PARALLEL BEAM-BEAM SIMULATION INCORPORATING MULTIPLE BUNCHES AND MULTIPLE INTERACTION REGIONS

F.W. Jones, TRIUMF, Vancouver, Canada
W. Herr, CERN, Geneva, Switzerland
T. Pieloni, EPFL, Lausanne and CERN, Geneva, Switzerland

Abstract

The simulation code COMBI has been developed to enable the study of coherent beam-beam effects in the full collision scenario of the LHC, with multiple bunches interacting at multiple crossing points over many turns. The program structure and input are conceived in a general way which allows arbitrary numbers and placements of bunches and interaction points (IP's), together with procedural options for head-on and parasitic collisions (in the strong-strong sense), beam transport, statistics gathering, harmonic analysis, and periodic output of simulation data. The scale of this problem, once we go beyond the simplest case of a pair of bunches interacting once per turn, quickly escalates into the parallel computing arena, and herein we will describe the construction of an MPI-based version of COMBI able to utilize arbitrary numbers of processors to support efficient calculation of multi-bunch multi-IP interactions and transport. Implementing the parallel version did not require extensive disruption of the basic computational routines, since all MPI functionality is isolated in a separate layer of steering routines. After an overview of the basic methods and numerical components of the code, the computational framework will be described in detail and parallel efficiency and scalability of the code will be evaluated. Initial results are shown for an example 4-IP LHC collision scheme.

INTRODUCTION

For the LHC and other colliders involving large numbers of bunches and multiple interaction regions, a logical progression in the study of coherent beam-beam effects is to account for the different collision patterns[1] experienced by the bunches. These are influenced by the bunch filling patterns (possibly different) in the two rings, and the location and number of IP's in operation. Moreover, around each crossing point will be grouped a series of parasitic collisions where bunches are in close proximity. The collision patterns define distinct equivalence classes for bunches, within which there will be unique coherent oscillation spectra and possible origins of instability.

Starting with a collision/transport model for a bunch pair and a single IP, parallel programming methods can be applied to study as many collision patterns as possible (with given computing cluster resources) via collective simulation of trains of bunches and multiple IP's. With appropriate allocation of processors to tasks, this problem lends itself well to message-passing parallelism on commodity clusters and scales up readily to allow a significant advance in the scope of this type of collider simulation.

COLLISION AND TRANSPORT MODEL

In developing the parallel COMBI code one of the intentions is to be able to accommodate collision models at varying levels of detail, from rigid-bunch and soft-Gaussian treatments to self-consistent field-solve/kick algorithms. For this initial development we chose the middle ground of the soft-Gaussian model, coupled with linear matrix transport, which can provide qualitatively correct results in coherent oscillation spectra and yet remains accessible to small-scale multi-bunch computations on a single processor, thus allowing a meaningful assessment of parallel efficiency.

The structure of the computational core, first developed[2] in a single-processor setting is based on nested stepping over turns, bunches, and ring locations. Zero, one, or two bunches (one from each beam) may occupy a ring location ("slot") and to any location there may be assigned an *action code* that specifies a computation to be done. In the non-parallel code these actions are performed in sequence for each ring location, whereas in the parallel code the computations are distributed to a set of processes which do the computations locally and with local data.

PARALLEL ALGORITHM

The parallel schema (Figure 1) comprises one *supervisor* process and a set of *worker* processes, with tasks as follows:

Supervisor

The supervisor initiates all computations performed by the worker processes: essentially it is an executive "shell" which performs no computations except for minimal bookkeeping of bunch positions, and hence is free to rapidly send task instructions to a large number of workers and keep them maximally occupied. The supervisor activities comprise the following:

1. Reads the input files (run parameters, fill patterns, collision and transport patterns).
2. Initialises the worker processes.

3. Maintains data structures for collision patterns, fill patterns, and bunch stepping.
4. At each step sends coordinated task messages to the workers for all active bunches.
5. Receives completion messages from workers and ensures proper task synchronization and data integrity.
6. Sends shutdown messages to all workers when the run is complete.

