Comput Sci Res Dev (2013) 28:185–192
DOI 10.1007/s00450-012-0216-2

SPECIAL ISSUE PAPER

A fast and scalable low dimensional solver for charged particle dynamics in large particle accelerators

Yves Ineichen · Andreas Adelmann · Costas Bekas · Alessandro Curioni · Peter Arbenz

Published online: 23 May 2012

Abstract Particle accelerators are invaluable tools for research in the basic and applied sciences, in fields such as materials science, chemistry, the biosciences, particle physics, nuclear physics and medicine. The design, commissioning, and operation of accelerator facilities is a non-trivial task, due to the large number of control parameters and the complex interplay of several conflicting design goals.

We propose to tackle this problem by means of multi-objective optimization algorithms which also facilitate massively parallel deployment. In order to compute solutions in a meaningful time frame, that can even admit online optimization, we require a fast and scalable software framework. In this paper, we focus on the key and most heavily used component of the optimization framework, the forward solver. We demonstrate that our parallel methods achieve a strong and weak scalability improvement of at least two orders of magnitude in today's actual particle beam configurations, reducing total time to solution by a substantial factor.

Our target platform is the Blue Gene/P (Blue Gene/P is a trademark of the International Business Machines Corporation in the United States, other countries, or both) supercomputer. The space-charge model used in the forward solver relies significantly on collective communication. Thus, the dedicated TREE network of the platform serves as an ideal vehicle for our purposes. We demonstrate excellent strong and weak scalability of our software which allows us to perform thousands of forward solves in a matter of minutes, thus already allowing close to online optimization capability.

Keywords Beam dynamics simulation · Scalability · Space charge · Multi-objective optimization · BG/P

1 Introduction

In contemporary scientific research, particle accelerators play a significant role. Fields, such as material science, chemistry, the biosciences, particle physics, nuclear physics and medicine rely on reliable and effective particle accelerators as research tools. Achieving the required performance is a complex and multifaceted problem in the design, commissioning, and operation of accelerator facilities. Today, tuning machine parameters, e.g., bunch charge, emission time and various parameters of beamline elements, is most commonly done manually by running simulation codes to scan the parameter space. This approach is tedious, time consuming and can be error prone. In order to be able to reliably identify optimal configurations of accelerators we propose to solve large multi-objective design optimization problems to automate the investigation for an optimal set of tuning

Y. Ineichen (✉)
IBM Research—Zurich & Paul Scherrer Institut, Rüschlikon, Switzerland
e-mail: yves.ineichen@psi.ch

A. Adelmann
Paul Scherrer Institut, Villigen, Switzerland
e-mail: andreas.adelmann@psi.ch

C. Bekas · A. Curioni
IBM Research—Zurich, Rüschlikon, Switzerland

C. Bekas
e-mail: bek@zurich.ibm.com

A. Curioni
e-mail: cur@zurich.ibm.com

P. Arbenz
Computer Science Department, ETH, Zürich, Switzerland
e-mail: arbenz@inf.ethz.ch

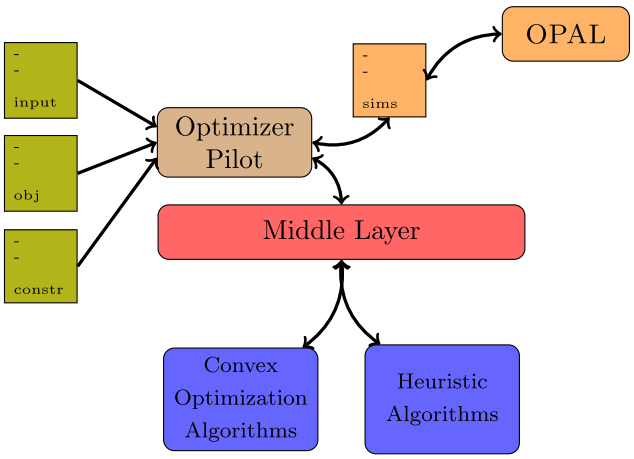


Fig. 1 Multi-objective framework: the pilot (master) solves the optimization problem specified in the input file by coordinating optimizer algorithm and workers running forward solves

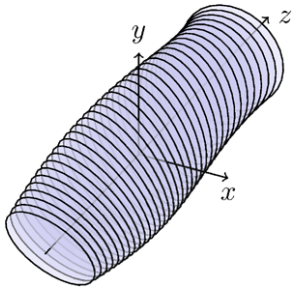
parameters. Observe that multiple and conflicting optimality criteria call for a multi-objective approach.

