Femtocode: query system for HEP

What’s a query system?
User asks a question, gets an answer quickly enough to explore the data. Like a Google query, but aggregating HEP data, returning (e.g.) histograms. Embedded within an analysis script; provides sliced projections of the data for users to fit/plot/analyze in any way they want.

How it differs from what we do now
Physicists arrange data as sets of files that have to be filtered into progressively smaller sets of files until the final set is small enough for real-time data analysis. Instead, we propose a service that serves aggregated views of analysis object data on demand. Must be responsive to requests in real-time: ~1 sec for each scan over a dataset.

High-level language
User writes expressions that pick apart the structure of objects within arbitrary-length lists, to any depth of nesting.

Higher-order functions like .map, .pairs, .filter, .reduce instead of explicit for loops.

Femtocode query language is distributed in quoted snippets throughout a structured workflow and tree of aggregators. (See http://histogrammar.org for histogram abstraction.)

Low-level execution
No objects at runtime:
All nested structures are represented as homogeneous arrays.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>values</td>
<td>[ [44], [14], [3, 2.2], [5, 1.3], [16] ]</td>
</tr>
</tbody>
</table>

becomes

<table>
<thead>
<tr>
<th>data</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>3</td>
</tr>
<tr>
<td>data</td>
<td>4</td>
</tr>
<tr>
<td>data</td>
<td>4</td>
</tr>
</tbody>
</table>

muons.map (mu1 => 
muons.map (mu2 => 
mu2.p**2 + 0.105658**2; 
mu1 + mu2.p**2 + 0.105658**2; 
mu1 + mu2)).max)

Each non-S statement can be a branchless, loopless GPU kernel or all statements can be grouped by @size as CPU loops.

Performance studies
- Typical analysis asks for ~10 TB of input data.
- Typical query touches < 1% of the columns.
- Disk read at 40 MB/sec => 2.6 sec for 1000 disks.
- Typical analysis touches < 10% of the columns.
- Therefore, fill 1 TB of a cluster’s RAM as cache.
- Cache read at 1 GB/sec => 0.1 sec for 1000 cores.

Microbenchmarks to get a sense of scale:
Just adding a constant to an array of 64-bit floats on different systems.

- 0.018 MHz CMSSW EDAnalyzer (C++ event framework)
- 1.5 MHz TTree::Draw in ROOT (HEP analysis toolkit)
- 2.8 MHz minimal disk read and unzip
- 12 MHz allocating C++ objects on heap, iterating, deleting
- 31 MHz allocating C++ objects on stack and iterating
- 54 MHz Femtocode loop, current implementation
- 256 MHz minimal single-threaded loop in C (our goal)
- 8 000 MHz same loop on 128 threads in KNL’s MCDRAM
- 57 000 MHz equivalent on Tesla P100-SXM2 GPU

Query server
Distributes query among compute nodes, avoids duplicate work.

Examples:
- mu1 = muons.size >= 2
- #1: map over #size
- #2@size: #2.size
- #3: #2.size
- #4: #size
- #5: #size
- #6: #size
- #0: mu1
- #7: map over muons

No runtime errors:
Type system includes min/max ranges of numbers and collection lengths, so that out-of-domain errors are caught at compile-time.

Examples: x/y must be "if y != 0: x/y else: None". filter("goodmuons.size > 2") above ensures that we can assign two of them as mu1, mu2.

Total functional language:
Every expression that compiles returns a result. (For simplicity, recursion is not allowed.)