Accelerating CASINO using GPUs

Ben Thorpe
Department of Physics, University of York

Neil Drummond
Department of Physics, University of Lancaster

PAX-HPC Meeting, Lancaster University

Monday 23rd April, 2024

"CASINÒ / CASINO" (Italian$\longrightarrow$English)

CASINO: quantum Monte Carlo program started by Richard Needs in the early '90s.

Main developers: **Richard Needs** (Cambridge), **Mike Towler** (TTI), **Neil Drummond** (Lancaster) and **Pablo López Ríos** (MPI Stuttgart).




casino m. **1** brothel, whorehouse **2** noise . . . **3** mess, ⟨*volg*⟩ cock-up

casinò m. casino

. . . so, for Italian speakers, our QMC code should really be called **CASINÒ**.

CASINO: Capabilities

- Variational Monte Carlo and diffusion Monte Carlo for 1D, 2D and 3D systems, with periodicity in 0, 1, 2 or 3 dimensions.

- *Ab initio* calculations for molecules and crystals **and** "exotic" or model systems: electron[–hole] gases, excitonic molecules, cold atomic gases, positronic systems, . . .

- Slater[–Jastrow[–backflow]] trial wave functions. The Slater part may consist of [spin-polarised] multiple determinants or [multiple] pairing (geminal) wave functions.
  - Basis functions: plane waves, blips, atom-centred Gaussians [with cusp corrections] and Slater functions. Numerical orbitals for atoms and molecular dimers.
  - Excited states from promotion, addition or subtraction of particles.
  - Wave-function optimisation by variance or energy minimisation.

- Periodic interactions with Ewald or model periodic Coulomb interactions.

- Various expectation values: total energy (and components), charge and spin density, pair correlation function, structure factor, one-particle and two-particle density matrices, electric dipole moment, momentum density, . . .

CASINO: Code Design Aims

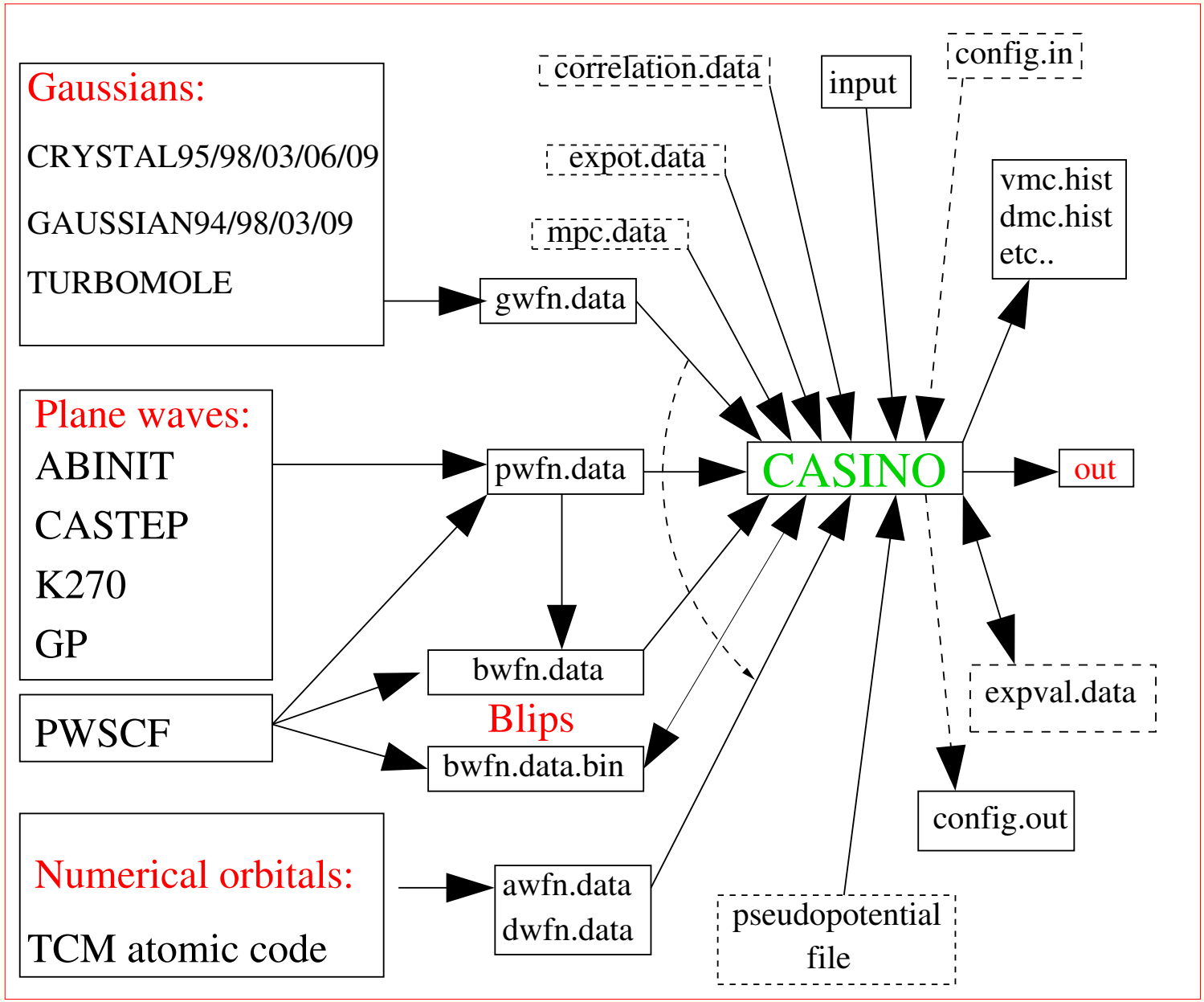
Generality VMC and DMC for systems of arbitrary size and geometry, including atoms, molecules, systems periodic in 1, 2 and 3 dimensions (polymers, slabs and crystalline solids), various electron and electron–hole phases, generalised quantum particles with arbitrary interactions (cold atoms, etc.). Choice of basis sets (plane waves, Gaussians, blips and Slaters) or grids for orbitals. Interfaces to a wide range of electronic structure codes for generating trial wave functions.

Portability Strict Fortran 2003. Apart from MPI no external libraries are needed; BLAS/LAPACK optional. Automatic, customisable compilation and setup.

Ease of use Shell-script automation. Full documentation: internal help system, comprehensive manual and interactive website, including pseudopotential library: `https://vallico.net/casinoqmc`. Wide range of examples. Discussion forum: `https://vallico.net/casino-forum`.

