalings start at a reference simulation for 8 bunches. The weak scaling of COMBI and COMBIp is presented in Fig. 7. Note that the efficiency of COMBI is calculated relative to a reference simulation that is also run with COMBI, to better show the behavior as the problem size increases.

The weak scaling with 1 bunch per 1-core-process is presented in Fig. 7(a). The efficiency of the COMBI algorithm falls quickly as the number of bunches increases, due to the synchronization after each calculation that was illustrated in Fig. 5. Note that the COMBI simulations were run with only the independent calculation, and should be compared to the blue curve labeled "Ind", which was




(a) 1 bunch per 1-core-process.




(b) 8 bunches per 8-core-process.

**Fig. 7.** Weak scaling for different collider models, with independent (Ind), beam–beam (BB) and wakefield (Wake) calculations. The weak scaling was also calculated for COMBI with the independent calculation, for 1 bunch per worker process.

simulated with COMBIp. The bottleneck discussed in Section 2 for collider models with both intra-beam and inter-beam calculations, is clearly visible on the curves labeled "BB,Wake" and "BB,Wake,Ind", which both drop below an efficiency of 0.2. The simulations with beam–beam interactions become less effective starting from 64 cores. This can partly be explained by the white triangles in the beginning and end of Fig. 3(b), because only 100 turns are simulated. A part of the work required by the wakefield calculations scales with the number of bunches. This scaling is negligible up to 128 bunches, according to these results.

The weak scaling with 8 bunches sharing each 8-core-process is presented in Fig. 7(b). There is a clear improvement by having multiple bunches on the same process in COMBIp. The root limitation, as discussed in Section 2 qualitatively and shown here quantitatively, is how many bunches causality allows to be calculated in parallel. The solution is therefore to distribute more bunches on each process, such that each calculation takes a shorter wall time, instead of trying to calculate every bunch in parallel. That is what COMBIp does. By achieving better load balancing in this manner, the impact of the bottleneck is pushed to a higher number of cores, such that the achievable speedup is higher.

## 5. Discussion

The goal of the new algorithm is to study Landau damping of multi-bunch beam–beam modes in the presence of wakefields, requiring intra-beam and inter-beam calculations [13,14]. The solution to the inherent bottleneck was to distribute the bunches over fewer processes, and instead parallelize maximally each calculation within each process. The speedup of this method is limited by how many cores there are on each node in the utilized cluster. One could go one step further by dividing each individual bunch over multiple nodes on separate processes, but this would require a significant amount of additional communication and implementation, and expected to lead to an increased efficiency only in marginal cases. Therefore, there is a limited, albeit large, speedup that can be achieved by the new algorithm for a given simulation.

The scalings in Section 4 were calculated with a limited number of bunches, and simple collider models. In realistic models for the Large Hadron Collider (LHC), there will be more independent calculations, up to about 120 long-range and 4 head-on beam–beam calculations for each bunch, and up to 2808 bunches per beam [15]. Although the simulations will be computationally heavy, multiple bunches can be calculated simultaneously. Hence, a large number of processes can be active with good load balancing in `COMBIp`, and a significant speedup can be achieved.

## 6. Conclusion

The constraints due to causality in multi-beam, multi-bunch simulations have been discussed in this paper. Simulations with either intra-beam or inter-beam calculations can be performed efficiently. In simulations including both of them, causality leads to a bottleneck of how many bunches that can be calculated in parallel.

A new parallel algorithm has been implemented in `COMBI`, named `COMBIp`, to improve the efficiency. The key points of the new algorithm are that each bunch is autonomous, their calculations are ordered in a pipeline, the required communication between bunches is performed asynchronously, and there can be multiple bunches per process. The new algorithm has achieved a speedup of up to the number of bunches per beam, compared to the previous algorithm implemented in the code. The performance is close to independent of causality constraints when simulating collider models with either intra-beam calculations or inter-beam calculations in the ultra-relativistic limit. The predicted bottleneck for collider models with both is now a limit on the number of compute nodes that can be used efficiently, instead of a limit on the number of bunches that can be simulated efficiently. The new algorithm is designed to efficiently simulate realistic models of the LHC.