We developed a modular multi-objective software framework (see Fig. 1) where the core functionality is decoupled from the "forward solver" and optimizer (master/slave). This allows to easily interchange optimizer algorithms, forward solvers and optimization problems. A "pilot" coordinates all efforts between the optimization algorithm and the forward solver. This forms a robust and general framework for massively parallel multi-objective optimization. Currently the framework offers one concrete optimization algorithm, a genetic algorithm employing a NSGAII [1] selector. Normally this methods are plagued by the trade-of between level of detail and time to solution. We address this problem by using forward solvers with different time and detail complexity.

In our application the forward solver provides information about the beam behavior for a given input configuration. The OPAL (Object Oriented Parallel Accelerator Library) framework [4] already features a parallel three dimensional macro particle tracker for beam dynamic simulations integrating millions of macro particles in time. During a simulation run the tracker calculates and applies forces acting on particles, e.g. space charge (electric interactions of charged particles) and external electric and magnetic fields induced by beam line elements such as solenoids and radio frequency cavities. In particular computing space charge forces is computationally intensive. Commonly (and also in OPAL) the arising N-Body problem is solved in every timestep using an FFT-based or iterative [3] solver. This, and additional computationally demanding components of a full 3D tracker, put a severe limitation on the time frame within which we can perform optimization of particle accelerators.

In order to facilitate a poly-algorithmic approach (favored by the optimizer), trading the level of detail for time

Fig. 2 Sketch of a sliced particle bunch



to solution depending on the situation, we incorporated a simplified model (similar to [5]) into OPAL (named "envelope tracker") providing a fast forward solver with reduced details while retaining important characteristics for the multi-objective optimizer. This can be achieved by replacing macro particles with slices (number of slices $\ll$ number of particles), as depicted in Fig. 2. Slices describe the beam envelope, i.e. extension of the beam in transversal direction. As with macro particles, slices are subject to forces and the ellipses deform accordingly. Since the Envelop Tracker employs a number of slices that is orders of magnitude smaller than the number of macro particles used in the 3D tracker (commonly less then 1000), execution time reduces drastically while beam characteristics are retained. For example, the space charge calculation N-Body problem becomes very small and can cheaply be solved by an $\mathcal{O}(\text{slices}^2)$ approach or even estimated cheaply in closed form. In this paper we solely focus on improvements towards a fast and scalable forward solver, in the context of the new fast envelope tracker:

- In Sect. 2 we discuss implementation and parallelization of a fast envelope tracker for beam dynamic simulations. In order to understand performance we provide an overview of the theoretical complexity of all involved components.
- In Sect. 3 we test the scalability of our methods. The compute platform of choice was the Blue Gene/ P Supercomputer. We achieve a very good parallel performance by taking advantage of fast MPI collectives on the Blue Gene/P TREE network.
- The incorporation and parallelization of the new envelope tracker within OPAL was far from being simple. Significant code re-engineering was crucial in achieving our purposes.

We report that in current beam dynamics simulations our parallelization of the envelope model reduces the time to solution by two orders of magnitude (from 3D tracker to $O(n^2)$ space charge slice tracker), and even more with the analytical space charge model. This opens the way for large scale multi-objective optimization design of particle accelerators.

Table 1 Description of employed variables

Variable	Description
i	slice index
c	speed of light
z_i	longitudinal position of slize i
R_i	radius of slice i
β_i	speed of slice i relative to speed of light
γ_i	Lorentz factor of slices i
Q	the bunch charge
L	the length of the bunch
E_z^{ext}	total external electric field
E_z^{sc}	total space charge field
K	sum of focusing gradient of all active beamline elements

2 The envelope tracker

The envelope tracker solves an ordinary differential equation describing the equations of motion for homogeneously charged slices. Slices are generally ellipses and in special cases circles (see Fig. 2) (see [6] for the complete mathematical framework). Important variables are summarized in Table 1.

