ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-MOP045

HED-Melt: A COUPLED FRAMEWORK FOR MODELING HIGH-ENERGY-DENSITY CONDITIONS IN ACCELERATORS *

A. J. Dick†, J. Dooling, M. Borland, A. Grannan, Y. Lee, R. R. Lindberg, G. Navrotski
Argonne National Laboratory, Lemont, IL, USA
D. Lee, S. Riedel, University of California Santa Cruz, Santa Cruz, CA, USA
N. Cook, RadiaSoft, Boulder, CO, USA

Abstract

The high-brightness beams used by modern light sources and accelerators present a new challenge for machine protection. These beams, through impacts on beam-intercepting components, may generate high-energy-density (HED) conditions capable of causing significant damage. One significant issue in studying these dynamics is the lack of simulation tools that capture all of the relevant physics. Particle tracking and particle-matter interaction codes are widely used in accelerator design but there is not significant interfacing between the two and few codes are available to simulate the melting and vaporization seen in some beam strikes. `HED-Melt` (**H**igh **E**nergy **D**ensity **M**odeling of **El**ectron Beam Impacts **T**oolkit) is a framework for coupling three physics codes, `elegant`, `FLUKA`, and `FLASH` to model HED conditions in various accelerator components. The toolkit automatically interfaces between the three codes to run a full-physics simulation of HED conditions for a user-defined machine lattice.

INTRODUCTION

The HED-Melt framework automatically interfaces between and executes three physics codes, `elegant` [1, 2], `FLUKA` [3], and `FLASH` [4] for full beam impact simulations in accelerators. Figure 1 shows the workflow of the coupled codes.

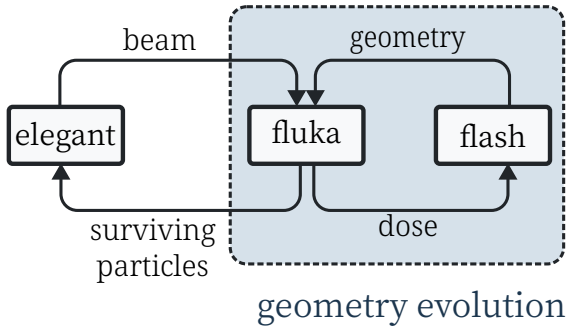

Figure 1: Diagram of HED-Melt workflow showing the information that is passed between codes. The geometry evolution loop occurs multiple times each revolution.

Each of the three codes is responsible for a distinct physics regime. A typical simulation will begin by tracking the beam in `elegant`, initiating an rf fault, until the beam begins to interact with the collimator. At this point, the particle distribution is passed in batches to `FLUKA` where it is used to simulate the energy deposited in, e.g., a collimator. `FLUKA` returns the surviving particles to `elegant` and produces a 3D dosemap that is used by `FLASH`, which simulates the energy deposition and thermodynamic evolution of the impacted material. When a cell is vaporized in `FLASH`, HED-Melt updates the geometry definition used by `FLUKA`. This turn-by-turn treatment gives more accurate modeling of material damage [5]. As the material erodes, the transmission of particles increases, resulting in a protracted beam loss. A static treatment of material geometry can significantly underestimate damage. The modular design allows for significant flexibility in the simulation configuration. Other facilities may use HED-Melt in their design processes to fully model beam impacts or to simply get an estimate of energy absorbed by beamline components.

SETUP AND EXECUTION

In order to keep the framework as accessible as possible, HED-Melt has minimal dependencies, beyond those required by each of the codes, and runs in standard High-Performance-Computing batch job environments. We have also developed tools for setting up generalized beam-dump simulations. The simulation parameters are defined by the user in a YAML "setup" file that is used by a python script to create the directory structure and program files. The user must supply certain parameters in the setup file, which define the loop structure. Additional parameters may be added to template files, which can be filled in at setup to make it easier to modify whole simulations.

