Australian Government

Defence

A Cat In The Matrix

Gareth Elliott

Research Computing

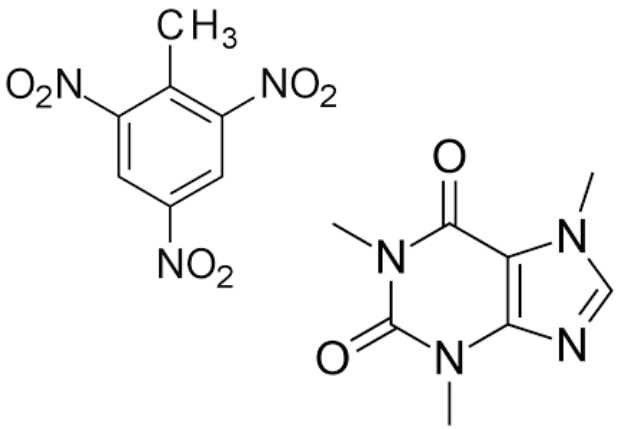
Computational Science

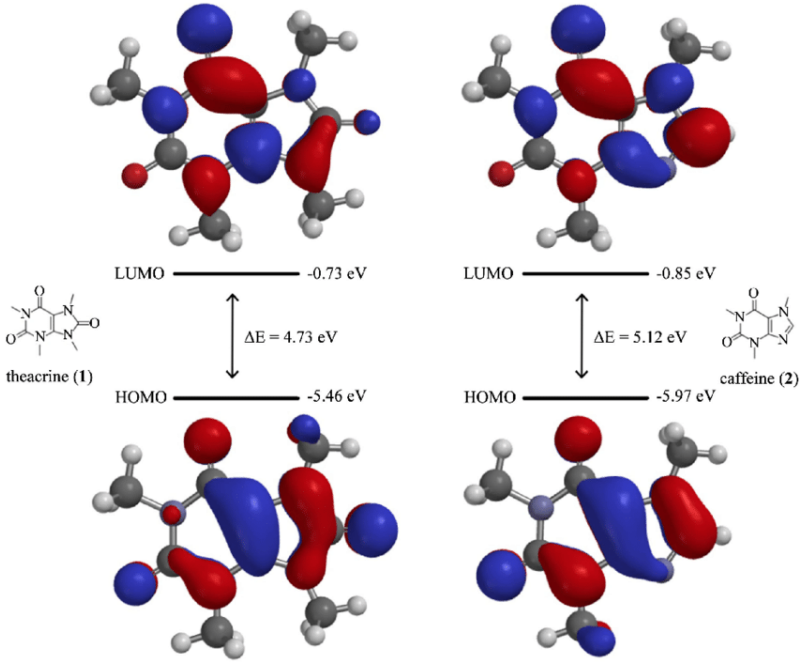
Getting the Neodymium from Harddrive Magnets

- Why do we do calculations?
  - Fun (make computers work hard)
  - Cheaper than building a full (or even small) scale model
  - Can easily vary input conditions or models
  - Run many scenarios in parallel
  - Avoid having to go into space
  - Distance yourself from toxic or radioactive substances
  - Distance yourself from the strange people who work with said toxic and radioactive substances

