A PARALLEL EXTENSION OF THE UAL ENVIRONMENT

N. Malitsky, A. Shishlo, BNL, Brookhaven, US

Abstract

The deployment of the Unified Accelerator Library (UAL) environment on the parallel cluster is presented. The approach is based on the Message-Passing Interface (MPI) library and the Perl adapter that allows one to control and mix together the existing conventional UAL components with the new MPI-based parallel extensions. In the paper, we provide timing results and describe the application of the new environment to the SNS Ring complex beam dynamics studies, particularly, simulations of several physical effects, such as space charge, field errors, fringe fields, and others.

1 MOTIVATION

The high level of beam loss in future high intensity proton accelerator complexes introduces a serious challenge for accelerator physicists. To make some realistic predictions at 10-4 level, one has to deal with a combination of different physical effects and dynamic processes. These effects can be evaluated and corrected independently with a set of general-purpose accelerator codes and specialized programs. However, the accurate modelling of necessary low-level beam losses requires the simultaneous consideration of several effects in a single scenario. The open architecture of the Unified Accelerator Libraries [1] addresses this task. In the UAL, accelerator algorithms are implemented as replaceable modules exchanging data via Common Accelerator Objects (Element, Bunch, Twiss, etc.). Thus for the Spallation Neutron Source (SNS) Ring study, a UAL-based simulation environment has been developed [2]. This environment includes injections painting, field errors, misalignments, fringe fields, and space charge effect. The space-charge effect has the largest impact on halo growth in the SNS accumulator ring. Then preliminary working points and injection schemes have been determined only by the space charge beam dynamics. For example, the working point with the splitting of tunes by a half-unit (Qx, Qy) = (6.3, 5.8) gave promising results based on numerical simulations. However, adding magnet errors and misalignments in this optimised scenario dramatically increased beam loss to unacceptable level. After detailed study [3], it was confirmed that such resonant behaviour was induced by a combined effect of the space charge and skew-quadrupole errors. The space charge depressed the tunes and made particles trapped into the sum resonance (Qx+Qy=12) driven by skew-quadrupole field. This study demonstrated the importance of integrated effects. Also, it showed performance limitation of the existing simulation environment for complex studies and necessity of the UAL parallel extension.

2 SOLUTION

The original UAL environment has been implemented using the two-language approach (C++ and Perl). The UAL internal infrastructure and all numerical algorithms are implemented as compiled C++ shared libraries. The integration of these libraries is provided via the Perl scripts. This scheme is designed to facilitate selecting and implementing the most appropriate accelerator approaches and supporting task-specific requirements. We have also found the same architecture can be perfectly fitted to the parallel environment based on the Message-Passing Interface (MPI) [4].

2.1 Embarrassingly Parallel Computations

Many accelerator simulation models can be represented by the combination of single-particle and multi-particle approaches. The single-particle beam dynamics suggests no communication between the separate processes and is classified as embarrassingly parallel. For this category of UAL applications, the MPI executables can be represented by the original sequential Perl scripts with few additional lines for initialisation and finalizing of the MPI processes (see Table 1).

Table 1: A set of Perl commands for the embarrassingly parallel simulation scenario.

Perl Command
1. Initialize the MPI process.
2. Read the MAD file with the design lattice description.
3. Add aperture, static and random field errors and misalignments for selected elements.
4. Select and specify the injection scenario.
5. Select and specify the tracking integrators.
6. Select and specify the required diagnostics.
7. Define the unique bunch of particles for each CPU based on the MPI process id.
8. Inject and track particles.
9. Finalize the MPI process.

The Perl interface to the two MPI C functions (1 and 9 commands, Table 1) is implemented as a Perl Short_MPI module based on the standard Perl XS mechanism used for embedding all UAL C++ classes. This approach not only preserves the sequential Perl scripts but also does not require recompilation of existing C++ shared libraries.

2.2 Communication

Development of collective space-charge effects and time consuming numerical algorithms (e.g. calculation of high-order Taylor maps) on parallel computers requires the integration of the additional communication mechanism in serial procedures. In the UAL environment, the parallel version of the original algorithm is implemented as a new C++ library that can be mixed together with other UAL sequential and/or parallel components. It enables one to encapsulate all communication procedures in a single place and reuse the existing simulation environment.

In the UAL, the space charge effect is currently developed through the ORBIT module [5]. ORBIT implements the 2.5D approach in which the space charge effect is modelled by the thin 2D transverse kick multiplied by the factor proportional to the longitudinal bunch density. The calculation of the transverse kick is based on the Particle-In-Cell (PIC) method employing the convolution algorithm for finding 2D forces. The flow logic and the strategy taken to parallelize this calculation are shown in Table 2.

Table 2: Parallel flow logic for the transverse space charge forces calculation.

Stage	**1st CPU**	**Commun.**	**Nth CPU**
1. Start	+		+
2. Set up PIC grid	+		+
3. Sync. PIC grid	+	$\leftarrow\rightarrow$	+
4. Bin macro-particles	+		+
5. Sum charge distribution	+	$\leftarrow\rightarrow$	+
6. Set up Greens function	+		+
7. FFT Greens function (GR)	+		+
8. FFT charge distribution (CH)	+		+
9. Convolute CH&GR FFT	+		+
10. Backward FFT for forces	+		+

According to Table 2, the serial and parallel versions for the transverse space charge calculations are different only in steps 3 and 5. Thus the new C++ class has been derived from the original one with overriding few methods. The data exchange between different CPU's is provided by the special optimised MPI function, MPI_Allreduce. All communications occur on the C++ level keeping the Perl API unchangeable. The user still has access to the Perl constructor of the new class and is able to integrate it into the previous embarrassingly parallel simulation scenario (command 5, Table 1).