The main equation describing the time evolution of each slice radius R_i (similar for axes by solving the equation twice for both axes independently) is

$$\begin{aligned}&\frac{d^2}{d^2t}R_i + \beta_i\gamma_i^2\frac{d}{dt}(\beta_i R_i) + R_i\sum_j K_i^j \\ &= \frac{2c^2k_p}{R_i\beta_i}\times\left(\frac{G(\Delta_i, A_r)}{\gamma_i^3} - \left(1-\beta_i^2\right)\frac{G(\delta_i, A_r)}{\gamma_i}\right) \\ &\quad + \frac{4\varepsilon_n c}{\gamma_i}\frac{1}{R_i^3},\end{aligned} \tag{1}$$

where ε_n is the computed root mean square normalized emittance, k_p the beam perveance, $G(\Delta_i, A_r)$ the radial space charge term and $\Delta_i = z_i - z_{\text{tail}}$ is the distance from slice i to the tail of the bunch, $\delta_i = z_i + z_{\text{head}}$. $A_{r,i}$ denotes the slice rest frame aspect ratio $R_i/(\gamma_i L)$. The evolution of longitudinal motion for each slice is

$$\begin{aligned}\frac{d}{dt}\beta_i &= \frac{e_0}{m_0 c\gamma_i^3}\left(E_z^{\text{ext}}(z_i,t) + E_z^{\text{sc}}(z_i,t)\right) \\ \frac{d}{dt}z_i &= c\beta_i.\end{aligned} \tag{2}$$

Algorithm 1 states all necessary steps in order to solve these equations. The first three methods (invoked on line 2,4 and 5) handle the sampling of the time and position dependant external electric and magnetic fields present in a particle accelerator, e.g., cavities and magnets. While `switchElement()` ensures that, at the current position

Algorithm 1 Core code of the Envelope Tracker

```
1:  for all timesteps do
2:      switchElements()
3:      for all slices do
4:          getExternalFields()
5:          getKFactors()
6:      end for
7:      synchronizeSlices()
8:      calcCurrent()
9:      calcSpaceCharge()
10:     if not all slices emitted then
11:         emission()
12:     end if
13:     for all emitted slices do
14:         timeIntegration()
15:     end for
16:     t ← t + Δt
17: end for
```

of the bunch, the correct elements are active, the other two calculate the external electric and magnetic field and slice deformation forces for each slice. Next we calculate self induced fields (see next two sections). During the emission process all slices are emitted at the cathode. Finally, in every timestep, we integrate the already emitted slices by a 5th order Runge-Kutta integration scheme with monitoring of local truncation errors (as presented in [8, pp. 714ff]) is used to solve (1) and (2). The ODE can be solved for each slice independently and, therefore, in parallel.

A comparison shows that important quantities are within a 5 % margin compared to the 3D tracker. In the context of our multi-objective optimization application this level of detail suffices.

N^2 space charge computation In order to compute space-charge and current profile efficiently in parallel we collect β (speed of a slice relative to the speed of light) and z position of all n slices a priori on all processors ("synchronization" of slice information). This synchronization alone requires two `MPI_Allreduce` over an array of size n.

The actual space charge and current profile calculation are the most computationally involved in Algorithm 2. The N-Body problem requires $\mathcal{O}(n^2)$ operations and one global reduction on single floating point numbers. The current profile calculation is more expensive due to a Savitzky-Golay smoothing filter (implemented as in [8]), solving a linear system of equations by performing an LU decomposition as well as two convolutions. Here, n_s denotes the number of points of the current density that need to be smoothed. Typically n_s is only a small fraction of n.

Analytical space charge computation In order to reduce the space charge computation costs we calculate an analyt-

Algorithm 2 $\mathcal{O}(n^2)$ space charge computation

```
for all i ∈ slices do
  for all j ∈ slices do
    d_z ← |z_j − z_i| {distance from slice j to i}
    if d_z > minimal slice distance we consider then
      v ← calculated influence of slice j {scaled by 1/√(d_z^2)}
      if z_j > z_i {check if j is left or right of i} then
        sm ← sm − v
      else
        sm ← sm + v
      end if
    end if
  end for
  F_{l,i} ← longitudinal space charge force depending on sm
  F_{t,i} ← transversal space charge force can be computed independently
end for
```

ical approximation of space charge forces. This is achieved by introducing a factor z/L denoting the fraction of slices to the right of the slice under consideration. Assuming a cylindrical beam shape the longitudinal space charge term becomes

$$E(z) = \frac{Q}{2\pi\epsilon_0 R^2}\left[\sqrt{\left(1-\frac{z}{L}\right)^2+\left(\frac{R}{L}\right)^2} - \sqrt{\left(\frac{z}{L}\right)^2+\left(\frac{R}{L}\right)^2} - \left|1-\frac{z}{L}\right| + \left|\frac{z}{L}\right|\right].$$

The radial space charge term can be deduced similarly. Since the analytic formulation does solely depend on the bunch length an the z position of the slice under consideration, parallelization is trivial: the bunch length ($z_{\text{head}} - z_{\text{tail}}$) has to be computed once by finding the minimal and maximal slice z position using an `MPI_Allreduce`. Note that computing the current profile (`calcI()`) is not required under the analytic space charge model.