Running a Calculation

Input → Engine → Output



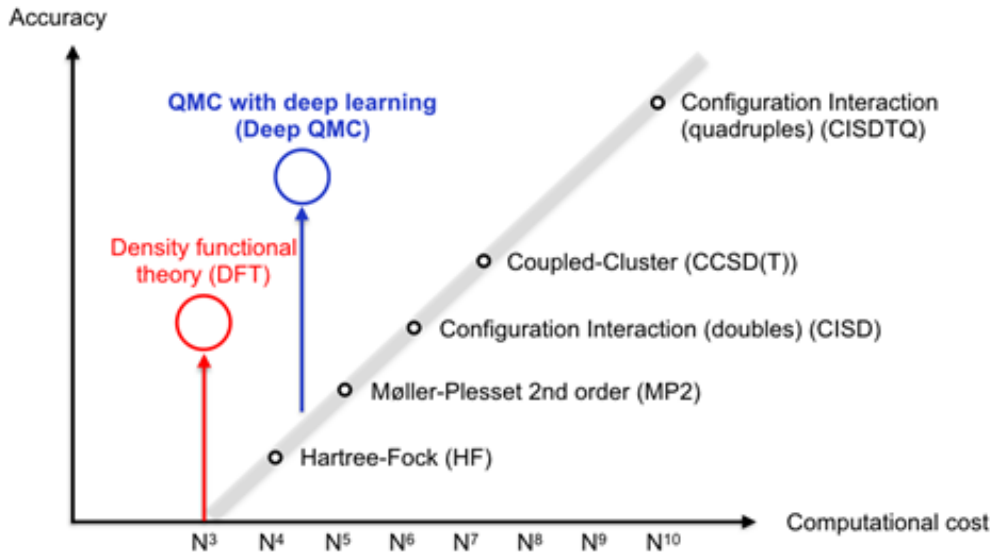


Ashburn, B. , Le, D. and Nishimura, C. (2019) Computational Analysis of Theacrine, a Purported Nootropic and Energy-Enhancing Nutritional Supplement. *Computational Chemistry*, 7, 27-37. doi: 10.4236/cc.2019.71002

Sometimes Calculation Take a While

- Large problem size
- Complicated geometry
- Many samples required for reasonable statistics
  - E.g., Many time steps in a molecular dynamics simulation
- Greater resolution required
  - E.g., Finer mesh in CFD around a critical point
- Require a more sophisticated model
  - E.g., Use Coupled Cluster instead of DFT

$$\hat{H}\psi(\mathbf{r}_1,\ldots,\mathbf{r}_N) = E\psi(\mathbf{r}_1,\ldots,\mathbf{r}_N)$$



https://www.chemistryworld.com/news/quantum-chemistry-simulations-offers-beguiling-possibility-of-solving-chemistry/4011541.article

The Solution?

"Just use more CPUs"

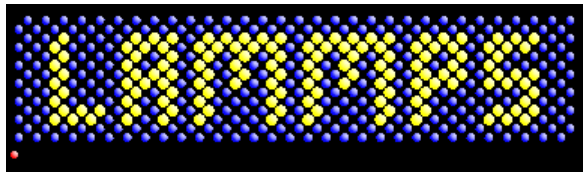
- Poorly written code
- Poor scaling
- Poor…
  - Don't have access to large enough compute
  - Have to pay for more/larger licenses

So they'll just have to run it for longer?

- Pretty much
- But…
- What happens if there is an "interruption"?
  - Node goes down
  - VM goes down
  - Entire machine goes down (cooling failure, power failure, …)
  - Something else
- Need to restart to apply security patches
- What happens if the researchers need to run a calculation for weeks…

Checkpointing!

- The ability to stop and res
- Require the ability to store
  - Random number states
  - Positions of particles
  - Velocities of particles
  - State of the optimiser
- The good news is that ma m already
- Its kind of like déjà vu…



restart command

Syntax

```
restart 0
restart N root keyword value ...
restart N file1 file2 keyword value ...
```

- N = write a restart file on timesteps which are multiples of N
- N can be a variable (see below)
- root = filename to which timestep # is appended
- file1,file2 = two full filenames, toggle between them when writing file
- zero or more keyword/value pairs may be appended
- keyword = *fileper* or *nfile*

```
fileper arg = Np
  Np = write one file for every this many processors
nfile arg = Nf
  Nf = write this many files, one from each of Nf processors
```

Examples

```
restart 0
restart 1000 poly.restart
restart 1000 restart.*.equil
restart 10000 poly.%.1 poly.%.2 nfile 10
restart v_mystep poly.restart
```

Gaussian 16 Frequently Asked Questions

How can I restart a job that was interrupted?

Many Gaussian jobs that are stopped prematurely — e.g., due to a machine crash, a power failure, manually killing the job — can be restarted. These include geometry optimizations, frequency calculations, and CCSD and EOM-CCSD calculations. The technique to restart the jobs varies depending on the type of job. This FAQ will discuss some common cases.

Be aware that all restarts require the checkpoint file from the previous job. Some job types also require the read-write file. If the required file(s) have been deleted, then the job cannot be restarted.

Restarting Interrupted Geometry Optimizations: Opt=Restart

If a geometry optimization is terminated before completion due to an external factor, you can resume it simply by adding the **Restart** option to the **Opt** keyword into the route section of the original job, as in this example:

```
%Chk=myfile
# Opt=Restart remainder of the original route
```

The **Opt=Restart** option always begins from the last completed point in the previous optimization.