## Acknowledgments

The authors would like to thank K. Li for a fruitful discussion on the requirements of the simulations, and M. Moan for a discussion on algorithms in general. The calculations have been performed using the facilities of the Scientific IT and Application Support Center of EPFL. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

## References

[1] M. Dohlus, R. Wanzenberg, CERN Yellow Rep.: Sch. Proc. of the CAS-CERN Accel. Sch. on Intensity Limit. in Part. Beams, 3, CERN, Geneva, Switzerland, 2017, pp. 15–41, http://dx.doi.org/10.23730/CYRSP-2017-003.15.

[2] W. Herr, T. Pieloni, in: W. Herr (Ed.), Proc. CERN Accel. Sch.: Adv. Accel. Phys. Course, Trondheim, Norway, 2013, pp. 431–459. http://dx.doi.org/10.5170/CERN-2014-009.431.

[3] H. Shan, E. Strohmaier, J. Qiang, Int. J. High Perform. Comput. Appl. 22 (2008) 21–32, http://dx.doi.org/10.1177/1094342006085024.

[4] E. Métral, T. Argyropoulos, H. Bartosik, N. Biancacci, X. Buffat, J.F. Esteban Muller, W. Herr, G. Iadarola, A. Lasheen, K. Li, A. Oeftiger, T. Pieloni, D. Quartullo, G. Rumolo, B. Salvant, M. Schenk, E. Shaposhnikova, C. Tambasco, H. Timko, C. Zannini, A. Burov, D. Banfi, J. Barranco, N. Mounet, O. Boine-Frankenheim, U. Niedermayer, V. Kornilov, S. White, IEEE Trans. Nucl. Sci. 63 (2) (2016) 1001–1050, http://dx.doi.org/10.1109/TNS.2015.2513752.

[5] CERN ABP Computing web, `COMBI`, 2018, https://twiki.cern.ch/twiki/bin/view/ABPComputing/COMBI. (Accessed 18 2018).

[6] F.W. Jones, W. Herr, T. Pieloni, in: C. Petit-Jean-Genaz (Ed.), 2007 Part. Accel. Conf, IEEE, New Mexico, USA, 2007, pp. 3235–3237, http://dx.doi.org/10.1109/PAC.2007.4440383, THPAN007.

[7] S.V. Furuseth, X. Buffat, Phys. Rev. Accel. Beams 21 (8) (2018) 081002, http://dx.doi.org/10.1103/PhysRevAccelBeams.21.081002.

[8] W. Clark, W.N. Polakov, F.W. Trabold, The Gantt Chart, a Working Tool of Management, The Ronald press company, New York, 1922.

[9] T. Pieloni, (Ph.D. thesis), EPFL, Lausanne, Switzerland, 2008, http://dx.doi.org/10.5075/epfl-thesis-4211.

[10] X. Buffat, (Ph.D. thesis), EPFL, Lausanne, Switzerland, 2015, http://dx.doi.org/10.5075/epfl-thesis-6321.

[11] Scientific IT and application support, Deneb, 2018, https://scitas.epfl.ch/hardware/deneb/. (Accessed 1 2018).

[12] P.S. Pacheco, An Introduction to Parallel Programming, Elsevier Inc., Amsterdam, 2011.

[13] J.S. Berg, F. Ruggiero, Rep. CERN-SL-AP-96-071-AP, CERN, Geneva, 1996, http://inspirehep.net/record/428328.

[14] Y.I. Alexahin, Part. Accel. 59 (1998) 43–74, Rep. CERN-SL-96-064-AP, http://inspirehep.net/record/425478.

[15] O.S. Brüning, P. Collier, P. Lebrun. S. Myers, R. Ostojic, J. Poole, P. Proudlock, LHC Design Report, Rep. CERN-2004-003-V-1, CERN, Geneva, 2004, http://dx.doi.org/10.5170/CERN-2004-003-V-1.