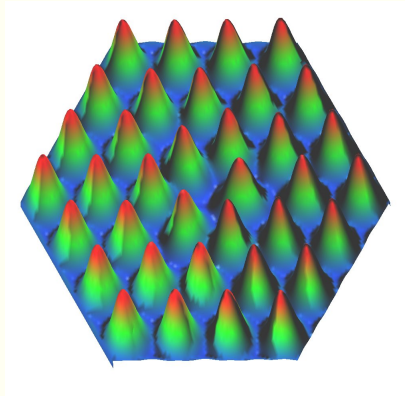
Speed and memory efficiency Efficient algorithms optimised for speed. Efficiently parallelised: MPI over (near-)independent random walks, OpenMP over particles. Shared memory between MPI processes. **OpenACC for offloading to accelerators.**

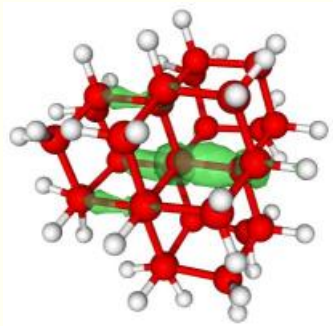
Interfaces to Other Codes



A Cornucopia of Recent Applications of CASINO

- QMC methods have been used to study (amongst other things):
  - 3D, 2D and 1D **electron gases**: ground-state energy, phase diagram, magnetic susceptibility, distribution functions, quasiparticle effective mass;
  - Ultra-cold bosonic and fermionic **atomic gases**;
  - Band structures of **crystalline solids**;
  - Optical band gaps of **nanocrystals**;
  - Defects in **semiconductors**;
  - Phase diagrams and equations of state of **materials at high pressure**;
  - Dispersion interactions between **low-dimensional materials**;
  - Binding of **molecules** and their excitation energies;
  - **Positronic** molecules and crystalline solids;
  - Ground-state properties without the Born–Oppenheimer approximation (nuclei treated as quantum particles).





Quantum Monte Carlo

- Expectation value of Hamiltonian:

$$\frac{\langle\Psi|\hat{H}|\Psi\rangle}{\langle\Psi|\Psi\rangle} = \frac{\int \Psi^* \hat{H}\Psi \, d\mathbf{R}}{\int |\Psi|^2 \, d\mathbf{R}} = \frac{\int |\Psi|^2 \frac{\hat{H}\Psi}{\Psi} \, d\mathbf{R}}{\int |\Psi|^2 \, d\mathbf{R}} = \langle E_{\mathrm{L}} \rangle_{|\Psi|^2} ,$$

  where the *local energy* is

$$E_{\mathrm{L}} = \frac{\hat{H}\Psi}{\Psi} = \sum_i -\frac{1}{2m_i}\frac{\nabla_i^2\Psi}{\Psi} + U.$$

- **VMC**: use the Metropolis algorithm to generate electron coordinates distributed as $|\Psi|^2$ and average the local energies to obtain an estimate of the energy.
  - Variance of the local energy reduces as trial wave function improves.

- **DMC**: simulate drift, diffusion and branching/dying processes governed by imaginary-time Schrödinger equation to project out the ground-state component of Ψ.
  - Fermionic antisymmetry is maintained by fixing the nodes of the wave function.

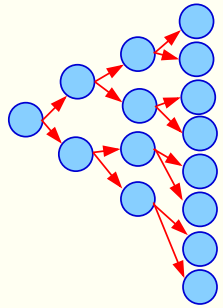
Diffusion Quantum Monte Carlo

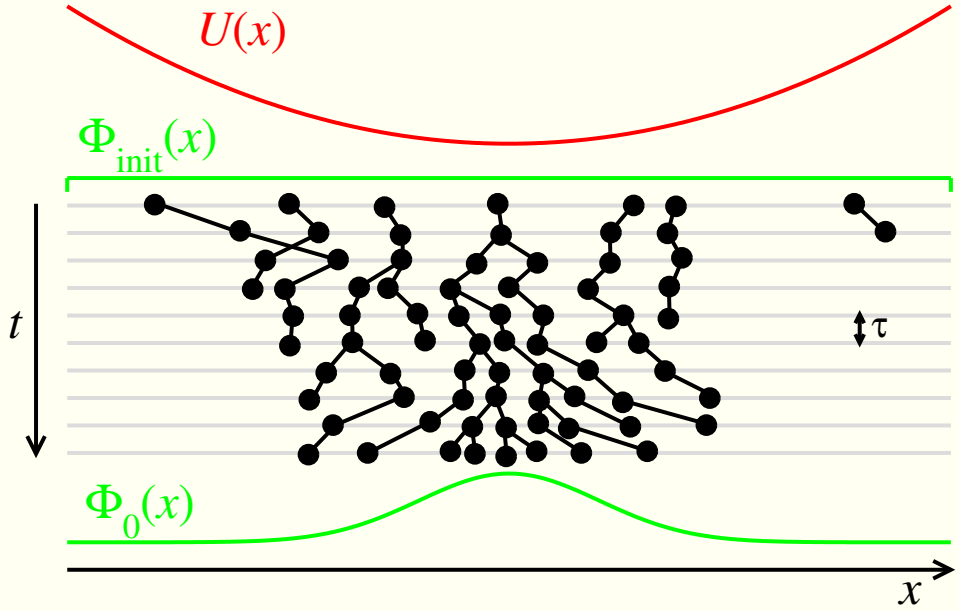
$$\sum_i -\frac{1}{2m_i}\nabla_i^2\Phi + U\Phi = -\frac{\partial\Phi}{\partial t}$$

$3N$-dimensional diffusion equation

$$\sum_i -\frac{1}{2m_i}\nabla_i^2\Phi + U\Phi = -\frac{\partial\Phi}{\partial t}$$

Rate equation





Slater–Jastrow Wave Functions

- Most QMC calculations use *Slater–Jastrow* trial wave functions:

$$\Psi(\mathbf{R}) = \exp[J(\mathbf{R})] \sum_n c_n D_n^{\uparrow}(\mathbf{R}) D_n^{\downarrow}(\mathbf{R}),$$

  where $D^{\uparrow}$ and $D^{\downarrow}$ are **Slater determinants** for spin-up and down electrons, and $\exp(J)$ is a **Jastrow factor**.

- Each Slater determinant is of the form

$$D^{\uparrow}(\mathbf{R}) = \begin{vmatrix} \psi_1^{\uparrow}(\mathbf{r}_1) & \cdots & \psi_{N_\uparrow}^{\uparrow}(\mathbf{r}_1) \\ \vdots & & \vdots \\ \psi_1^{\uparrow}(\mathbf{r}_{N_\uparrow}) & \cdots & \psi_{N_\uparrow}^{\uparrow}(\mathbf{r}_{N_\uparrow}) \end{vmatrix},$$

  and similarly for spin-down determinants.

  – Orbitals $\{\psi_i^{\sigma}\}$ are usually generated in either DFT or HF calculations.
  – Updating the orbitals contributes to the $\mathcal{O}(N^2)$ cost per time step; evaluating the determinant contributes a (negligible) $\mathcal{O}(\epsilon N^3)$ cost.

Jastrow Factor

- Jastrow factor $\exp(J)$ is an explicit function of interparticle distances, allowing compact parameterisation of correlation.
  - Slater wave function has the required exchange antisymmetry and the required space group symmetry, so J should be exchange symmetric and transform as the trivial representation of the space group.
  - J contains free parameters, to be determined by an optimisation method.
  - J must be twice differentiable where the potential is finite.