2.1 Theoretical complexity

To understand the performance of the code we first cover a detailed analysis of the complexity of all methods introduced in Algorithm 1. In the following analysis we will denote the total number of slices by n, the total number of timesteps by t, with m the total number of beamline elements and s is the degrees of freedom of a slice. With help of these abbreviations we deduce worst case bounds for the number of floating point operations (FLOPs), and summarized in Table 2.

Table 2 Complexity summary and number of `MPI_Allreduce` (NM) for various phases of envelope tracker (* denotes analytical space charge computation)

Phase	FLOPs	NM	Size
switchElements()	$\mathcal{O}(m)$	–	
getExternalFields()	$\mathcal{O}(m)$	–	
getKFactors()	$\mathcal{O}(m)$	–	
synchronizeSlices()	$\mathcal{O}(1)$	2	n
calcCurrent()	$\mathcal{O}(n^3)$	1	n
calcSpaceCharge()	$\mathcal{O}(n^2)$	2	1
calcSpaceCharge()*	$\mathcal{O}(1)$	2	1
emission()	$\mathcal{O}(n)$	1	1
timeIntegration()	$\mathcal{O}(ns^2)$	–	

Starting from the top, we note that external fields methods do not require communication and operations are proportional to the number of currently active beamline elements. Subsequently, space charge forces have to be computed by one of the two described methods. Emission requires one collective reduction over one double and is done n times in total (independent of t). Finally, the Runge-Kutta integrator is called, requiring no communication and only $\mathcal{O}(ns^2)$ FLOPs (here $s = 10$).

Table 2 illustrates that the n^2 space charge model is computationally the most expensive part. The other phases are comparably cheap and do not require additional communication. Therefore, we expect them to scale perfectly.

2.2 Parallelization

In this section we describe the parallelization of all relevant sections of the code, without considering the analytic space charge computation (embarrassingly parallel).

We distribute all slices in contiguous blocks on all available processors, creating a distribution with a load imbalance of at most 1 slice. The number of synchronization points is small, i.e. one per timestep in `synchronizeSlices()`. Once this synchronization takes place we only depend on a small number of collectives for single variables. The last communication related expensive part is calculating beam statistics. For the moment we can neglect this because, from an optimization point of view, we only need to calculate beam statistics a constant number of times during a simulation ($\ll t$).

The envelope tracker is parallelized by employing MPI collectives, such as e.g. `MPI_Allreduce`. On the other hand OPAL relies on another framework, the Independent Parallel Particle Layer (IPPL) [2], providing an abstract layer for handling parallel fields and particles. The envelope tracker implementation only uses OPAL's features to incorporate external electric and magnetic fields and the infrastructure, e.g. to handle input files. Since these features do not

require much communication, reported performance results generally do not benchmark IPPL, but only our pure MPI implementation of the envelope tracker. However, when examining the total MPI time it is important to take into consideration that using OPAL's features employ the IPPL message class (MPI layer using pre-posted MPI_Isend's).

During the development of the code we first encountered quite limited scalability. An extensive profiling and benchmarking process revealed that some parts of the original code were responsible as they were purely serial (Amdahl's law). We used the IBM HPCTOOLKIT (see below) for profiling. Thus, while other parts scaled perfectly the serial part of the code became dominant and parallel performance dropped fast. This led to a large code review resulting in more optimizations and parallelization of serial parts. Currently the Savitzky-Golay smoother is the last part of the code that remains serial. Since it smooths only a very small fraction of the total number of slices ($n_s \ll n$), it currently only negligibly affects overall scalability, but is likewise affected by Amdahl's law.