The setup file has two main object types, the "simulation" block and three "code" blocks for `elegant`, `FLUKA`, and `FLASH`. The simulation block has the form

```
simulation:
    rootname: example
    domain:
        x: {min: -0.1, max: 0.1, n: 128}
        y: {min: -0.1, max: 0.1, n: 128}
        z: {min: 0.0, max: 1.0, n: 32}
    geometry:
        type: "slab"
        material: "Al"
        center: {x: 0, y: 0, z: 0.5}
        size: {x: 0.02, y: 0.02, z: 0.1}
    params:
```

* Work supported by the U.S. D.O.E.,Office of Science, Office of Basic Energy Sciences, under contract number DE-AC02- 06CH11357.
† ajdick@anl.gov

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-MOP045


```
        splits_per_turn: 6
        rev_period: 1.0e-7
        bunches: 48
```

In this example, a simulation domain ranges from $x/y = [-0.1, 0.1]$ cm, $z = [0.0, 1.0]$ cm and a rectangular aluminum target centered at $(0, 0, 0.5)$ cm has a size of $w = 0.02$, $h = 0.02$, $d = 0.1$ cm. The revolution period is 100 ns with 48 bunches divided into 6 batches.

Each of the code objects have several, machine-specific parameters. These parameters point to the build directory for each code on your machine and define the environment parameters required to run a job. Each code is setup within a subdirectory and accepts four types of files. Job files are batch scripts used to submit and run the job, they are prepended with the machine-specific "env_file" header. The "includes" consists of all files needed to run the code that are not altered by the user. These are often files such as magnet aperture definitions or equations of state tables. "dat_dats" is a list of directories to be created in the code directory. Finally, "par_files" are template files which are filled in with the parameter values defined in the YAML file. This allows users to define their own directory structure and simulation templates within the HED-Melt framework. Below is an example of the `elegant` parameter file block for an APSU simulation.

```
elegant:
    build_dir: "path/to/build"
    machine: "APSU"

    job_files: "init.sh␣run.sh"
    includes: "lattice.lte"
    dat_dirs: "ele_out␣flu_out"
    par_files: "init.ele␣run.ele"

    job_params:
        job_env: "env_file"
        walltime: "HH:MM:SS"
        nodes: 1
    params:
        ...
```

Available Use-Cases

The framework was designed to simulate conditions in the Advanced Photon Source (APS) storage ring [6]. To extend the capabilities to other facilities, HED-Melt supports several geometry definitions. Currently supported are rectangular structure (slab), solid cylinder along x, y, or z-axis (x/y/zcyl), hollow cylinder or beam pipe (x/y/zpipe), or a plane bisecting the simulation domain (plane). Each of these correspond to geometry definitions in `FLUKA` and `FLASH`.

User Modification

Integrating HED-Melt into existing `elegant` tracking simulations requires a few lattice modifications and a template command file. The script element must be inserted in the beam line at the interaction location using the definition

```
TARG: SCRIPT, L=0.0, START_PASS=0, &
    COMMAND="./ctrl-ele %i %o %p"
```

This element will call the `ctrl-ele` script beginning at `START_PASS`. For other simulation purposes, it may be helpful to define three lattice lines, (i) no interaction, (ii) a "static" collimator/aperture which does not call the `FLUKA-FLASH` loop, and (iii) a script beamline which calls to HED-Melt.

Users may add parameters to their lattice or command template files that are filled in by the setup script, such as beam current, number of macro-particles, rf parameters, etc. This makes it easier to change global parameters from the setup YAML file and have all the relevant files updated at once. A placeholder flag "<name>" will be replaced by the parameter 'name: value' listed in the YAML file.

Basic Demonstration

To demonstrate the real-time geometry evolution capabilities of HED-Melt, we simulated an APS-Upgrade [7] beam loss [5]. Figure 2 shows the transverse cross-section of an aluminum collimator during loss of 114 mA with no vertical decoherence. The material is heated by the beam until it is eventually vaporized around the fifth pass. Figure 2(d) shows the difference in beam loss using both static and evolving (HED-Melt) collimator models. The beam loss decreases when the material is vaporized and shows an extended tail.