- In CASINO the Jastrow exponent J is a sum of:
  - Truncated polynomials in e-e distance, satisfying the Kato cusp conditions $[u(r_{ij})]$;
  - Truncated polynomials in e-n distance, satisfying the Kato cusp conditions $[\chi(r_{iI})]$;
  - Truncated polynomials in e-e-n distances $[f(r_{ij}, r_{iI}, r_{jI})]$.
  - Plane-wave expansions in e-e separation $[p(\mathbf{r}_{ij})]$.
  - Plane-wave expansions in e position $[q(\mathbf{r}_i)]$.
  - Truncated three-body polynomials $[H(r_{ij}, r_{ik}, r_{jk})]$.

- The two-body terms contribute to the $\mathcal{O}(N^2)$ scaling per time step.

Two-Body Plane-Wave Jastrow Term

- The plane wave term in the Jastrow exponent J is $\sum_{i>j} p(\mathbf{r}_{ij})$, where

$$p(\mathbf{r}_{ij}) = \sum_A a_A \sum_{\mathbf{G}_A^+} \cos(\mathbf{G}_A \cdot \mathbf{r}_{ij})$$

  and the $\{\mathbf{G}_A\}$ are reciprocal lattice points of the simulation cell belonging to the Ath star (only one out of each $\pm\mathbf{G}_A$ pair) and the $\{a_A\}$ are optimisable parameters.

- For $\mathbf{G} = n_1\mathbf{b}_1 + n_2\mathbf{b}_2 + n_3\mathbf{b}_3$, where $\{\mathbf{b}_i\}$ are supercell reciprocal lattice vectors,

$$\cos(\mathbf{G}_A \cdot \mathbf{r}_{ij}) = \mathrm{Re}\left[\left(e^{i\mathbf{b}_1\cdot\mathbf{r}_{ij}}\right)^{n_1}\left(e^{i\mathbf{b}_2\cdot\mathbf{r}_{ij}}\right)^{n_2}\left(e^{i\mathbf{b}_3\cdot\mathbf{r}_{ij}}\right)^{n_3}\right],$$

  so only three complex exponentials are required to compute all the required cosines. The powers of these exponentials are computed and buffered.

- Typical number of stars: 3–15 ($\sim$10–$\sim$100 reciprocal lattice points).

- To-do: allow a B-spline re-representation of the p term in VMC and DMC.

Ewald Interactions

- In a molecular system, the pairwise Coulomb potential energy in $U(\mathbf{R})$ is just a pairwise sum of $1/r_{ij}$ interactions.

- In a periodic solid we use **Ewald's method** to calculate the interparticle potential:
  - The pairwise solution to Poisson's equation is evaluated using the periodic solution to Poisson's equation; this in turn is evaluated using rapidly convergent sums over real and reciprocal lattice vectors.

- These calculations must be performed every time a particle is moved and hence they contribute to the $\mathcal{O}(N^2)$ scaling per time step.

- In rare use cases (i) the short-range electron–electron interaction may be replaced by a pseudopotential and (ii) the electric field (gradient of potential) may be required to evaluate core-polarisation terms in ionic pseudopotentials.

Tasks in CASINO to be Targeted for Offloading to GPUs

The following $\mathcal{O}(N^2 + \epsilon N^3)$ tasks are likely targets for offloading that could be of practical benefit:

1. Evaluating Ewald interactions ($\sim$10% of total time in periodic systems).
2. Evaluating two-body Jastrow terms, especially p (5–70% of run time).
3. Evaluating relative positions of closest images of pairs of particles ($<$5% of run time).
4. Evaluating single-particle orbitals (5–50% of run time, depending on basis set).
5. Updating Slater determinants after electron moves ($<$5% of run time).

Before the start of the PAX-HPC project, Neil had implemented **OpenACC** offloading of items 1–3, achieving a GPU speedup of 10–12% on item 1 on the Lancaster High-End Computing Cluster and Bede, and a large slowdown on items 2 and 3. . .

. . . Over to Ben for the current status of items 1 and 2!

Ewald interactions

Disapointing start but I'm sure it gets better, right?

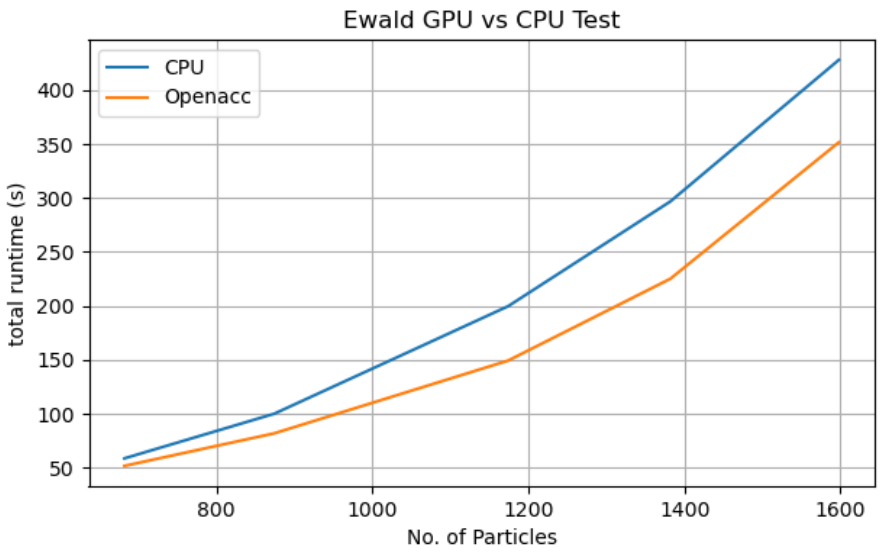


Figure: Runtime Vs Num. of particles for 3D HEG. All runs were on performed Bede using single core of 32 core Power9 CPU @ 2.7GHz with an Nvidia V100 GPU

- Initial attempt
- 3D heterogenous electron gas (HEG)
- 1 big loop over all particles.
- Used Openacc to split the loop across the GPU.
- Performance was faster but, underwhelming.

Do loop = 1, n

- **Correct for Pseudopotential (if used)**
- **Sum over Energies in Real-space**
- **Sum over Energies in Reciprical-space**
- **Caculate Total Energy**
- **Calcuate E-field (if needed)**

