# User's Manual for latest `Pelegant`

**Yusong Wang, Michael Borland, Robert Soliday**

**APS Accelerator Systems Division, Advanced Photon Source**

## 1 Introduction

`Pelegant` stands for "parallel elegant", which is a parallelized version of `elegant` [1]. Written in the C programming language with MPICH, the `Pelegant` has been successfully ported to several clusters and supercomputers, such as the "weed" cluster at APS, the Fusion cluster at Argonne National Lab, and the BlueGene supercomputer at Argonne Leadership Computing Facility. Thanks to careful design in parallelization and good architecture of the serial `elegant`, the `Pelegant` achieves very good performance. For example, for a simulation of 100,000 particles in APS including symplectic element-by-element tracking, accelerating cavities, and crab cavities, the simulation time was reduced from 14.3 days to 42 minutes on 512 CPUs of the BlueGene/L (BG/L) supercomputer. The speedup for this particular simulation is 484 with efficiency near 95%.

This document describes how to build `Pelegant`, run the code and optimize the performance. An explicit list of elements which have NOT been parallelized can be found in the appendices. The user can also check if an element has been parallelized from the latest `elegant` manual. The list of commands which have been used in the regression tests is available at the end of the appendices. The user should be familiar with the User's Manual for `elegant` before reading this document.

## 2 `Pelegant` installation

The binaries of `Pelegant` for different platforms have been distributed together with the `elegant` executables. The detail information about the prebuilt `Pelegant` can be found at Prebuilt Pelegant usage for multi-core computers.

For some users, it might be useful to build Pelegant from the source code. We provide the steps in building `Pelegant` on Linux as follows:

1. Install MPICH (1 or 2). This step can be done by either the user or the cluster administrator. Add the location of MPI executables in your PATH.

2. If you are using an x86 processor, set the environment variables HOST_ARCH and EPICS_HOST_ARCH to linux-x86 (or linux-x86_64 for 64 bits). If you are using a ppc processor you can set these variables to linux-ppc.

3. If EPICS/Base is already installed on your computer you can skip to the next step. `Pelegant` is built using the EPICS/Base configure files available from the OAG web site at APS. You will need to unpack this to create epics/base/configure. Go to the epics/base directory and type make.

4. Next you will need to download the EPICS extensions configure files from the OAG web site. This will unpack to create epics/extensions/configure. Go to the epics/extensions/configure directory and type make.

5. Download the latest SDDS source code from the OAG web site. This will unpack to create epics/extensions/src/SDDS. Go to this directory and type make.

6. Go to the epics/extensions/src/SDDS/SDDSlib directory and run the following commands to build parallel SDDS library:

   ```
   make clean
   make MPI=1
   ```

7. Go to the epics/extensions/src/SDDS/pgapack directory to build the parallel genetic algorithm library:

   ```
   make
   ```

8. Download the OAG configure files from the OAG web site. This will unpack to create oag/apps/configure. Go to this directory and type make.

9. Download the `Pelegant` source code from the OAG web site. This will unpack to create oag/apps/src/elegant.

10. Set the path for your installation of MPICH in Makefile.OAG in oag/apps/src/elegant. This is not required if the MPI executables are in your PATH.

11. From oag/apps/src/elegant, run the following command to build `Pelegant`:

    ```
    make Pelegant
    ```

    `Pelegant` should now exist at oag/apps/bin/linux-x86/Pelegant. (To build the serial version, just type "make." This will also build related software.)

    `elegant` and `Pelegant` share the same source code. Makefile will decide which part of code will be compiled according to the binary file (either `Pelegant` or `elegant`) you want to build.

If you have any question about installing the `Pelegant`, please send an email to soliday@aps.anl.gov

# 3 Running a simulation with `Pelegant`

Running a parallel job is just as easy as starting a serial job, while the time spent on a job can be reduced from several days to hours, or even minutes. `Pelegant` has been tested under both MPICH-1 and MPICH-2. We suggest to use MPICH-2 if possible, as it shows better stability in our regression test. In the future, we may also add new features available in MPI-2, such as remote memory operations, to improve the performance of `Pelegant` for some simulations.

### 3.1 Running `Pelegant` with MPI command

An examples directory is available under the elegant directory in the source code. User can run a simulation with given lattice and input files according to the version of the MPI implementation.

