Tips for the Scientific Programmer

Michele Simionato@GEM Foundation
This talk is about "Middle Performance Computing"

- profiling is invaluable for finding bottlenecks like slow operations in inner loops, but I do that 1-2 times per year
- what it is really essential is instrumenting your code
- what makes the difference is using the right library and the right architecture / data structure
Input/output formats

• I learned the hard way a very essential lesson: *never, EVER change the input formats*
• You cannot. Really, you can not.
• Even if it is impossible to get right the input format at the beginning 😞
• There is more freedom with the output formats
• Where you can really work is on the internal formats
Inputs formats we are using

- INI (good, but **TOML** would have been better)
- XML/NRML/XSD (could have been simpler)
- CSV (should have been used more)
- HDF5 (in rare cases: UCERF3, GMPE tables)
- ZIP (okay)
Output formats we are using

- **XML / NRML**: we are removing it
- **CSV** with pre-header: we are using it more and more
- **HDF5**: used sometimes
- **NPZ**: by necessity
## Outputs from calculation 24329

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>32393</td>
<td>Full Report</td>
<td>fullreport</td>
<td>Download rtt</td>
</tr>
<tr>
<td>32394</td>
<td>Hazard Curves</td>
<td>hcurves</td>
<td>Download csv Download xml Download rpe</td>
</tr>
<tr>
<td>32395</td>
<td>Hazard Maps</td>
<td>hmaps</td>
<td>Download csv Download xml Download rpe</td>
</tr>
<tr>
<td>32396</td>
<td>Input Files</td>
<td>input</td>
<td>Download zip</td>
</tr>
<tr>
<td>32397</td>
<td>Realizations</td>
<td>realizations</td>
<td>Download csv</td>
</tr>
<tr>
<td>32398</td>
<td>Seismic Source Groups</td>
<td>sourcegroups</td>
<td>Download csv</td>
</tr>
<tr>
<td>32399</td>
<td>Uniform Hazard Spectra</td>
<td>uhs</td>
<td>Download csv Download xml Download rpe</td>
</tr>
</tbody>
</table>
Internal formats we are using

- .hdf5
- .toml
- .sqlite

They are good 😊
The choice of the data format has a big performance impact

- XML/CSV exporters
- XML/CSV importers
- clearly the choice of the internal formats is even more important: **HDF5 is the way to go**
Task distribution

- we are using *multiprocessing/zmq* on a single machine
- and *celery/rabbitmq/zmq* on a cluster

*celery/rabbitmq* is not ideal for our use case but it works enough, including the REVOKE functionality
our biggest issue :-(

mean=615±349 seconds
Slow tasks

- slow tasks have been a PITA for years 😞
- a few months ago we had a breakthrough: subtasks
- we made the output receiver able to recognize tuples of the form (callable, arg1, arg2, ...) and to send them as tasks
task producing subtasks:

def task_splitter(sources, arg1, arg2, ...):
    blocks = split_in_blocks(sources, maxweight)
    for block in blocks[:-1]:
        yield (task_func, block, arg1, arg2, ...)
    yield task_func(block[-1], arg1, arg2, ...)

heavy tasks can be split in many light tasks
the weight of a seismic source is the number of earthquakes it can produce
it can be very different from the duration of the calculation
Calibrating the computation

- we introduced a task splitter able to perform a subset of the calculation and to **estimate** the expected task duration depending on the weight
- it can split the calculation in subtasks with estimated runtime smaller that an user-given `task_duration` parameter
Automatic task splitting

- successively, we made the engine smart enough to determine a sensible default for the \textit{task\_duration}, depending on the number of ruptures, sites and levels
- \textit{=> slow tasks are greatly reduced}
- \textit{except for non-splittable sources}
Solving the data transfer issue

• we switched to using zmq to return the outputs
• we switched to NFS to read the inputs (and it is also useful for sharing the code)
• **important**: do not produce too many tasks, the data transfer will kill you, or the output queue will run out of memory, or both
Memory occupation

- A big problem we had to fight constantly is running out of memory (even with 1280 GB split on 10 machines)
- Notice that running out of memory *early* can be a good thing
- It is all about the tradeoff memory/speed
- NB: Memory allocation can be the *dominating* factor for performance
How to reduce the required memory

- use as much as possible numpy arrays instead of Python objects
- use a site-by-site algorithm if you really must
- remember that big tasks are still better, if you have enough memory
- we measure the memory with psutil.Process(pid).memory_info()
Saving memory by yielding partial results

def big_task(sources, arg1, arg2, ...):
    accum = []
    for src in sources:
        accum.append(process(src, arg1, arg2, ...))
        if len(accum) > max_size:
            yield accum
            accum.clear()  # save memory
    if accum:
        yield accum

Lesson: a nice parallelization framework really helps