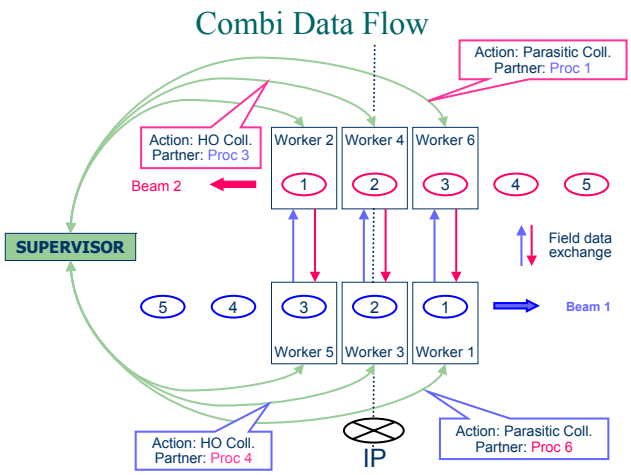

Figure 1: Parallel architecture and communications paths

Worker

Each worker process is associated with a particular bunch and stores and calculates all data (coordinates and statistics) for that bunch locally. Depending on available resources and the number of parasitic collisions to be modelled at each IP, in the implementation each worker process may actually be responsible for several bunches, but for clarity of exposition we retain the processor-bunch duality. Workers perform *self-actions* (e.g. transport to the next interaction point) or *pair-actions* (e.g. a head-on or parasitic collision with another bunch).

After initialization, each worker goes into "listen" mode where it waits for an instruction from the supervisor, acts on it, and sends a completion message back to the supervisor. This continues until a shutdown message is received, upon which the process exits. Each instruction message contains the relevant information to perform the task, such as betatron phase advances, and identity of partner bunches. The actions can be itemized as follows:

1. Initialization. Generate initial particle coordinates sampled from the bunch distribution, using the random seed sent by the supervisor. On completion return the random seed to the supervisor.
2. Transport. Advance the particles according to the betatron phases sent by the supervisor. Then send a completion message to the supervisor.
3. Collision. The type of collision (head-on, parasitic) and identity (process id) of the partner (opposing bunch) are sent by the supervisor. Compute the electric field of the local bunch and send the field data to the partner. Receive a message from the partner containing its field data, and perform the resulting kicks on the local bunch coordinates. Send a completion message to the supervisor.
4. Other actions. Additional single- or dual-bunch actions or variants identified by action code and partner bunch number.
5. Output. Workers are responsible for all output involving the local bunch data, and open independent output streams for these. Output can include bunch statistics, centroid coordinates for frequency analysis, and particle coordinates for tracking analysis.
6. Shutdown. Worker process exits.

PERFORMANCE

We performance-tested the parallel Combi code on a commodity cluster of Opteron 250 (2.4GHz, 1024KB cache) processors in a dual-cpu MYRINET configuration, employing the Portland Group compilers together with the MYRINET GM implementation of MPI. We also performed some preliminary timings on an IBM Blue Gene supercomputer (8192 CPUs and 2TB memory). We include the latter architecture since it provides a reference point for possible very-large-scale simulations in which the number of simulated bunches may approach the number of real bunches circulating in the LHC.

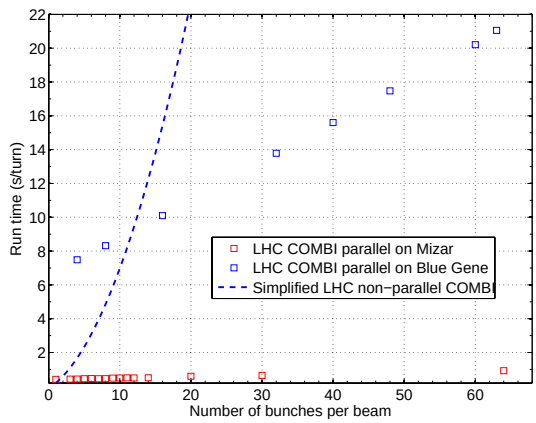

Figure 2: Timings of original (single processor) and parallel codes with varying numbers of bunches

As seen in Figure 2 the parallel code performs effectively in the MPI setting and scales to large numbers of bunches on the commodity cluster with a modest, and linear, communications overhead, reflecting the predominantly one-to-one messaging scheme, with no all-to-all messages required and minimal one-to-all and all-to-one messaging for task assignments and synchronisation. The timing trend for the non-parallel code is as expected and indicates that by $\sim$8 bunches the parallel speed-up is approximately equal to the number of processors.

The initial results for Blue Gene reflect its lower cpu clock speed and also suggest a non-optimal communications overhead, so further testing and development will be required to establish its applicability to large-scale problems.