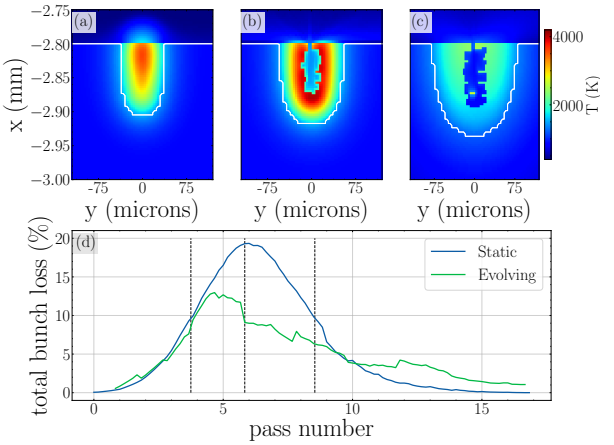

Figure 2: Evolution of collimator temperature and surface during an APS-U beam loss (top). Comparison of evolving and non-evolving models of an APS-U beam loss (bottom).

DATA STRUCTURES AND VISUALIZATION

The data passed between codes (as seen in Fig. 1) comes in three types, particle distribution, 3D dosemap, and 3D geometry definition. It is often useful to check the status of the simulation at these interfaces to validate or analyze results. A collection of SDDS- [8–10] and python-based visualization tools is included in HED-Melt to display several of these standard data files.

`elegant` Particle Loss Each turn, `elegant` records the beam parameters to a `WATCH` file, similar to the data

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-MOP045

collected through machine beam position monitors. The bash script `plotElegantLoss` produces a figure similar to Fig. 3 showing the transverse beam position and the loss rate of particles over the duration of the beam loss. It reads the `elegant` `WATCH` file in and plots the data using SDDS tools.

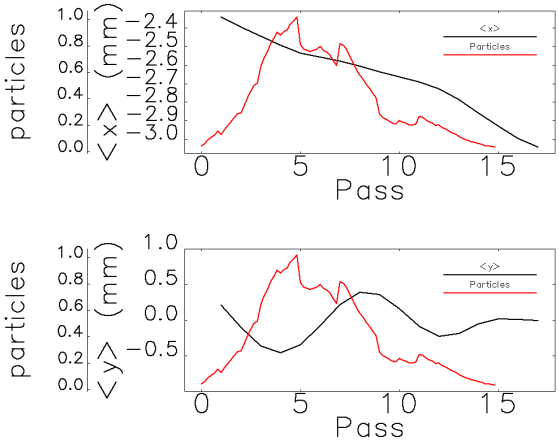

Figure 3: Example of `plotElegantLoss` output. The horizontal and vertical centroids are shown against the loss of macro-particles per turn.

`FLUKA` Dosemaps `FLUKA` produces a dose map of absorbed energy for all cells in the simulation domain. This 3D array has the same dimensions and resolution of the `FLASH` simulation grid, therefore the dose recorded in each cell is directly deposited into the corresponding cell in `FLASH`. The python `plotFlukaDosemap.py` script displays a cross section of the dose, like that in Fig. 4, for a supplied dosemap file (typically contained in the subdirectory `fluka/fluka_dosemaps`).

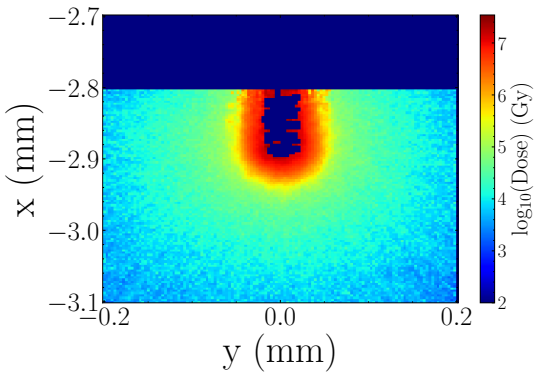

Figure 4: Example of the `plotFlukaDosemap.py` output. The transverse cross-section at the center of the collimator shows the dose absorbed by the material.

Figure 4 shows a damaged region which absorbs no energy. As the material erodes, the damaged region is removed from the geometry definition. This allows particles to pass through the newly vacant area with out being lost or depositing energy.