3 Results

Results presented in this section were measured on an IBM Blue Gene/P system. One node consists of a Quad core 450 PowerPC running at 850 MHz and a peak performance of 13.6 GFLOP/s (per node). The machine has 5 networks, two of which are of particular interest for our application: a 3D Torus network and a (tree) network for collectives. The Torus network has a bandwidth of 6 GB/s. The collective network has a bandwidth of 2 GB/s and a round-trip worst case latency of 2.5 µs. More details are given in [9].

3.1 Experimental setting

The Blue Gene/P system offers 3 modes of operation. In VN mode (default for all our experiments), each of the 4 cores per compute node spawns an MPI process. In the DUAL mode we have 2 MPI processes and each of them can spawn 2 shared memory threads. Finally, the SMP mode features 1 MPI process per node which can have 4 threads.

We measured timings of different components with help of IPPL timers (providing a simple wrapper for MPI_Wtime for minimum, average and maximum) and the HPCTOOLKIT [7] for MPI analysis and in-depth details regarding spent cycles. Total time measures the total run time from start to end. In addition, we report timings of space charge and current density calculation (lines 8 and 9 in Algorithm 1), the time integration (line 14 in Algorithm 1) and external field evaluation (line 4 in Algorithm 1). The simulation performs the first 2000 timesteps of the actual SWISSFEL Injector, Phase 1.

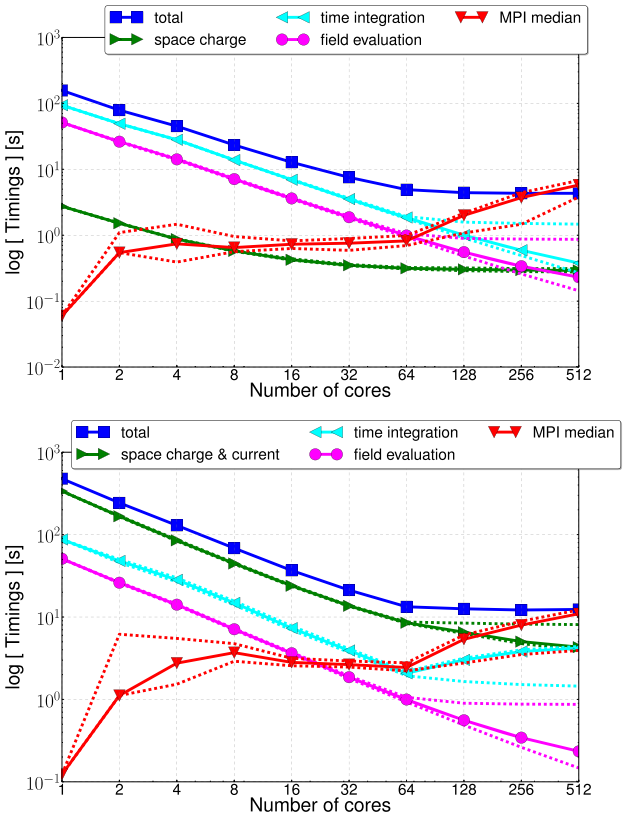


Fig. 3 Timings for simulation with 1,000 slices with analytical (*Top*) and n^2 (*Bottom*) space charge model, minimum and maximum *dashed*, average *solid line*

3.2 Strong scaling

In Fig. 3 minimum, maximum and average timings for 1,000 slices are plotted. We see that minimum and maximum timings are narrow, showing that the problem is well balanced. As expected space charge and current profile dominate parallel performance. Timings of the ODE solver and external field evaluation are one magnitude smaller and at some point scale below of the total communication time (after 128 cores).

As of two cores the MPI timings are constant, independent of the total number of cores employed. Notice that, even with this rather small problem size, the achieved bandwidth is quite satisfactory, although still far form the peak. In particular, for 128 cores on core 0 we get

$$\frac{N_{\text{calls}} \times \text{bytes} \times 8}{t \times 2^{20}} = \frac{7999 \times 7988.0\,\text{b} \times 8}{1.583\,\text{s} \times 2^{20}} \approx 308\,\text{MB/s},$$

for all MPI_Allreduce with size n. Collectives of single values (4 and 8 bytes) are latency dominated. One indication is the total time of 6007×4 byte messages and 16006×8 byte messages is roughly the same (0.245 and 0.333). Fur-