EndDo

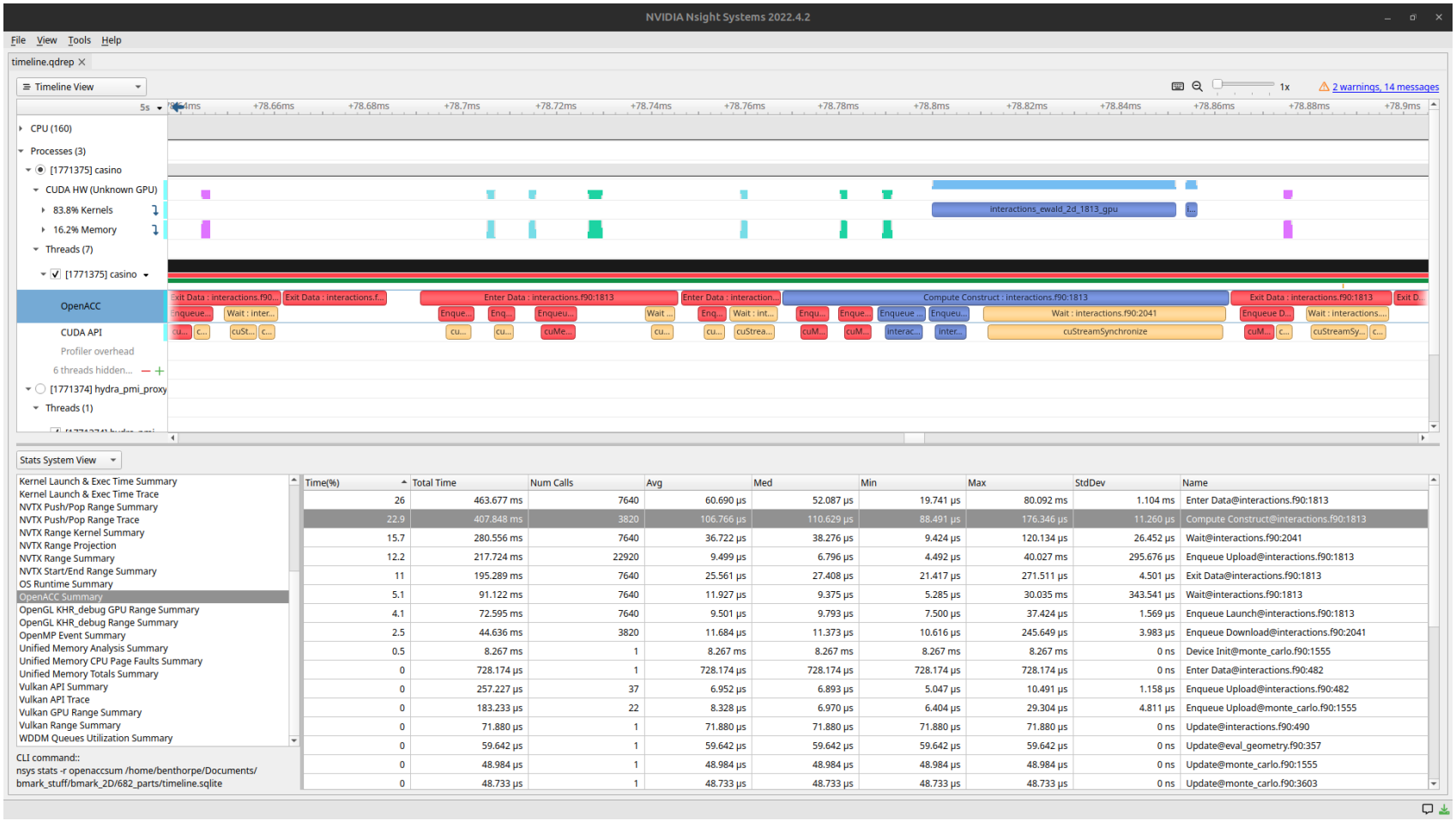
Enter, Nsight toolkit

Save us Nvidia, or at least point us in the right direction.

- Nvidia GPU profiling/development tools
- **Nsight Systems** Analyse what the code is doing at a given time.
- **Nsight Compute** Allows for more In-depth analysis.
- They are very useful but not prettiest.

Nsight Systems

Nice, screenshot. If only they could read it.

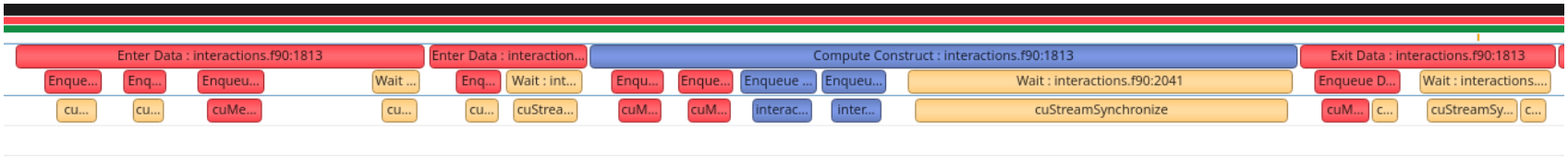


Time(%)	Total Time	Num Calls	Avg	Med	Min	Max	StdDev	Name
26	463.677 ms	7640	60.690 µs	52.087 µs	19.741 µs	80.092 ms	1.104 ms	Enter Data@interactions.f90:1813
22.9	407.848 ms	3820	106.766 µs	110.629 µs	88.491 µs	176.346 µs	11.260 µs	Compute Construct@interactions.f90:1813
15.7	280.556 ms	7640	36.722 µs	38.276 µs	9.424 µs	120.134 µs	26.452 µs	Wait@interactions.f90:2041
12.2	217.724 ms	22920	9.499 µs	6.796 µs	4.492 µs	40.027 ms	295.676 µs	Enqueue Upload@interactions.f90:1813
11	195.289 ms	7640	25.561 µs	27.408 µs	21.417 µs	271.511 µs	4.501 µs	Exit Data@interactions.f90:1813
5.1	91.122 ms	7640	11.927 µs	9.375 µs	5.285 µs	30.035 ms	343.541 µs	Wait@interactions.f90:1813
4.1	72.595 ms	7640	9.501 µs	9.793 µs	7.500 µs	37.424 µs	1.569 µs	Enqueue Launch@interactions.f90:1813
2.5	44.636 ms	3820	11.684 µs	11.373 µs	10.616 µs	245.649 µs	3.983 µs	Enqueue Download@interactions.f90:2041
0.5	8.267 ms	1	8.267 ms	8.267 ms	8.267 ms	8.267 ms	0 ns	Device Init@monte_carlo.f90:1555
0	728.174 µs	1	728.174 µs	728.174 µs	728.174 µs	728.174 µs	0 ns	Enter Data@interactions.f90:482
0	257.227 µs	37	6.952 µs	6.893 µs	5.047 µs	10.491 µs	1.158 µs	Enqueue Upload@interactions.f90:482
0	183.233 µs	22	8.328 µs	6.970 µs	6.404 µs	29.304 µs	4.811 µs	Enqueue Upload@monte_carlo.f90:1555
0	71.880 µs	1	71.880 µs	71.880 µs	71.880 µs	71.880 µs	0 ns	Update@interactions.f90:490
0	59.642 µs	1	59.642 µs	59.642 µs	59.642 µs	59.642 µs	0 ns	Update@eval_geometry.f90:357
0	48.984 µs	1	48.984 µs	48.984 µs	48.984 µs	48.984 µs	0 ns	Update@monte_carlo.f90:1555
0	48.733 µs	1	48.733 µs	48.733 µs	48.733 µs	48.733 µs	0 ns	Update@monte_carlo.f90:3603

Don't worry. This is here for illustration. I don't expect you to be able to read this!!