CheckpointReporter

class openmm.app.checkpointreporter.**CheckpointReporter**(*file*, *reportInterval*, *writeState=False*)

CheckpointReporter saves periodic checkpoints of a simulation. The checkpoints will overwrite one another – only the last checkpoint will be saved in the file. Optionally you can saved serialized State objects instead of checkpoints. This is a more portable but less thorough way of recording the state of a simulation.

To use it, create a CheckpointReporter, then add it to the Simulation's list of reporters. To load a checkpoint file and continue a simulation, use the following recipe:

```
>>> simulation.loadCheckpoint('checkpoint.chk')
```

Reloading a saved State can be done like this:

```
>>> simulation.loadState('state.xml')
```

Notes: A checkpoint contains not only publicly visible data such as the particle positions and velocities, but also internal data such as the states of random number generators. Ideally, loading a checkpoint should restore the Context to an identical state to when it was written, such that continuing the simulation will produce an identical trajectory. This is not strictly guaranteed to be true, however, and should not be relied on. For most purposes, however, the internal state should be close enough to be reasonably considered equivalent.

PyTorch

Table of Contents

torch.utils.checkpoint

• NOTE

Checkpointing is implemented by rerunning a forward-pass segment for each checkpointed segment during backward propagation. This can cause persistent states like the RNG state to be more advanced than they would without checkpointing. By default, checkpointing includes logic to juggle the RNG state such that checkpointed passes making use of RNG (through dropout for example) have deterministic output as compared to non-checkpointed passes. The logic to stash and restore RNG states can incur a moderate performance hit depending on the runtime of checkpointed operations. If deterministic output compared to non-checkpointed passes is not required, supply `preserve_rng_state=False` to `checkpoint` or `checkpoint_sequential` to omit stashing and restoring the RNG state during each checkpoint.

The stashing logic saves and restores the RNG state for CPU and another device type (infer the device type from Tensor arguments excluding CPU tensors by `_infer_device_type`) to the `run_fn`. If there are multiple device, device state will only be saved for devices of a single device type, and the remaining devices will be ignored. Consequently, if any checkpointed functions involve randomness, this may result in incorrect gradients. (Note that if CUDA devices are among the devices detected, it will be prioritized; otherwise, the first device encountered will be selected.) If there are no CPU-tensors, the default device type state (default value is *cuda*, and it could be set to other device by `DefaultDeviceType`) will be saved and restored. However, the logic has no way to anticipate if the user will move Tensors to a new device within the `run_fn` itself. Therefore, if you move Tensors to a new device ("new" meaning not belonging to the set of [current device + devices of Tensor arguments]) within `run_fn`, deterministic output compared to non-checkpointed passes is never guaranteed.

torch.utils.checkpoint.checkpoint(*function*, **args*, *use_reentrant=None*, *context_fn=<function noop_context_fn>*, *determinism_check='default'*, *debug=False*, ***kwargs*) [SOURCE]

Checkpoint a model or part of the model.

Activation checkpointing is a technique that trades compute for memory. Instead of keeping tensors needed for backward alive until they are used in gradient computation during backward, forward

There are two different ways to checkpoint an **ANSYS FLUENT** simulation, depending upon how the simulation has been started.

1. **ANSYS FLUENT** running under LSF or SGE

ANSYS FLUENT is integrated with load management tools like LSF and SGE. These two tools allow you to checkpoint any job running under them. You can use the standard method provided by these tools to checkpoint the **ANSYS FLUENT** job.

For more information on using **ANSYS FLUENT** and SGE or LSF, go to the Load Management Documentation page on the User Services Center.

Not ideal...

2. Independently running **ANSYS FLUENT**

When not using tools such as LSF or SGE, a different checkpointing mechanism can be used when running an **ANSYS FLUENT** simulation. You can checkpoint an **ANSYS FLUENT** simulation while iterating/time-stepping, so that **ANSYS FLUENT** saves the case and data files and then continues the calculation, or so that **ANSYS FLUENT** saves the case and data files and then exits.

