AthenaMP: parallelizing Athena using `fork` and Copy-On-Write

Sébastien Binet

Lawrence Berkeley Laboratory

10-10-2008

Thread-based or Process-based concurrency

Problem(s)

- how to parallelize a huge code base as `Athena` ?
- how to easily harvest the multi/many cores' computing power ?
- how to efficiently use the limited memory available per core ?

thread-based parallelism

- same address space: memory efficiently shared
- concurrency ok
- hard to get them right (locks, races, . . .)

process-based parallelism

- different processes $\Rightarrow$ different address spaces: no mess
- rely on the kernel to efficiently share memory for us
  - let the experts do their magic
  - 1 stone to kill 2 birds

Thread-based or Process-based concurrency - II

`fork` & COW recipe

- initialize your application
- `fork` off as many subprocesses as you wish or can
- let’em run
- join/finalize when processing is done

Issues

- sharing memory is easy but once *’unshared’*, you can’t *’reshare’*
  - need to optimize when to `fork`
- I/O (mostly the writting part)
  - apart from HDF5+MPI (marginally used in Atlas) no parallel I/O available
  - multiplexing through `select/poll/epoll/asio/...`
  - *’poor man’* parallel I/O (each process has its own output file which are concatenated/massaged in a post-processing step)

AthenaMP implementation

rational

- avoid client changes
- shove the MP-stuff inside Athena instead of putting it as a layer on top of it
- use the python module multiprocessing (backported from 2.6) for the process management
- write a new event loop manager as a usual Gaudi component to encapsulate the parallelism handling
- modify the I/O-related components appropriately

```
class MpEventLoopMgr (PyAthena.Svc):
    def executeRun (self, maxevt):
        """Process `maxevt` events as Run (beginRun->endRun)
        """
        if self._ncpus <= 0:
            return self._evtloop_mgr.executeRun (maxevt)

        import multiprocessing as mp
        _info ("nbr of workers: %i", self._ncpus)
        _info ("master workdir: %s", self._wkdir)
        workers = mp.Pool (processes=self._ncpus,
                           initializer=self._worker_bootstrap)
        results = workers.map_async (func=batch_run,
                                     iterable=(maxevt,)*self._ncpus)
```

_worker_bootstrap

- function called after `fork`
- change work dir
- reopen file descriptors
- tickle the `IoComponentMgr`

```
class IIoComponentMgr
{
  /** allow a @c IIoComponent to register itself with this
   *  manager so appropriate actions can be taken when e.g.
   *  a @c fork(2) has been issued (this is usually handled
   *  by calling @c IIoComponent::io_reinit on every registered
   *  component)
   */
  virtual
  StatusCode io_register (IIoComponent* iocomponent) = 0;

  /** @brief: reinitialize the I/O subsystem.
   *  This effectively calls @c IIoComponent::io_reinit on all
   *  the registered @c IIoComponent.
   */
  virtual
  StatusCode io_reinitialize () = 0;

  /** @brief: finalize the I/O subsystem.
   *  Hook to allow to e.g. give a chance to I/O subsystems to
   *  merge output files.
   */
  virtual
  StatusCode io_finalize () = 0;
};
```

```
class IIoComponent
{
  /** callback method to reinitialize the internal state of
   *  the component for I/O purposes (e.g. upon @c fork(2))
   */
  virtual
  StatusCode io_reinit () = 0;
};
```