`FLASH` Material Properties and Evolution The most relevant material properties are output by `FLASH` in a series of plot files during the simulation and the full state of the simulation is recorded in a checkpoint file at the end of each batch of depositions. The plot files are used to visualize specific parameters while the checkpoint files are used to restart the simulation and to create the `FLUKA` geometry definition.

The python script `plotFlashState.py` displays a cross section of a specific variable (phase, temperature, density, etc) of a single plot file output. Figure 5 shows an example of the electron temperature at the central cross section near the end of a beam loss. The same damaged region is seen here as in the `FLUKA` dosemap. The python script `plotFlashEvolution.py` produces an animation of a specific cell variable for the duration of the simulation using all plot files in the `flash/outputs/` subdirectory.

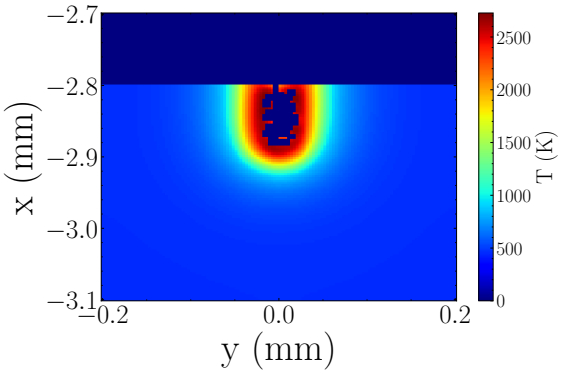

Figure 5: Example of the `plotFlashState.py` output. The transverse cross-section at the center of the collimator shows electron temperature at a single timestep. The script may also be used to plot other cell properites.

CONCLUSION

HED-Melt is a generally applicable toolkit for simulating HED conditions in beam intercepting components of storage rings. The framework allows for easy integration with existing `elegant` tracking simulations. This method allows accelerator facilities to estimate the dose and damage to beamline elements during beam losses and may aid in the development of damage mitigation methods.

REFERENCES

[1] M. Borland, "Elegant: a flexible SDDS-compliant code for accelerator simulation", Argonne National Laboratory, Rep. LS-287, 2000. doi:10.2172/761286

[2] Y. Wang and M. Borland, "Pelegant: a parallel accelerator simulation code for electron generation and tracking", in *AIP Conf. Proc. Vol. 877*, vol. 877, Lake Geneva, Wisconsin, USA, Jul. 2006, pp. 241–247. doi:10.1063/1.2409141

[3] G. Battistoni *et al.*, "Overview of the FLUKA code", *Ann. Nucl. Energy*, vol. 82, pp. 10–18, 2015. doi:10.1016/j.anucene.2014.11.007

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-MOP045

[4] B. Fryxell *et al.*, "Flash: an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes", *Astrophys. J. Suppl.*, vol. 131, no. 1, p. 273, 2000. doi:10.1086/317361

[5] AJ. Dick *et al.*, "Coupled simulation framework for modeling high-energy-density conditions in accelerators", presented at IPAC'25, Taipei, Taiwan, Jun. 2025, paper MOPS076, unpublished.

[6] R. Fuja *et al.*, "The APS machine protection system (MPS)", in *AIP Conf. Proc. Vol. 390*, vol. 390, Argonne, Illinois, USA, May 1997, pp. 454–459. doi:10.1063/1.52324

[7] M. Borland *et al.*, "The upgrade of the Advanced Photon Source", in *Proc. IPAC'18*, Vancouver, Canada, Apr.–May 2018, pp. 2872–2877. doi:10.18429/JACoW-IPAC2018-THXGBD1

[8] M. Borland, "A self-describing file protocol for simulation integration and shared post-processors", in *Proc. PAC'95*, Dallas, Texas, USA, May 1995, paper WAE11, pp. 2184–2186.

[9] M. Borland, "Applications Toolkit for Accelerator Control and Analysis", in *Proc. PAC'97*, Vancouver, B.C., Canada, May 1997, paper 6P020, pp. 2487–2489.

[10] R. Soliday, M. Borland, L. Emery, and H. Shang, "New Features in the SDDS Toolkit", in *Proc. PAC'03*, Portland, Oregon, USA, May 2003, paper FPAG005, pp. 3473–3475. https://jacow.org/p03/papers/FPAG005.pdf