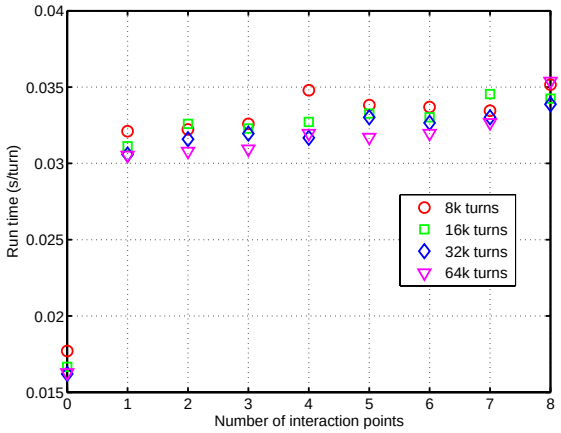

Figure 3: Timings of parallel code with varying numbers of interaction points

On the commodity cluster another series of timings were performed for a fixed number of circulating bunches and increasing numbers of IP's, until all bunches were colliding at every step. Figure 3 shows again that increasing numbers of collisions can be accommodated with low overhead and good scaling properties.

EXAMPLE APPLICATION

As an example we simulated an arbitrary configuration for the LHC where the two colliding beams are made of 24 and 29 bunches, respectively. The collision pattern consists of 4 head-on (HO) collisions (IP 1, 2, 5 and 8 of the LHC layout) and 5 parasitic interactions left and right of IP 1 and 5. The non-regular beam filling scheme together with the non-symmetric collision pattern lead to several different families of bunches. In Figure 4 we compare the horizontal tune spectra of a "nominal" bunch (bottom) and a particular "SuperPacman" bunch (top). The nominal bunch experiences 4 HO and 14 parasitic collisions while the SuperPacman bunch collides HO only in IP 2 and 8. The SuperPacman bunch doesn't show any coherent oscillation except at the machine unperturbed tune ($Q_x \approx 0.31$) while the nominal bunch (bottom), due to the 4 HO interactions, will experience oscillations also at $Q_x \approx 0.298$. Only a code that allows a multi-bunch beam structure and multiple interaction points can reproduce coherent beam-beam effects for all different bunch categories. For this example, the parallel code running time on 54 clustered 2.4GHz CPUs is 0.411 s/turn, which for a minimum required number of turns of 2^{14} means less than 2 hours of computation time. By comparison, the non-parallel code has a running time of 26.2 s/turn and would have required 5 days on the same hardware.

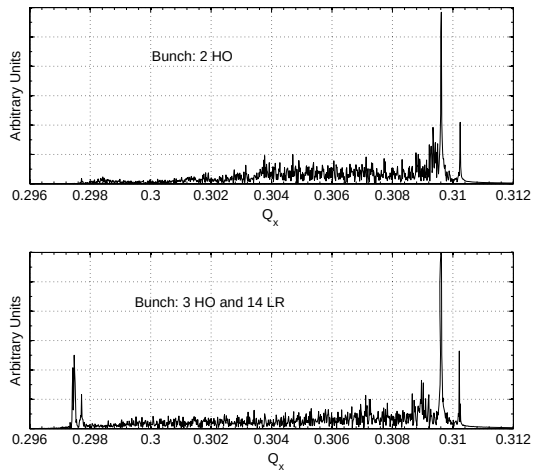

Figure 4: Horizontal tune spectra for a nominal bunch (4 HO and 14 parasitic interactions, bottom) and a SuperPacman bunch (2 HO interactions, top).

CONCLUSIONS

The parallel code COMBI has expanded the scope of multi-bunch and multi-IP beam-beam simulations and can be used effectively on commodity clusters. This enables new studies of coherent beam-beam effects in the LHC, ranging over the variety of collision patterns that will be present, and in the RHIC collider exploring experimental evidence of bunch to bunch differences[3].

The structure of the code allows further refinements such as the recent additions of chromaticity and measurement devices (transverse beam transfer functions[3]) and the inclusion of a self-consistent field-solve/kick model which is the next planned development.

ACKNOWLEDGEMENTS

The authors would like to thank colleagues at the EPFL HPC Lausanne for their support of this project and for invaluable access to advanced hardware and software, in particular Drs. J. Menu and J.C. Leballeur for the MIZAR cluster and C. Clemencon for the BlueGene/L Supercomputer facilities.

REFERENCES

[1] T. Pieloni and W. Herr, Coherent beam-beam modes in the CERN Large Hadron Collider (LHC) for multiple bunches, different schemes and machine symmetries, Particle Accelerator Conference 2005, Knoxville, USA.

[2] T. Pieloni and W. Herr, Models to Study Multi Bunch Coupling Through Head-On and Long-Range Beam-Beam Interactions, EPAC 2006, Edinburgh.

[3] W. Fischer et al., Transverse Beam Transfer Functions of Colliding Beams in RHIC, *these proceedings.*