- Saving case and data files and continuing the calculation:

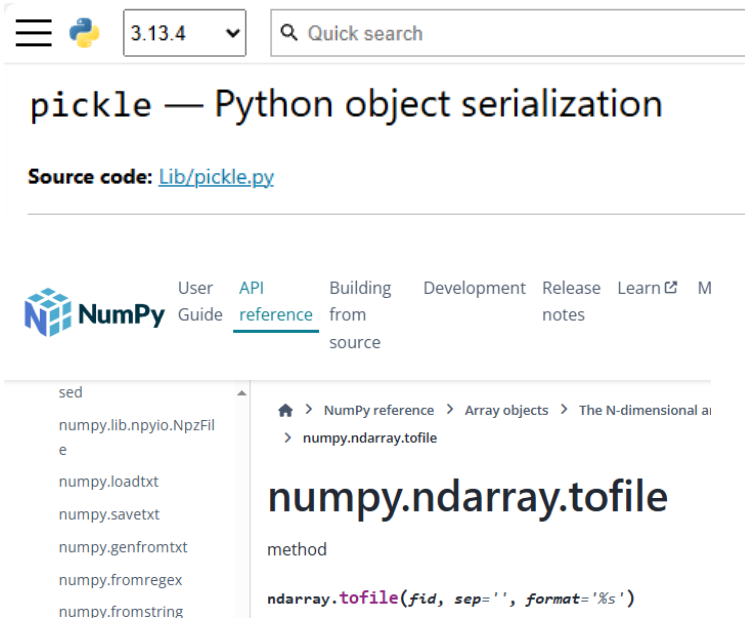
  On Linux/UNIX, create a file called `check-fluent`, i.e.,