- implemented by `THistSvc`, `AthenaPoolSvc`, ...
- reopen input ROOT files
- open output ROOT files
  - created in the worker's own directory
  - take care of migrating all the objects of *'already opened for writting'* ROOT files to the new ones

```python
class MpEventLoopMgr (PyAthena.Svc):
    def executeRun (self, maxevt):
        """Process `maxevt` events as Run (beginRun->endRun)
        """
        if self._ncpus <= 0:
            return self._evtloop_mgr.executeRun (maxevt)

        import multiprocessing as mp
        _info ("nbr of workers: %i", self._ncpus)
        _info ("master workdir: %s", self._wkdir)
        workers = mp.Pool (processes=self._ncpus,
                           initializer=self._worker_bootstrap)
        results = workers.map_async (func=batch_run,
                                     iterable=(maxevt,)*self._ncpus)
```

batch_run

- inject a filter algorithm in front of alg-sequence
  - accept/reject events based on local process-id and current event number
- effectively implement a round-robin filter
- call the `executeRun` of the wrapped event loop manager

```
class MpEventLoopMgr (PyAthena.Svc):
    def finalize (self): ...
```

- tickle `IIoComponentMgr::io_finalize` (when a forked process)
- master will run the merge of output files
  - usually trivial for ROOT files containing histos and ntuples
  - trickier for ROOT/POOL files
    - take care of POOL links/references
    - actually just a few integers here and there to offset by the right amount
    - needs some modifications in the `AthenaPOOL` layer to enable usage of the fast-merge mode (*à la* `hadd`)
    - right now: pedestrian/manual approach (slower)
  - I wish there were a general `pool_merge` command !

Status

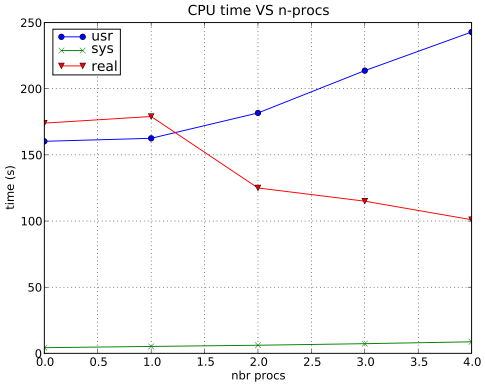
- tested on `Athena` reconstruction and physics jobs
- runs **ok** but detailed cross check needed
  - development+validation of the tools to perform cross check of data files' content $\Rightarrow$ **in progress**
- `fork` is fired after `initialize`
  - doesn't capture the *first event* lazy initialization
    - loading of reflex dictionaries
    - some conditions data callbacks being triggered
  - plan is to leverage the new `Gaudi` final state machine to migrate some of this lazy initialization into `start`
  - `fork`ing **after** first event is a bit more complicated but will be done
- haven't (yet?) parallelized interactive event loop manager

- package `multiprocessing` backported from python 2.6 has a few races fixed *w.r.t.* `pyprocessing`
- still some races remain

preliminary results

cpu

```
4procs 242.85s user 8.71s system 249% cpu 1:40.99 total
3procs 213.67s user 7.30s system 191% cpu 1:55.12 total
2procs 181.67s user 6.13s system 149% cpu 2:05.77 total
1procs 162.52s user 5.22s system 093% cpu 2:59.18 total
0procs 160.25s user 4.28s system 094% cpu 2:53.45 total
```

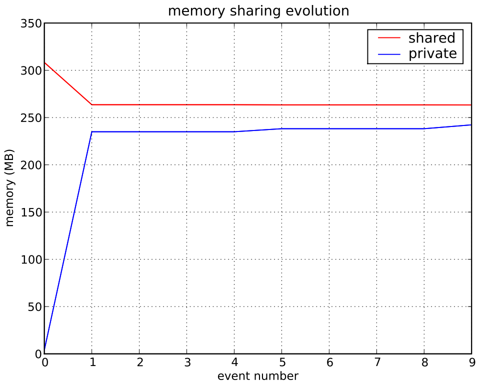


preliminary results - II

memory

process: $\sim$ 700*MB VMem* and $\sim$ 420*MB RSS*

```
(before) evt 0:  private:  004 MB | shared:  310 MB
(before) evt 1:  private:  235 MB | shared:  265 MB
...
(before) evt50:  private:  250 MB | shared:  263 MB
```


memory sharing evolution
shared
private
memory (MB)
event number