Overview:

- What is GPID and where to get it
- Where is input information for GPID coming from
- Structure of GPID bank
- Structure of `GPID.map`
- Calibration procedure and `gpid_mon` utility
- Usage hints, limitations, things to do.
- GPID is extension of PART/TBID for photon runs with the Start Counter. It tries to do particle ID on a track by track basis. The method uses the momentum of detected particle, and sequentially calculates trial values of $\beta$ for the particle for all possible particle identities. Each one of the possible identities is tested by the trial value $\beta$ for a given particle type to the empirically measured value of $\beta$ (as determined by CLAS tracking and time-of-flight information). The particle is assigned the identity that provides the closest trial value of $\beta$ to the empirically measured value of $\beta$. The GPID algorithm also attempts to find a matching photon in the tagging system for every charged particle detected in CLAS.

- The source is in clas CVS repository: packages/pid/make_gpid.c
  The source code has many comments in it. They explain how and what it is doing.

- List of banks required to build GPID bank: PART, TBID, TBTR, TDPL, SCRC, STR, TAGR

- Map file used: GPID.map (or GPID system from caldb)

- Functions:
  - `int initGPID(int run)` reads cuts from GPID.map for run number `run`
  - `clasGPID_t *makeGPID(int bankNum, int calib)` makes GPID bank. `bankNum` is a bank number to make. It also directs which PART/TBID banks to use. Usually it is 0 during cooking and 1 when you rebuild BID banks. If `calib=1`, `makeGPID` runs in calibration mode with cuts wide open. For normal running `calib=0`
Table 1: Structure of GPID bank

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>pid</td>
</tr>
<tr>
<td>vector3_t</td>
<td>vert</td>
</tr>
<tr>
<td>vector4_t</td>
<td>p</td>
</tr>
<tr>
<td>int</td>
<td>q</td>
</tr>
<tr>
<td>int</td>
<td>trkid</td>
</tr>
<tr>
<td>int</td>
<td>sec</td>
</tr>
<tr>
<td>int</td>
<td>paddle</td>
</tr>
<tr>
<td>float</td>
<td>dedx</td>
</tr>
<tr>
<td>float</td>
<td>beta</td>
</tr>
<tr>
<td>int</td>
<td>sc_stat</td>
</tr>
<tr>
<td></td>
<td>sc_stat = 0</td>
</tr>
<tr>
<td>float</td>
<td>sc_time</td>
</tr>
<tr>
<td>float</td>
<td>sc_len</td>
</tr>
<tr>
<td>int</td>
<td>st_stat</td>
</tr>
<tr>
<td></td>
<td>st_stat = 0</td>
</tr>
<tr>
<td>float</td>
<td>st_time</td>
</tr>
<tr>
<td>float</td>
<td>st_len</td>
</tr>
<tr>
<td>float</td>
<td>mass</td>
</tr>
<tr>
<td>int</td>
<td>mass_ref</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>betam</td>
</tr>
<tr>
<td></td>
<td>0, 2: (\beta = \frac{sc_{\text{len}}}{c(sc_{\text{time}} - tpho - tprop)})</td>
</tr>
<tr>
<td></td>
<td>1: (\beta = \frac{c(st_{\text{len}} - st_{\text{time}})}{c(sc_{\text{time}} - st_{\text{time}})})</td>
</tr>
<tr>
<td>float</td>
<td>epho</td>
</tr>
<tr>
<td>float</td>
<td>tpho</td>
</tr>
<tr>
<td>int</td>
<td>tagrid</td>
</tr>
<tr>
<td>int</td>
<td>ngrf</td>
</tr>
<tr>
<td>int</td>
<td>ppid</td>
</tr>
</tbody>
</table>
Structure of GPID.map

Map: GPID.map

Subsystem: deuteron, nitems: 3
  Item: dbeta, length: 1, type: float, narray:1
  Item: high, length: 50, type: float, narray:1
  Item: low, length: 50, type: float, narray:1

Subsystem: kaon, nitems: 3
  Item: dbeta, length: 1, type: float, narray:1
  Item: high, length: 50, type: float, narray:2
  Item: low, length: 50, type: float, narray:2

Subsystem: kpi, nitems: 2
  Item: offset, length: 1, type: float, narray:1
  Item: slope, length: 1, type: float, narray:1

Subsystem: pion, nitems: 3
  Item: dbeta, length: 1, type: float, narray:1
  Item: high, length: 50, type: float, narray:2
  Item: low, length: 50, type: float, narray:2

Subsystem: proton, nitems: 3
  Item: dbeta, length: 1, type: float, narray:1
  Item: high, length: 50, type: float, narray:2
  Item: low, length: 50, type: float, narray:2

Subsystem: triton, nitems: 3
  Item: dbeta, length: 1, type: float, narray:1
  Item: high, length: 50, type: float, narray:1
  Item: low, length: 50, type: float, narray:1
gpid_mon utility

Location: packages/utilities/gpid_mon/

asu.jlab.org{pasyuk}: gpid_mon -h

Usage: gpid_mon [options] file1 [file2] etc....

Options:
  -o<outfile>   output hbook file (default=gpid_monXXXXX.hbook)
  -M<#>        Process only # number of events
  -R            Regenerate the TBID/PART and associated banks
  -c            Run in calibration mode
  -T<#>        Set trigger bit mask (default=0xffff)
  -h            Print this message.
Figure 1: Vertex time difference vs. $\text{betam}$ (left). $\text{betam}$-$\beta$ vs. $\text{betam}$ (right)

There are kumac files in the same directory.

mpr_deuteron.kumac
mpr_kaon.kumac
mpr_pion.kumac
mpr_proton.kumac

These macros slice appropriate histograms of vertex time difference vs. beta into 50 slices. In each beta slice one should choose appropriate cuts (low/high) around central peak. Use your judgment. Usually it is about $0.8 - 1.5$ ns from the peak. Don’t cut too tight.
Figure 2: Left is before the cuts applied. Right, after the cuts.

After calibration you will see something like this. \texttt{dbeta} cut inverts the sign of \texttt{pid} if particle fails it. At the moment this feature is commented out.
Basic hints on usage

- There is no universal recipe, each analysis is unique
- Use particles with ngrf>0.
- Ignore particles with ngrf=0 as if they were not detected at all. Do not include them in number of particles if it is one of your event selection criteria
- ngrf=1 is unambiguous. ngrf>1 requires special treatment. You have two choices: either throw away this event and account for this in inefficiency, or use means other than timing to choose between photons (kinematical cuts)
- As usual, be careful when selecting kaons. For skimming kaons do not rely on pid that comes out from GPID alone. Use a cut on the mass too. Something like this:

```cpp
if (abs(GPID->gpid[j].pid) == KPlus ||
    (GPID->gpid[j].mass <= 0.7 &&
     GPID->gpid[j].mass >= 0.3 &&
     GPID->gpid[j].q >0))
    Kp_found++;
```

- For multi track events it is possible that GPID associates particles with the same photon, but they are coming from different interactions (accidental coincidence). A comparison of their vertex times and z-components of vertex often helps.

Status, Limitations and Things to Do

- It is working
- GPID is included in a1c and gflux
- Neutral particles identification is not done in GPID. It has just a copy from PART/TBID. This is general problem with all PID packages used in CLAS.