Nsight Systems

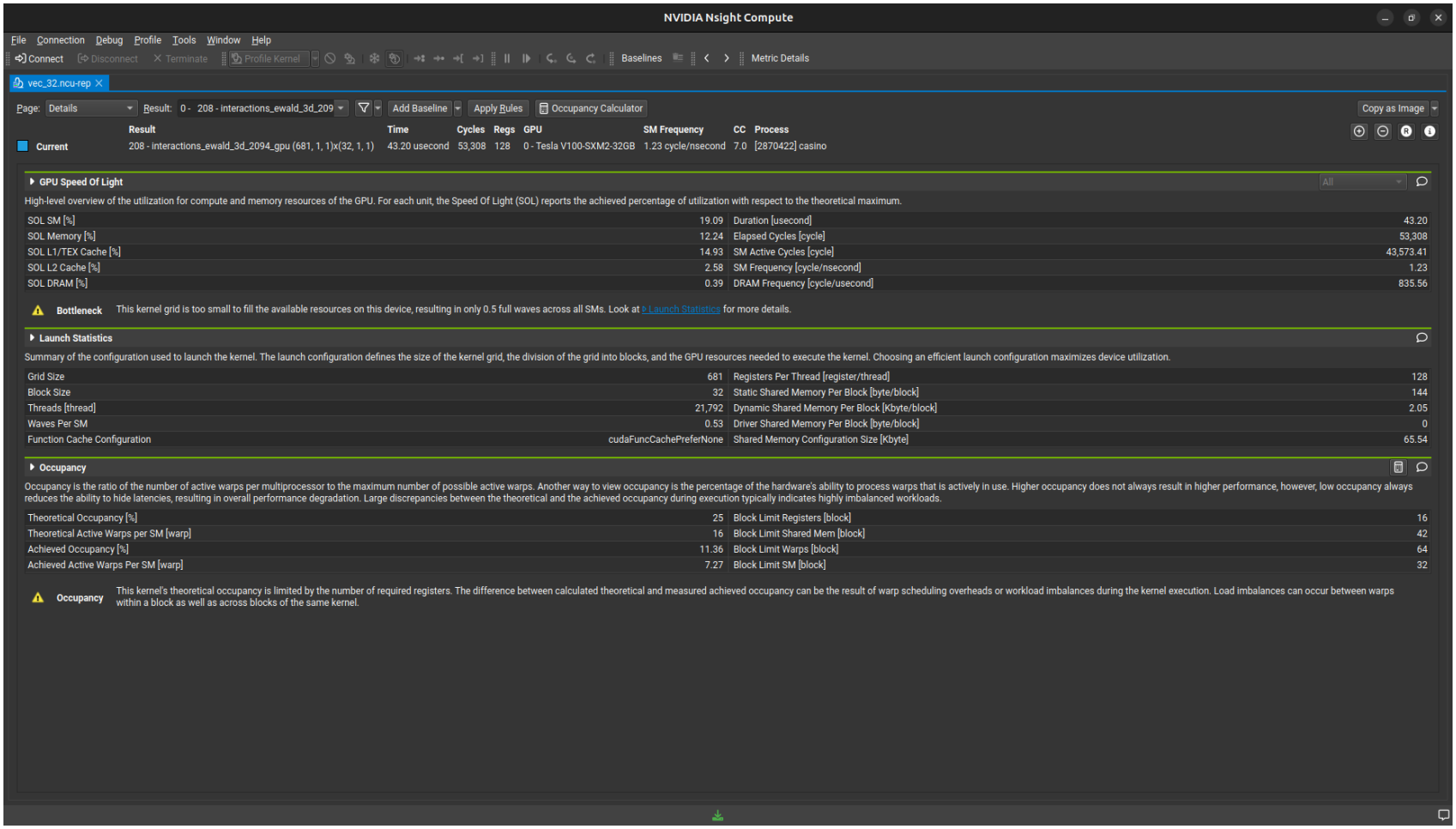
So what is going on?



- We're spending a good chunk of time doing very little
- The Data (Red) and Wait (orange) regions are both bigger than we'd like
- The waiting looks suspicious
- We will need to dig a bit deeper

Nsight Compute

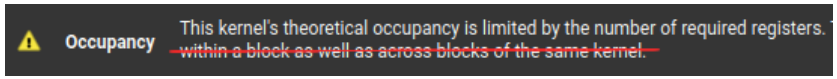
And you thought Nsight Systems was bad.



Again this is far to dense with information. So Don't worry, I don't expect you to be able to read this!!

Nsight Compute

I wonder who the murder is. Probably the buttler?



- The GPU hardware was limiting the number of concurrent threads.
- This limits the actual performance compared to what the device is capable of.
- This is not great as for GPUs threads are essentially everything
- The prime culprit was register pressure.

Figure: Hercule Poirot ©ITV Studios: used under fair use. Oh and fyi I know he's Belgian really.

What are Threads and why are they important?

Threads are independent instructions that are handled by a CPU/GPU. CPUs are faster per thread but can only handle a few at a time. GPUs by contast can handle 1000's of threads but are much slower per thread.

So what is Register pressure?

There's to many registers, I can't take it anymore!

- All GPUs (and CPUs) have a very limited amount of super fast memory (10's Kb).
- This is reserved for storing small variables that are frequently used or thread specific.
- Each variable is allocated a number of registers for storage.
- All other variables are stored in global memory, which is shared by all SM's but is substantially slower.

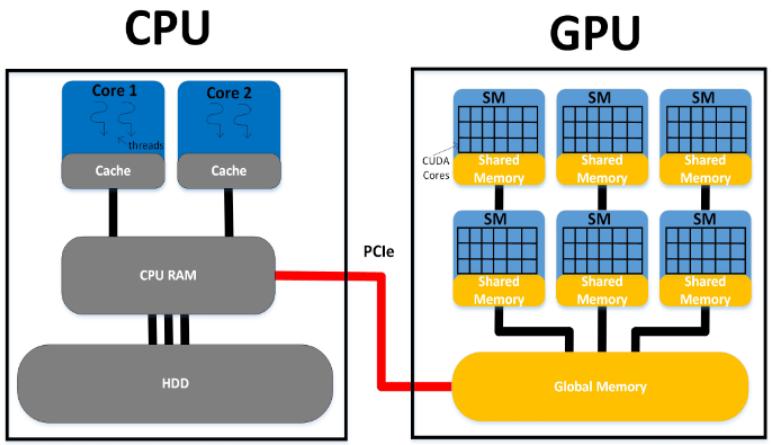


Figure: Diagram of memory layout for CPU and GPU.[1]

So what is Register pressure?

There's to many registers, I can't take it anymore!

- We are using lots of private variables within our Do loop.
- These are being stored for each thread in shared memory.
- Each SM only has so much room to store variables for all the threads.
- This limits the number of threads each SM can run, as threads compete for room in the shared memory.