For example, we can choose the Pelegant_ringTracking1 example to run the parallel `elegant` on 11 processors (10 working processors) with the following commands: [1]

1) For MPI-1, we can use the following syntax:
   mpirun -np 11 Pelegant manyParticles_p.ele
2) For MPI-2, mpiexec is strongly encouraged to start MPI programs, e.g.:
   mpiexec -np 11 Pelegant manyParticles_p.ele

One can run another simulation with the serial version of `elegant` for comparison at the same time:

   elegant manyParticles_s.ele

The contents of the input files are same, while "_p" and "_s" are corresponding to the input file of `Pelegant` and `elegant` respectively. User should check the `elegant` manual to prepare the input file.

In principle, user can run simulations on any number of processors. We have tested the program with more than 30,000 cores on the Intrepid (Bluegene/p) supercomputer at Argonne National Lab. While the number of processors can not be more than the number of particles to track for most of simulations, as it will not use the resource efficiently. For certain simulations, such as parallel optimizations, the number of CPUs is limited to the number of scenarios to be simulated for a parameter set.

### 3.2 Validating the result

The simulations above were finished in around 1 minute for `Pelegant` and 10 minutes for `elegant` on the AMD Athlon nodes of the weed cluster in APS. To convince ourselves the results of the two versions of `elegant` are same, one can compare the output files with the sddsdiff command, which should be available in the SDDS toolkit. For example, to examine the particle coordinates at interior points, we can type:

sddsdiff manyParticles_p.w2 manyParticles_s.w2

which should return that two results are identical.

Validating a parallel program against a uniprocessor program with the requirement of bitwise identical result is notoriously difficult [2], as we may meet some new problems raised from parallel computing, such as different ordering of summations, non-scalable random number generator. Although the simulation results with the discrepancies should conform to IEEE 754 within some tolerance, more consistent results can be expected with more accurate numerical algorithm, such as Kahan's summation formula [3], which has been employed in both serial and parallel versions of `elegant`.

We ran a regression test of 92 cases and validated the results of `Pelegant` with `elegant`. As the random number sequences generated by one CPU and multiple CPUs usually are not same, some test examples can't be validated by comparing the results of `elegant` and `Pelegant`. Those examples have been validated either by mathematical formulae or their physical meaning.

[1] For the weed cluster in APS, all the MPI related commands have been put in the options of csub command. User could just type:
csub -mvapich2 11 Pelegant manyParticles_p.ele

# 4 Using Pelegant efficiently

As the elegant parallelization is an on-going project, we first parallelized the tracking elements through partitioning the particles to multiple CPUs. We have parallelized 95 out of 102 elements for tracking simulations. The elements which have not been parallelized are listed at the end of this manual. As most time-intensive elements have been parallelized, we can expect a good speedup and efficiency for this type of simulations. If a simulation is slow due to a particular element, the user is encouraged to send the input files to us for a performance study. After the tracking elements have been parallelized, I/O becomes the bottleneck of the simulation, especially for simulations requiring a very large number of particles. We developed parallel SDDS library [4] to meet the I/O requirement for large-scale computation. The parallelization of frequency map analysis, dynamic aperture search, and position-dependent momentum aperture determination is discussed in [5]. Recently, we also added several parallel optimization options to Pelegant, including Parallel Genetic Algorithm (PGA), Hybrid Parallel Simplex (HPS) and Parallel Particle Swarm Optimization (PPSO). The usage of these parallel optimization methods can be found in the parallel_optimization_setup section of User's Manual for elegant.

## 4.1 Parallelization overview

To help users run simulations with Pelegant more efficiently, we would like to introduce our parallelization approach for the tracking elements briefly. We parallelize elegant using a master/slaves (manager/workers) model. The time-intensive tracking parts of elegant are being parallelized gradually. The other parts are done (redundantly) by all the processors, which is acceptable since those processors have already been allocated to a particularly Pelegant run. We divide the beamline elements to four classes:

1. Parallel element: only the slave processors will do the tracking. Each slave is responsible for a portion of particles.

2. MP (multiprocessor) algorithm: the master will participate in the tracking, but it only gets the result of collective computations (e.g. sum, average) from the slaves, without doing any computations itself.

3. Uniprocessor element: must be done by master (for now) and modifies particle coordinates.

4. Diagnostic: could run on master or slaves, but doesn't change particle coordinates.

A flag was added to elegant's dictionary for each beamline element to identify its classification. The master is responsible for gathering and scattering particles as needed according to this classification. Communications are minimized to achieve the best efficiency of parallelization. For example, it is not necessary to communicate the coordinates of particles between master and slaves when tracking through two continuous parallel elements.