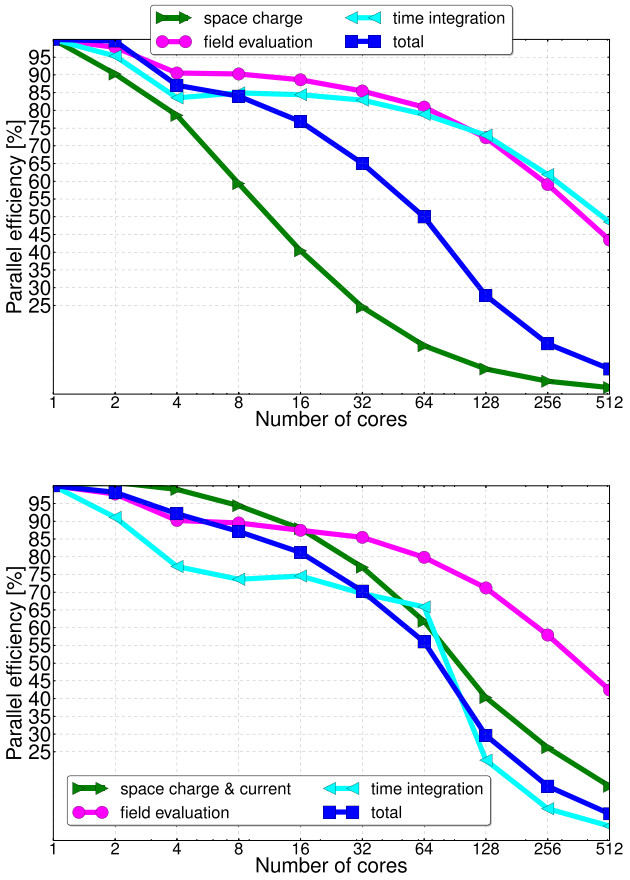


Fig. 4 Parallel efficiency for average timings of 1,000 slices for analytical (*Top*) and n^2 (*Bottom*) space charge

thermore, considering the "ideal" latency

$$\frac{t}{N_{\text{calls}}} = \frac{0.245\ \text{s}}{6007} \approx 41\ \mu\text{s},$$

we are only latency bound for small collectives.

On the Blue Gene/P it is possible to switch between different communication strategies for the collectives. This can be achieved by changing environment variables that control the DCMF layer (see [9]), i.e. `DCMF_*`. We experimented with two different values: `TREE` and `GLOBAL` for the all reduce operation. `TREE` forces the `MPI_Allreduce` to use the collective network whereas `GLOBAL` uses the global collective network protocol (see [9] for more details). The default is using the tree and direct put protocol. In both cases we did not see any improvement. In fact for `TREE` communication time even increases slightly.

The parallel efficiency for measured average timings is shown in Fig. 4. When increasing the number of cores from 1 to 4 field evaluation and ODE solver suffer from cache effects. Independently of where you cache utilization is on 1 core, by increasing the number of cores, the amount of data per core shrinks and therefore cache performance drops.