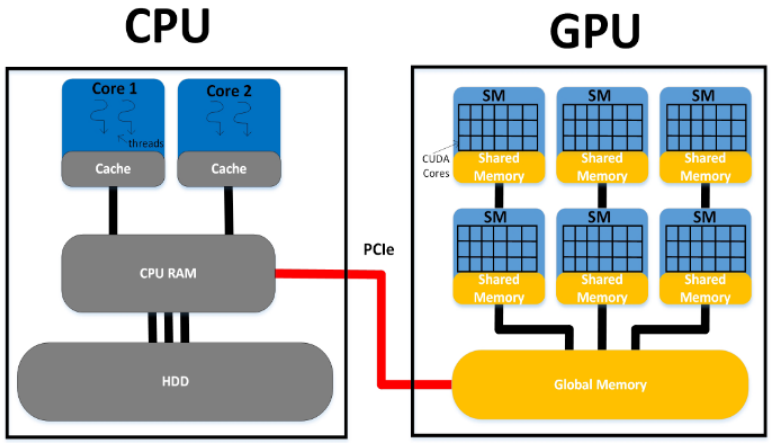
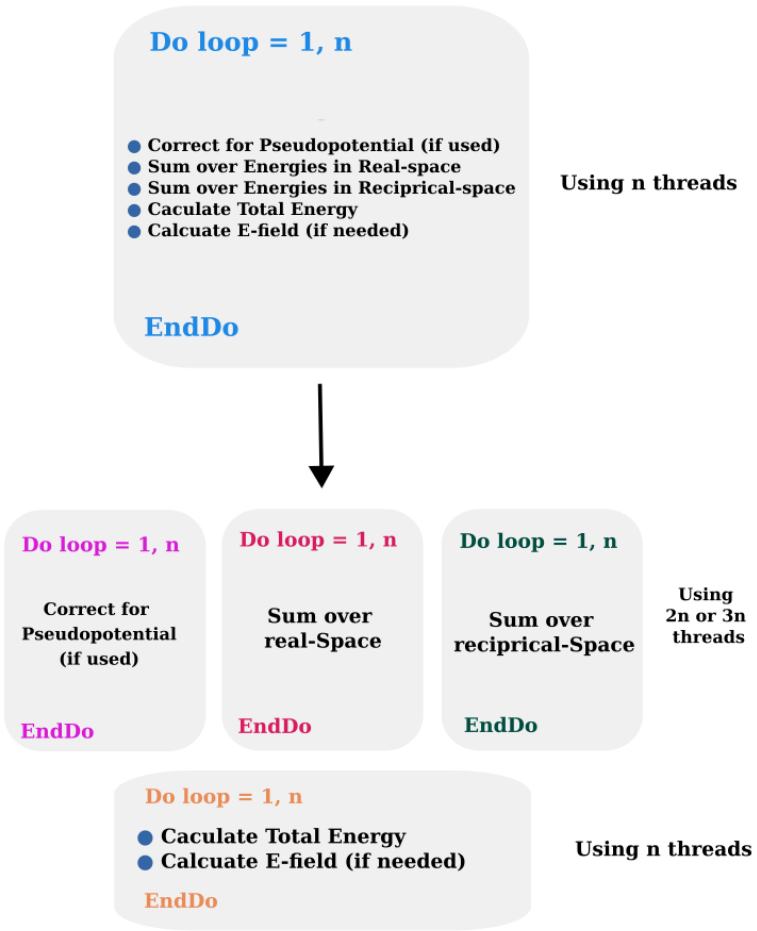


Figure: Diagram of memory layout for CPU and GPU.[1]

Back to Ewald Interactions

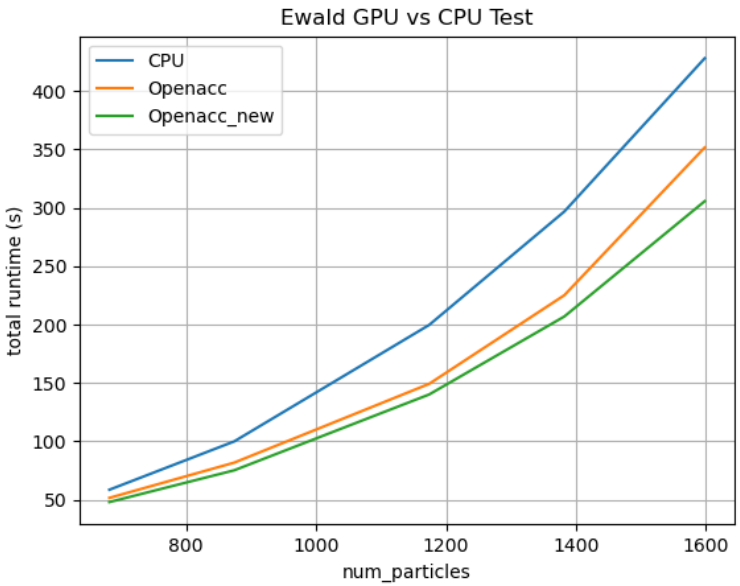
So what have we learned?

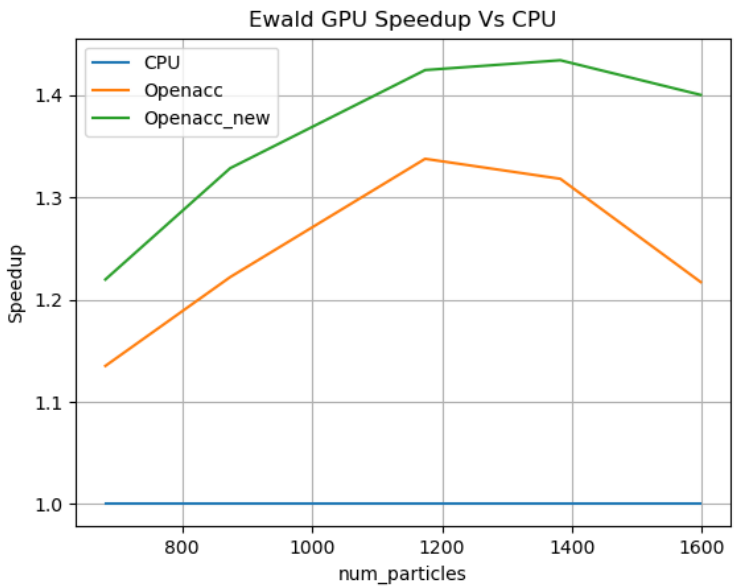
- Main Point: We need to reduce the number of private variables in the loop.
- We'll do this by breaking the loop up into smaller parts.
- Also adding more threads couldn't hurt.
- We however, need to keep in mind not to slow down the CPU version.



Results: Ewald Interactions

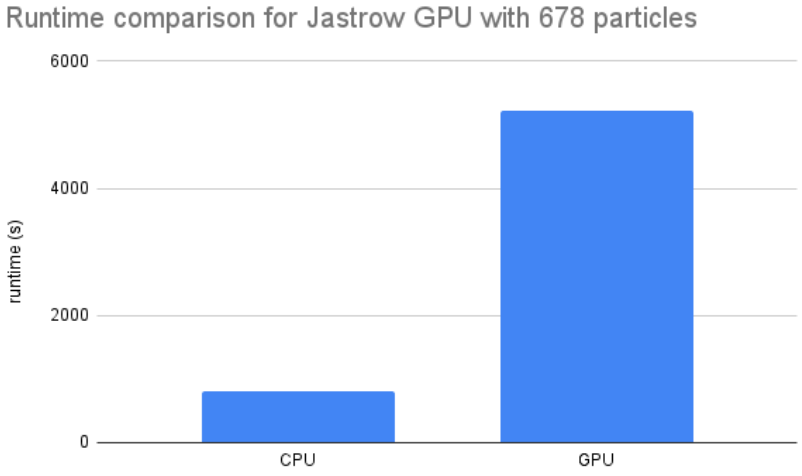
Hardly earth shattering but it's progress