3 TIMING RESULTS

The simulation environment was deployed on the Linux workstation cluster including six dual i586 CPUs (512 kBytes L2 cache, 512 MB RAM) and the 100 Mb/s Fast Ethernet switch. The MPICH 1.2.1 library installed under the Red Hat 7.0 Linux operating system provided the communication between different CPU's. For the timing purposes, the typical SNS ring simulation scenario can be described by the following major characteristics:

Table 3: Parameters of the SNS Ring simulation scenario.

Parameter	**Value**
Number of	
Lattice elements	670
Space charge (SC) nodes	670
Injection turns	1000
Macro-particles injected per turn	200
PIC bins	128x128
Integrator of	
Space charge (SC) nodes	ORBIT[5]
Lattice elements	TEAPOT[6]

The number of injected macro-particles is determined by the beam loss level. According to Table 4, 200 K macro-particles have been selected as an optimal quantity that gives the satisfactory accuracy in the 1.0e-3 – 1.0e-4 beam loss range.

Table 4: The loss level estimation for the 200 K macro-particle scenario.

Loss level	**Number of lost particles**	**Error of the loss level estimation**, [%]
η	$\eta \times N_{part}$	$100\% * 1/\sqrt{\eta \times N_{part}}$
1.0e-3	200	7
1.0e-4	20	22

The total time of the one-turn simulation is linearly proportional to the number of propagated particles (see Figure 1)

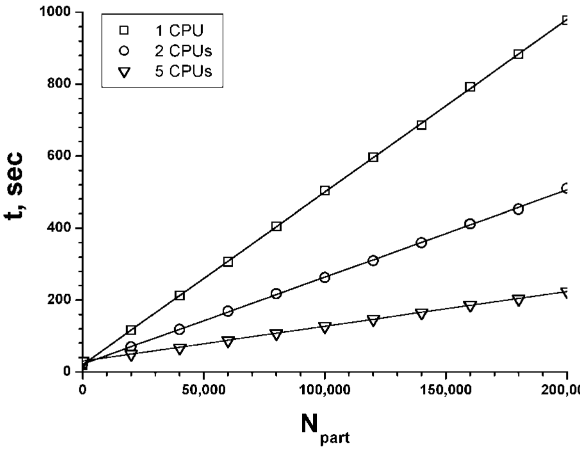

Figure 1. The parallel timing results for the one-turn simulation for the SNS lattice.

In our injection-painting scenario, the number of macro-particles is the sum of an arithmetic progression, therefore, the total time of the serial calculation can be presented as

$$t_{serial} = \tau_\alpha \cdot n_{inj} \cdot N_t^2 + \tau_\beta \cdot N_t \qquad (1)$$

where τ_α and τ_β are parameters of the timing model, and N_t is the number of turns. For the case of parallel simulation the formula (1) can be rewritten to take into account that the injected macro-particles are scattered throughout CPUs

$$t_{||} = \frac{\tau_\alpha n_{inj} \cdot N_t^2}{N_{CPU}} + (\tau_\beta + (N_{CPU} - 1) \cdot \tau_c) N_t \qquad (2)$$

In the formula (2) we suppose that the communication time $(N_{CPU} - 1) \cdot \tau_c$ is proportional to the number of CPUs minus one. The parameters τ_α, τ_β, and τ_c were determined by the timing of the real calculations and are shown in the Table 5.

Table 5. The parameters of the formula (2) for the total time of the calculation.

τ_α, **[sec]**	τ_β,**[sec]**	τ_c,**[sec]**
0.0024	18.7	2.8

The formula (2) predicts the total calculation time with accuracy about 5-10%. In addition, it enables us to determine the optimal number of CPUs

$$N_{CPU}^{opt} = \sqrt{\frac{\tau_\alpha \cdot n_{inj}}{\tau_c} \cdot N_t} \qquad (3)$$

and the parallel efficiency

$$\eta_{||} = \frac{t_{serial}}{t_{||} \cdot N_{CPU}} \qquad (4)$$

For the considered case, the optimal number of CPUs is about 10 that corresponds to the capacity of our Linux cluster. Because of the dual CPU effect we employ 5 CPUs for each task. It gives us the 70% parallel efficiency and the calculation time decreases from 80 to 23 hours.

5 SUMMARY

The parallel extension of the UAL 1.0 environment has been developed using Message-Passing Interface (MPI) and deployed on the SNS Linux cluster. It gives physicists the affordable time for the SNS Ring complex beam dynamics studies.

6 ACKNOWLEDGEMENT

This work could not be done without the professional support of the BNL Information Technology Division team including R. Casella, E. Efstathiadis, E. McFadden, and L. Slatest.
We also thank A. Fedotov, J. Smith, and J. Wei for many useful discussions.

7 REFERENCES

[1] N.Malitsky and R.Talman. Unified Accelerator Libraries, AIP 391 (1996)
[2] N.Malitsky et al. UAL-Based Simulation Environment for Spallation Neutron Source Ring, PAC 99, 1999.
[3] A.Fedotov, N.Malitsky, and J.Wei. Space-Charge Simulations for the Spallation Neutron Source (SNS) Ring Using Unified Accelerator Libraries, BNL/SNS 086, 2001
[4] B.Wilkinson and M.Allen. Parallel Programming. Prentice Hall, 1999.
[5] J.Galambos et al. ORBIT – A Ring Injection Code with Space Charge, PAC 99, 1999.
[6] L.Schchinger and R.Talman. TEAPOT: A Thin-Element Accelerator Program for Optics and Tracking, Particle Accelerators, 22, 35(1987).