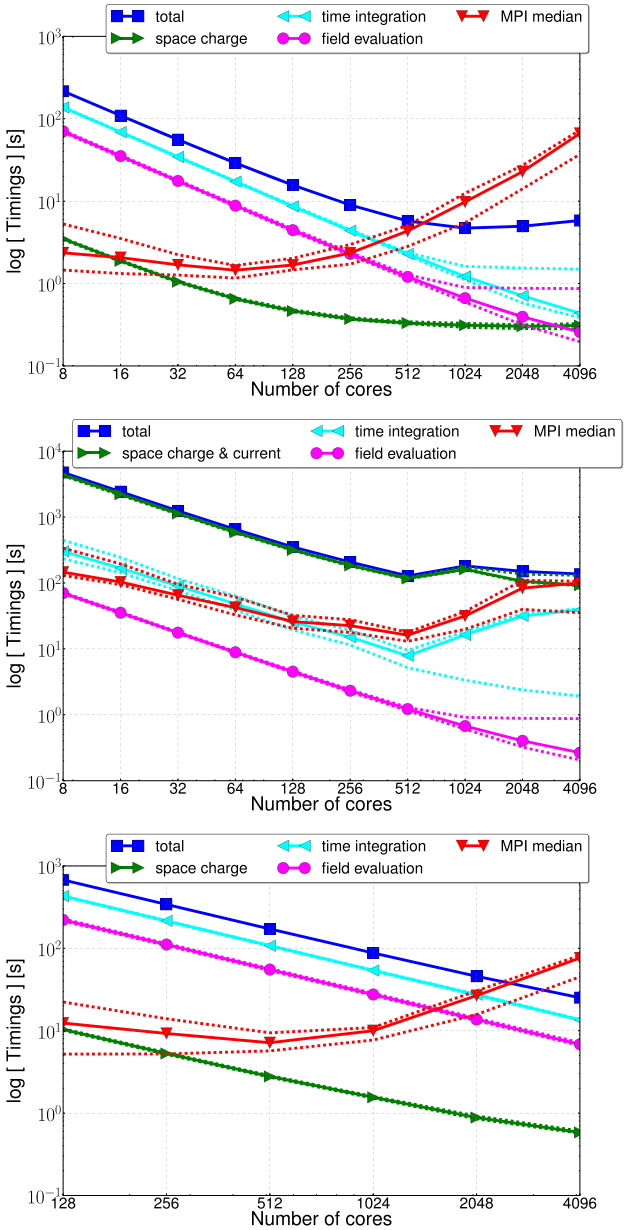


Fig. 5 Average timings (solid line) for analytical (*Top*), n^2 space charge model (*Middle*) with 10,000 slices and 500,000 (*Bottom*) slices with analytical space charge model. (Minimum and maximum *dashed line*)

As we already mentioned, current simulations use up to 1000 slices, which clearly limits overall scalability to at most 1000 MPI processes. However, in the future (inclusion of coherent synchrotron radiation and other physical phenomena) we can require a significantly higher resolution and therefore more slices. In view of these facts and to demonstrate the parallel performance potential of the code with regard to larger number of cores we provide the average timings for a larger problem (10,000 slices) in Fig. 5. As expected we see a much better parallel efficiency (≈85 % at

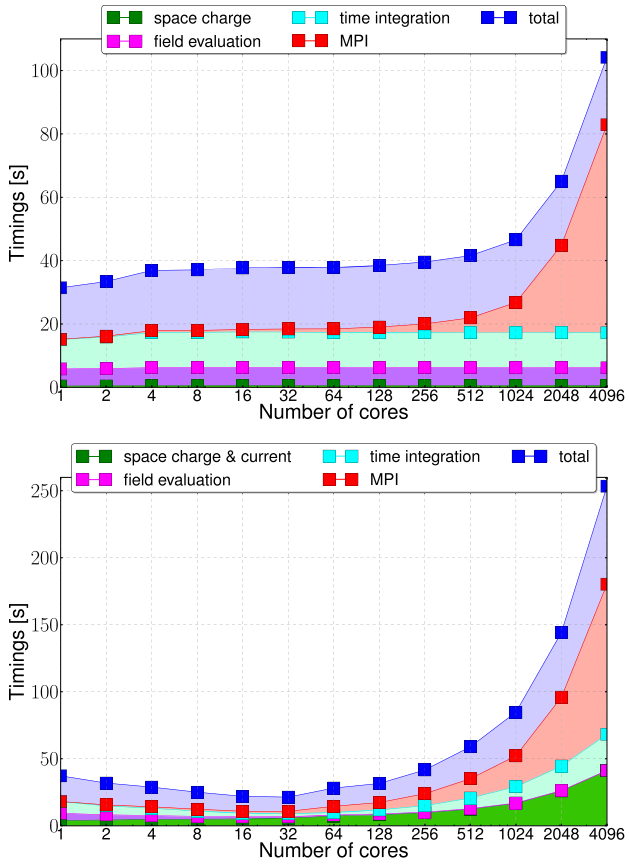


Fig. 6 Weak scaling (average timings) of analytical (*Top*) and n^2 (*Bottom*) space charge model

128 relative to 8 cores). The reason for this can be found by inspecting the saturation of the bandwidth. For 10,000 slices on core 0 out of 128 we get

$$\frac{7999 \times 80000.0\,\text{b} \times 8}{7.385\,\text{s} \times 2^{20}} \approx 661\,\text{MB/s}.$$

We also note that the timings for ODE solver, field evaluation as well as MPI timings are almost two magnitudes smaller than space charge and current density calculation for the n^2 space charge model.