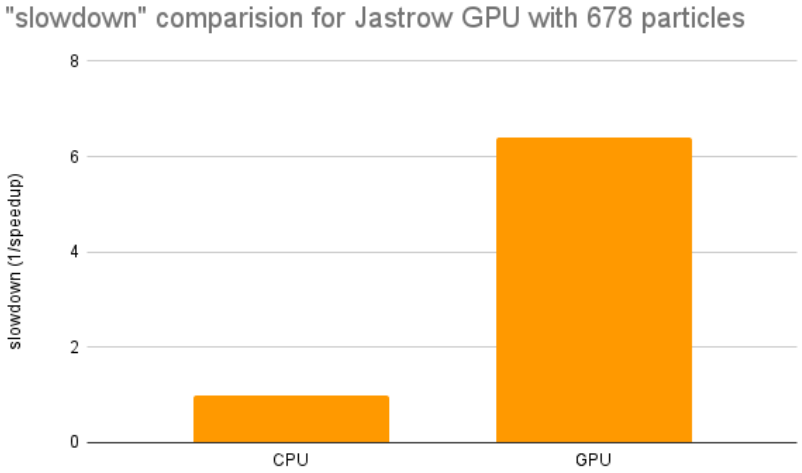




Jastrow Factor

Oh dear





- Initial attempt
- 3D heterogenous electron gas (HEG)
- Jastrow P and U terms each have 1 big loop over all particles.
- Split up over the GPU threads
- Performance was, non-existent (around 6 times slower).

Back to Nsight toolkit

So it really was the P-term, I'd never have guessed.

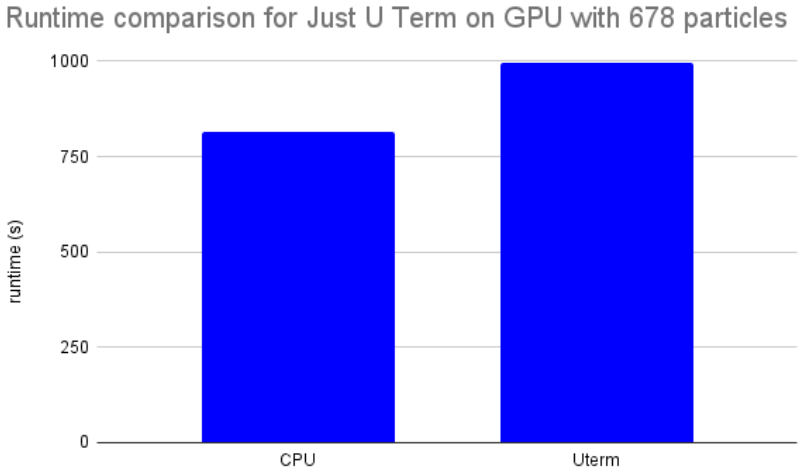
- The main problem seems to lie with the P-term.
- It appears to be only running on a single thread.
- The Uterm, however, is not completely innocent, as it to is running slowly.

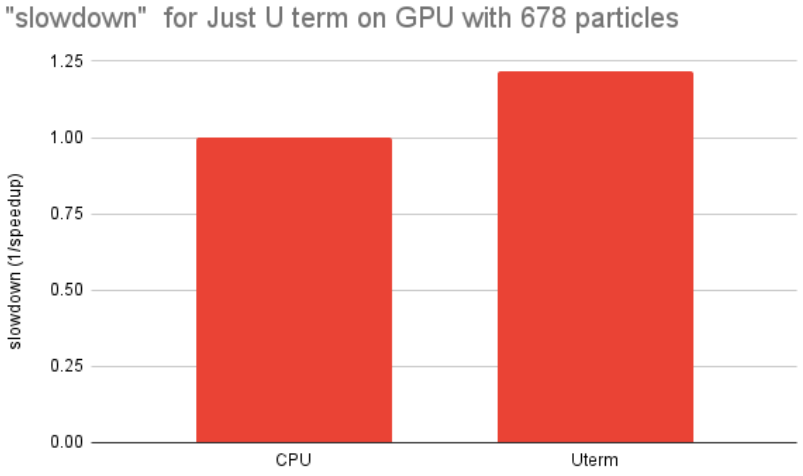
Figure: Hercule Poirot ©ITV Studios: I tried to warn him this Joke wasn't funny but he insited.

Fixing the U-term

Well I guess you can't expect miracles.

- We are doing to much work per thread.
- Unfortunately, we also can't break up the loop any further.
- There is scope to calculate U in the background on the CPU, whilst P is running on the GPU.
- Otherwise I'm afraid this is a dead end.





Shared memory Issues

Why can't gpu threads just learn to get along?

- The issue affecting the P-term is similar to Ewald, only much worse.
- It's not the number of private variables however, it's their size.
- Each thread needs its own copy of the 3 arrays storing $Exp_B(n)$.
- This was quickly filling the shared memory and meant the SM's could only run 1 thread at a time.

Do j = 1,netot

For j ≠ k:

Do n = 1, max_gvec

$$Exp_{B1}(n) = \exp\left(i\vec{b1} \cdot \vec{r}_{kj}\right)^n$$
$$Exp_{B2}(n) = \exp\left(i\vec{b2} \cdot \vec{r}_{kj}\right)^n$$
$$Exp_{B3}(n) = \exp\left(i\vec{b3} \cdot \vec{r}_{kj}\right)^n$$

EndDo

Using up to netot-1 threads

* FD_p and SD_p are only calculated when required

$$Val_p(j) = \sum_{n=1}^{max_{gvec}} -|\vec{G(n)}|^2 * \mathrm{Re}\left(Exp_{B1}(n) * Exp_{B2}(n) * Exp_{B3}(n)\right)$$

* $$FD_p(j) = \sum_{n=1}^{max_{gvec}} |\vec{G(n)}|^2 * \mathrm{Im}\left(Exp_{B1}(n) * Exp_{B2}(n) * Exp_{B3}(n)\right)$$

* $$SD_p(j) = \sum_{n=1}^{max_{gvec}} -|\vec{G(n)}|^2 * \mathrm{Re}\left(Exp_{B1}(n) * Exp_{B2}(n) * Exp_{B3}(n)\right) + Expcoef(n)$$

EndDo

ACC Reduction: Sum Val_p over j

* ACC Reduction: Sum FD_p over j

* ACC Reduction: Sum SD_p over j

Shared memory Issues

Why can't gpu threads just learn to get along?

The solution was to 3 $max_{gvec} \times netot$ arrays in global memory. We can then give each thread 1 row, corresponding to each particle j in the loop(s).