## 4.2 Achieving high performance

As Pelegant uses parallel SDDS library for parallel I/O operations, it is suggested to use a parallel file system to achieve best performance. Pelegant has been tested on several

parallel file systems, such as GPFS, PVFS, Lustre, etc.. The NFS file system is compatible with parallel I/O if configured properly, but the speed for I/O operations is not scalable.

In our master/slave model, the master is responsible for tracking in the Uniprocessor elements. To run simulations efficiently, we also suggest when possible that the user arrange all serial elements in a continuous sequence, which will minimize the communication overhead for gathering and scattering particles. This become less necessary for the latest Pelegant, as most elements have been parallelized.

For ANL users (or others who have an ALCF account), we can provide help to perform runs on the Fusion cluster (2,560 Intel CPU-cores, each with a 2.67 GHz Xeon) or Intrepid supercomputer (163,840 cores, PowerPC 450, 850 MHz) at ANL. `Pelegant` is pre-built and available on both systems.

# 5 What is not supported

In our regression test for `Pelegant`, we excluded certain types of tests, which are not supported in this version of `Pelegant`:

1. All of the tests tracking beam with one particle (or the number of particles is less than the number of processors) were excluded, as such a simulation will not benefit from the parallelization approach we employed.

2. The second type of tests we excluded are those needing slice analysis. They are not supported in `Pelegant` at present.

It is suggested to run a simulation with a small workload first before trying the final time-intensive simulation. Also, use of WATCH elements with `FLUSH_INTERVAL=1` can be helpful in verifying that progress is being made. We listed the commands have been tested in appendices. Users can report bugs with all the input files to ywang25@aps.anl.gov or borland@aps.anl.gov.

# 6 Appendices

## 6.1 Elements that have been parallelized in `Pelegant`

The particular physical elements that take advantage of parallel computation in the present version of the code are:

1. Drift spaces, dipoles, quadrupoles, sextuoples, higher multipoles, dipole correctors, and wiggler magnets, whether symplectically integrated or modeled with a transport matrix (up to 3rd order). Symplectically integrated elements can optionally include both quantum and classical synchrotron radiation effects.

2. Radio frequency cavities, including accelerating and deflecting cavities, with constant field amplitude.

3. Accelerating cavities with phase, voltage, and frequency modulation or ramping.

4. Beam collimating and scraping elements.

5. Field-map integration elements, such as dipoles, x-y dependent maps, and solenoids.

6. Reference energy matching points.

7. Beam watch points, which may involve parallel computation of beam moments or dumping of particle coordinates.

8. Scattering elements, including lumped-element simulation of synchrotron radiation.

We have a very small number of elements which have not been parallelized. They are either new elements or used infrequently. Here is an explicit list of the elements which have NOT been parallelized:

```
RMDF IBSCATTER REMCOR TFBPICKUP TFBDRIVER MHISTOGRAM
```

### 6.2 Commands that have been tested for `Pelegant`

In addition to tracking, the following features of `elegant` may be used in the parallel version:

1. Optimization with tracking. In this case, user can choose the optimization to be supervised at the serial level or parallel level, while the tracking is done in parallel.

2. Computation and output of Twiss parameters and transport matrices along a beamline.

3. Computation and output of beam statistics along a beamline.

4. Alteration of element properties, loading of element parameters from external files, and transmutation of element types.

5. Scanning of element properties in loops. In this case, the scanning is supervised at the serial level while the tracking is done in parallel.

6. Use of internally-generated or externally-supplied particle distributions.

7. Addition of random errors to accelerator components.

8. Computation and output of closed orbits.

Here is an explicit list of the commands:

```
alter_elements
bunched_beam
closed_orbit
error_control
error_element
link_elements
load_parameters
matrix_output
optimization_setup
optimization_term
optimization_variable
optimize
```

```
run_control
run_setup
sasefel
save_lattice
sdds_beam
stop
subprocess
track
transmute_elements
twiss_output
vary_element
```

## References


[1] M. Borland, "elegant: A Flexible SDDS-Compliant Code for Accelerator Simulation," Advanced Photon Source LS-287, September 2000.

[2] W. D. Gropp, "Accuracy and Reliability in Scientific Computing," chapter Issues in Accurate and Reliable Use of Parallel Computing in Numerical Program. SIAM, 2005.

[3] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," ACM Computing Surveys, 23(1):5-48, March 1991.

[4] H. Shang *et al.*, "Parallel SDDS: A Scientific High-Performance I/O Interface," Proc. ICAP2009, 347-350.

[5] Y. Wang *et al.*, "Recent Progress on Parallel ELEGANT," Proc. ICAP2009, 355-358.