3.3 Weak scaling

Weak scaling is shown in Fig. 6. Since the overall complexity of the n^2 space charge model is quadratic, we fix the work per core to

$$\lceil \sqrt{n_c} \times 100 \rceil,$$

and for the analytical space charge model respectively to

$$n_c \times 100,$$

slices for n_c cores and starting with 100 slices on 1 core. For n^2 space charge we note a drop in execution time for all parts except for space charge and current density calculation, where we notice a slightly increasing tendency caused by the serial smoother. An increasing number of latency bound small all reduce collectives imply a growing trend of total MPI time with increasing number of cores. Again, field evaluation and ODE solver performance is very good. The steep increase in MPI time towards large number of cores is caused by IPPL posting lots of `MPI_Iprobe`'s (16.8s on 2048 cores).

The analytical space charge calculation has almost ideal weak scaling, only the MPI time shows an increasing tendency towards the end. As seen with the n^2 space charge model, this is caused by IPPL (here 12.3 s on 2048 cores and 20.3 s on 4096 cores).

4 Conclusion

We presented a scalable parallel fast solver for beam dynamic simulations. The envelope tracker is fully integrated into OPAL and can be used for production mode beam dynamic simulations. OPAL is a large and complex software suite. Thus, integrating and parallelizing the envelope tracker required extensive algorithmic adaptation and code re-engineering. Both, the envelope tracker and the discussed scalability results, are a crucial cornerstone in our massively parallel multi-objective optimization framework.

We were able to achieve satisfactory parallel efficiency for even the small number of slices of current simulations. Indeed, we achieved an almost 2 orders of magnitude reduction of runtime for relevant cases in the setting of our multi-objective optimization problem. We demonstrated that future simulations will immediately benefit from our code. In particular, in the context of multi-objective optimization, we are able to tune the run-time parameters of the forward solver in such a way that we will be able to maximize overall performance and scalability.

Even though we notice a loss in parallel efficiency for small problems, we demonstrated that the code is well parallelized (e.g. for larger problems). Bandwidth saturation enforces a hard limit on performance with respect to problem size. Fortunately, the optimizer requires thousands of forward solves when solving a multi-objective optimization problem. This enables us to determine how many cores are necessary to maximal saturate the bandwidth or to achieve an acceptable parallel efficiency for one forward solve and then run the maximal number of parallel forward solves as possible.

The nature of the introduced forward solver invites additional parallelization methods, such as e.g. using extended multithreading, that will be included and benchmarked in the future.

References

1. (2003) PISA—a platform and programming language independent interface for search algorithms. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L (eds) Evolutionary multi-criterion optimization (EMO 2003). Lecture notes in computer science. Springer, Berlin, pp 494–508
2. Adelmann A The IP^{2}L (independent parallel particle layer) framework. Technical Report PSI-PR-09-05, Paul Scherrer Institut, 2009–2010. http://amas.web.psi.ch/docs/ippl-doc/ippl_user_guide.pdf
3. Adelmann A, Arbenz P, Ineichen Y (2010) A fast parallel Poisson solver on irregular domains applied to beam dynamics simulations. J Comput Phys 229(12):4554–4566
4. Adelmann A, Kraus C, Ineichen Y, Yang JJ The OPAL (Object Oriented Parallel Accelerator Library) framework. Technical Report PSI-PR-08-02, Paul Scherrer Institut, 2008–2010. http://amas.web.psi.ch/docs/opal/opal_user_guide-1.1.6.pdf
5. Ferrario M (2006) Homdyn user guide. Technical report, LNF. http://nicadd.niu.edu/fnpl/homdyn/manual.pdf
6. Ferrario M, Boscolo M, Fusco V, Vaccarezza C, Ronsivalle C, Rosenzweig JB, Serafini L (2003) Recent advances and novel ideas for high brightness electron beam production based on photo-injectors. In: Rosenzweig J, Travish G, Serafini L (eds) The physics and applications of high brightness electron beams, pp 45–74
7. Lakner G, I-Hsin C, Guojing C, Fadden S, Goracke N, Klepacki D, Lien J, Pospiech C, Seelam SR, Wen H-F (2009) IBM System Blue Gene solution: performance analysis tool. http://www.redbooks.ibm.com/abstracts/redp4256.html
8. Press W, Teukolsky S, Vetterling W, Flannery B (1992) Numerical recipes in C, 2nd edn. Cambridge University Press, Cambridge
9. Sosa C, Knudson B (2010) IBM System Blue Gene solution: Blue Gene/P application development. http://www.redbooks.ibm.com/abstracts/sg247287.html?Open