Do j = 1, netot
For j ≠ k:
Do n = 1, max_gvec
$Exp_{B1}(n,j) = \exp\left(ib\vec{1} \cdot \vec{r}_{kj}\right)^n$
EndDo
EndDo

Do j = 1, netot
For j ≠ k:
Do n = 1, max_gvec
$Exp_{B2}(n,j) = \exp\left(ib\vec{2} \cdot \vec{r}_{kj}\right)^n$
EndDo
EndDo

Do j = 1, netot
For j ≠ k:
Do n = 1, max_gvec
$Exp_{B3}(n,j) = \exp\left(ib\vec{3} \cdot \vec{r}_{kj}\right)^n$
EndDo
EndDo

Using up to netot-1, 2*(netot-1) or 3*(netot-1) threads for 1D, 2D or 3D

Do j = 1, netot
For j ≠ k:
$\mathrm{Val}_p(j) = \sum_{n=1}^{max_{gvec}} -|G(\vec{n})|^2 * \mathrm{Re}\left(Exp_{B1}(n,j) * Exp_{B2}(n,j) * Exp_{B3}(n,j)\right)$
EndDo

ACC Reduction:
Sum Val_p over j

Do j = 1, netot
For j ≠ k:
$\mathrm{FD}_p(j) = \sum_{n=1}^{max_{gvec}} |G(\vec{n})|^2 * \mathrm{Im}\left(Exp_{B1}(n,j) * Exp_{B2}(n,j) * Exp_{B3}(n,j)\right)$
EndDo

* ACC Reduction:
Sum FD_p over j

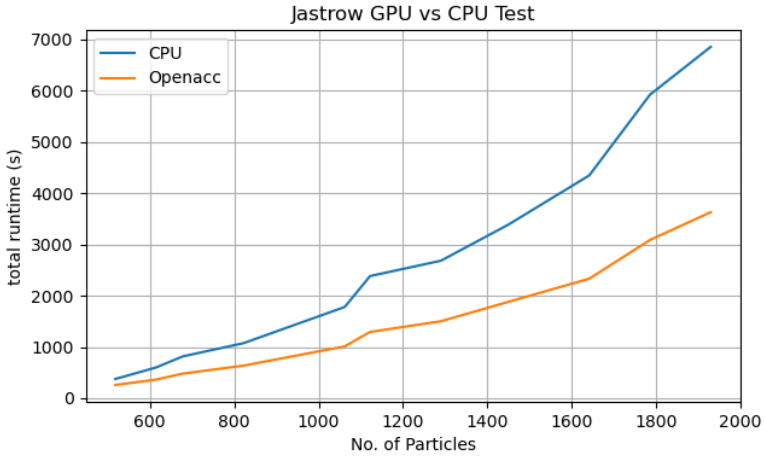
Do j = 1, netot
For j ≠ k:
$SD_p(j) = \sum_{n=1}^{max_{gvec}} -|G(\vec{n})|^2 * \mathrm{Re}\left(Exp_{B1}(n,j) * Exp_{B2}(n,j) * Exp_{B3}(n,j)\right) + Expcoef(n)$
EndDo

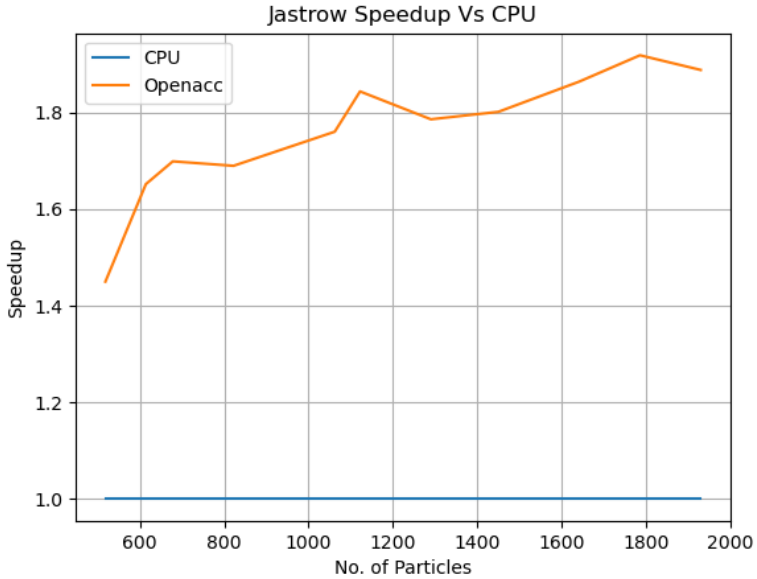
* ACC Reduction:
Sum SD_p over j

Results: P-term

Now were talking.

- We see a speed-up of between 1.45 and 1.89 times.
- This is very impressive given the P-term is $\approx 50\%$ of the runtime.
- Max theoretical speed-up is 2.0.
- It also appears to be steadily increasing with number of particles.
- So overall I'm happy

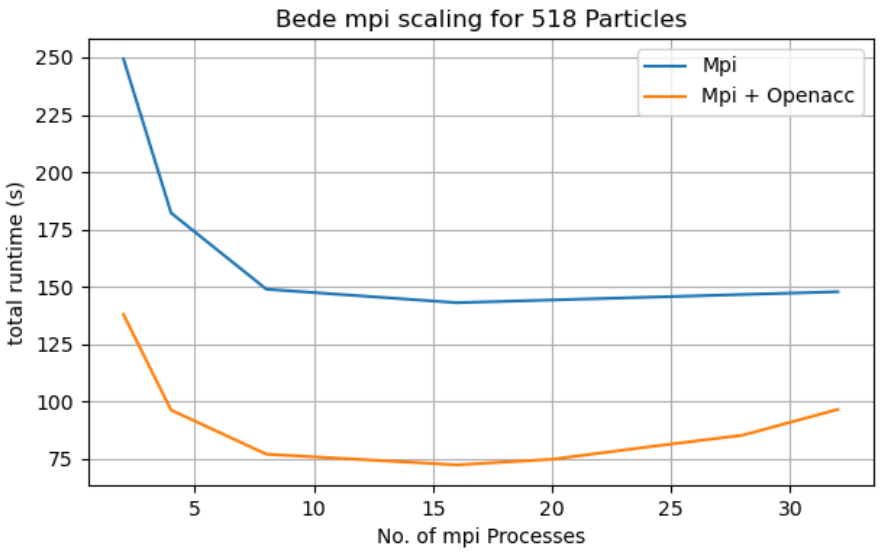




A quick word on MPI performance

Ah. . . yes, we may have issues there.

- The problem is Openacc can only "see" 1 GPU by default.
- We get good speed-up for 2–4 processes.
- Then the processes start competing for resources.
- Short version is we will need to look into multi-gpu



Conclusion

Wait, there really was a point to all this?

- The GPU port is finally starting to make some headway.
- We see a small speed-up for Ewald calculations
- We also see a more significant speed-up for the Two-body Jastrow P-term.
- Work is needed on combining OpenMp and Openacc
- MPI performance still needs some work.

Acknowledgments

Don't worry you can switch off now.

- Phil Hasnip & Matt Smith
- Excalibur Project, specifically Pax HPC, for the Funding
- N8 Research Partnership, for use of Bede

Bibliography

[1] Muaaz Gul Awan, Taban Eslami, and Fahad Saeed. "GPU-DAEMON: GPU algorithm design, data management and optimization template for array based big omics data". In: *Computers in Biology and Medicine* 101 (2018), pp. 163–173. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2018.08.015. URL: https://www.sciencedirect.com/science/article/pii/S001048251830235X.