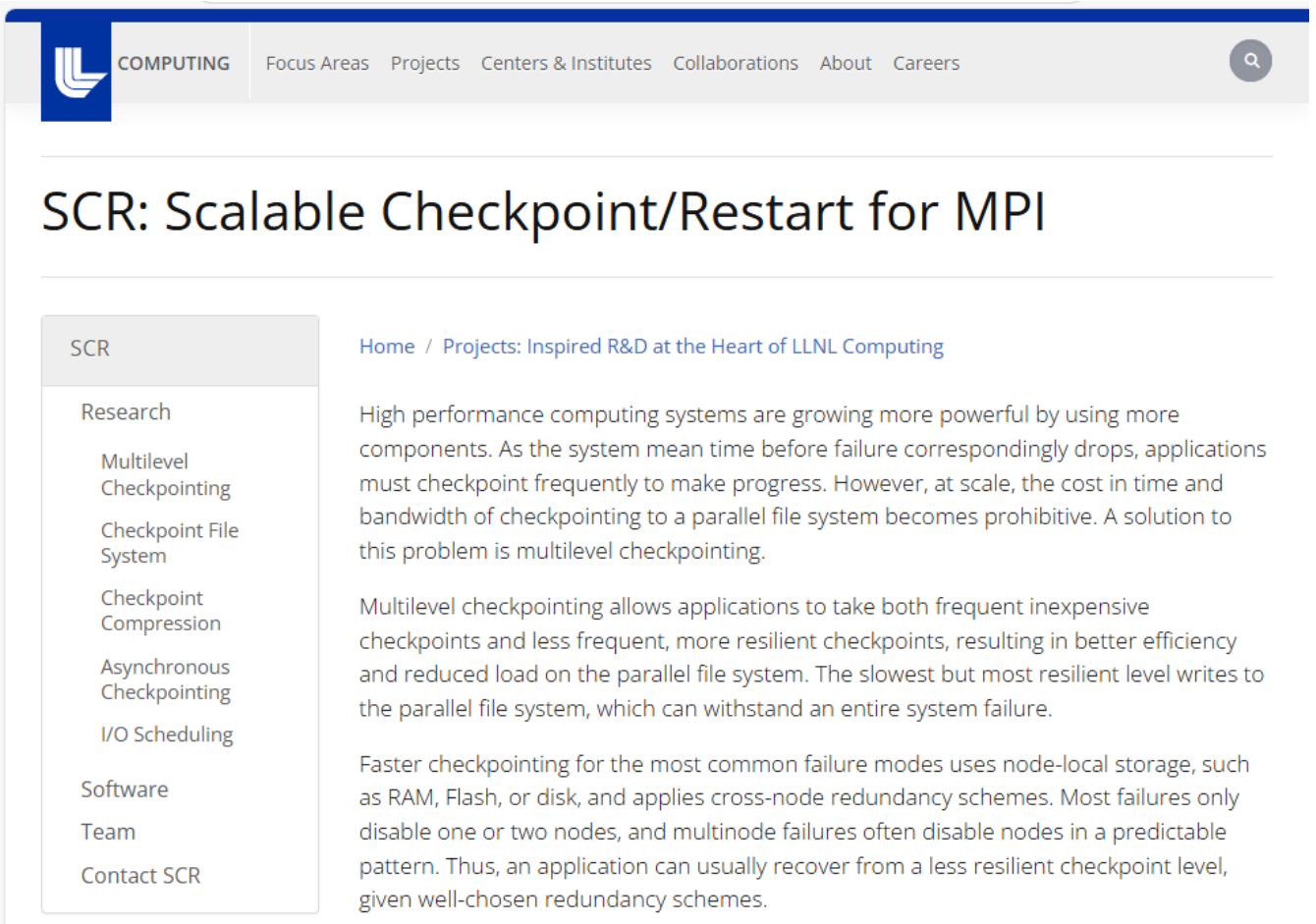
```
/tmp/check-fluent
```

  On Windows, create a file called `check-fluent.txt`, i.e.,

```
C:%USERPROFILE%\check-fluent.txt
```

If you develop/write the code yourself





But what happens if it isn't implemented?

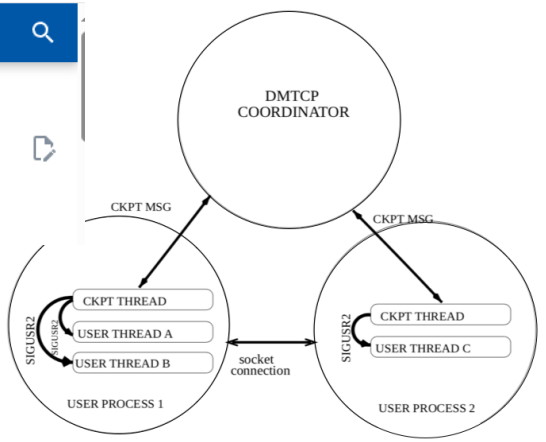
- Researchers using a code for an optimisation problem
- Pushing it a bit outside its use case
- Running it for several months
- No checkpointing!
- I didn't really feel like modifying the code
- Wouldn't it be nice if there was a tool to do it automatically?

Possible Candidates

- DMTCP – Distributed MultiThreaded CheckPointing
- MANA
- CRIU – Checkpoint and Restore in Userspace
- cuda_checkpoint

NERSC Documentation

DMTCP: Distributed MultiThreaded Checkpointing



MANA

MANA (MPI-Agnostic Network-Agnostic Checkpointing) is an implementation of transparent checkpointing for MPI. It is built as a plugin on top of the DMTCP checkpointing package. DMTCP stands for "Distributed MultiThreaded CheckPointing".

NVIDIA DEVELOPER Join

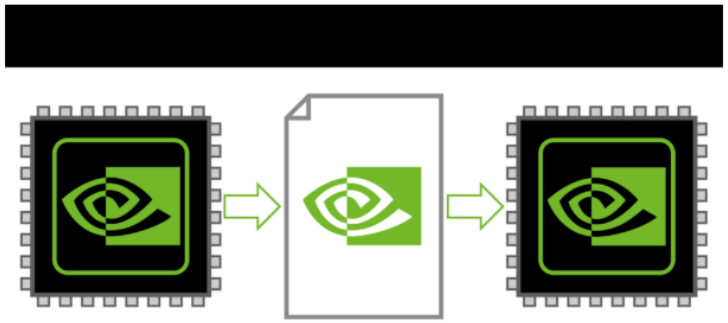
Technical Blog Subscribe

Simulation / Modeling / Design English

Checkpointing CUDA Applications with CRIU

Jul 02, 2024 +6 Like Discuss (1)

By Steven Gurfinkel



DMTCP

- Allow network data to drain into the memory of the process
- Make system calls to get state
- Copy all process's virtual memory into a file
- Use that to return the program to the same state the snapshot was taken at
- "Its like déjà vu, like I've been here before" - Alex the lion

https://dmtcp.sourceforge.io/dmtcp-mug-17.pdf

Demo

- Simple python example – infinite loop that adds 1 each second to a list of numbers, then prints the result
- Start coordinator
- Start process with dmtcp_launch – join the coordinator
- Checkpoint every 5 s

Conclusions

- Researchers should be trying to use checkpointing where possible and practical, especially for long-running simulations
- DMTCP provided a nice way of generic checkpointing functionality, where in-built functionality did not exist
- Worked for several python and C++ examples that I tried with it, and more importantly it worked for the use case I mentioned earlier
- I'd like to try some of the other tools described earlier, particularly MANA (for MPI)
- Thanks to Kiowa for drawing the image