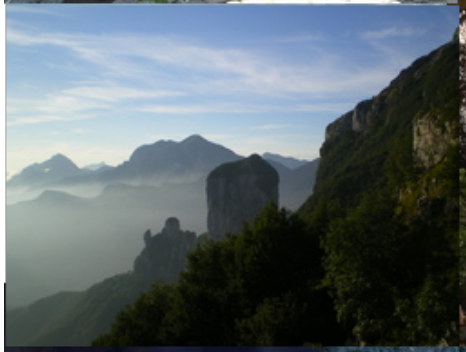
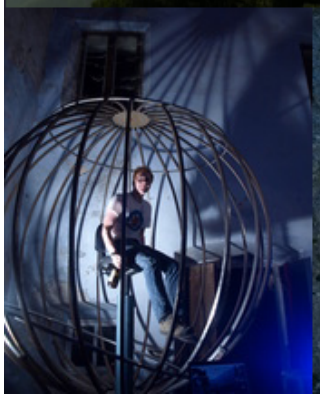


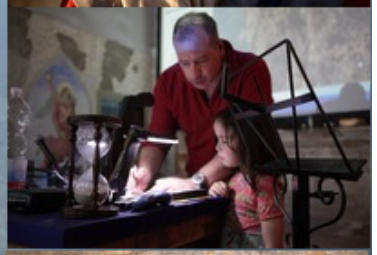


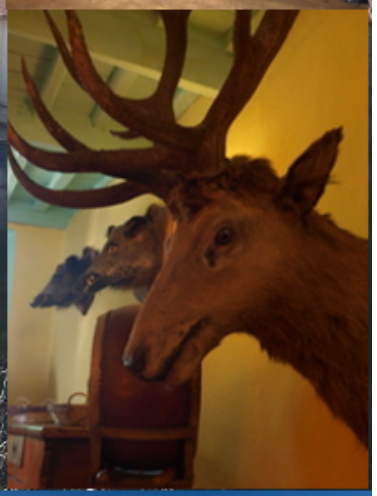












The Towler Institute

2014 International Summer School

Quantum Monte Carlo and the CASINO program IX

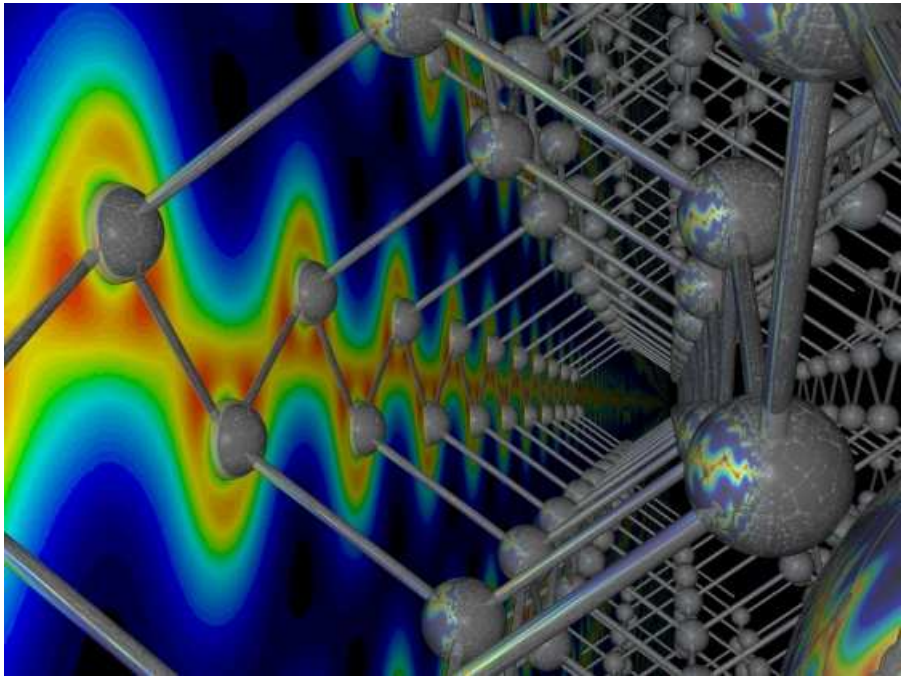
Vallico Sotto, Tuscany, Italy 3rd - 10th August 2014

vallico.net/tti/tti.html

email : mdt26 at cam.ac.uk

QUANTUM MONTE CARLO

Three QMC scaling problems



Mike Towler

TCM Group, Cavendish Laboratory, University of Cambridge

QMC web page: `vallico.net/casinoqmc`

Email: `mdt26@cam.ac.uk`

Three scaling problems

Many atoms

- Expected QMC scaling of CPU time with system size normally stated to be N^3.

Is this right, and can we do better than this?

Many protons

- Theoretical arguments suggest CPU time for all-electron DMC scales with atomic number Z as $Z^{5.5} - Z^{6.5}$. Stability problems for heavy atoms. Essentially no all-electron calculations published for atoms heavier than neon ($Z = 10$).

What is the scaling in practice? What is the heaviest atom for which it is feasible to do a DMC calculation? Can we solve the stability problem? Can timestep errors be brought under control? Are all-electron calculations more accurate than pseudopotential calculations?

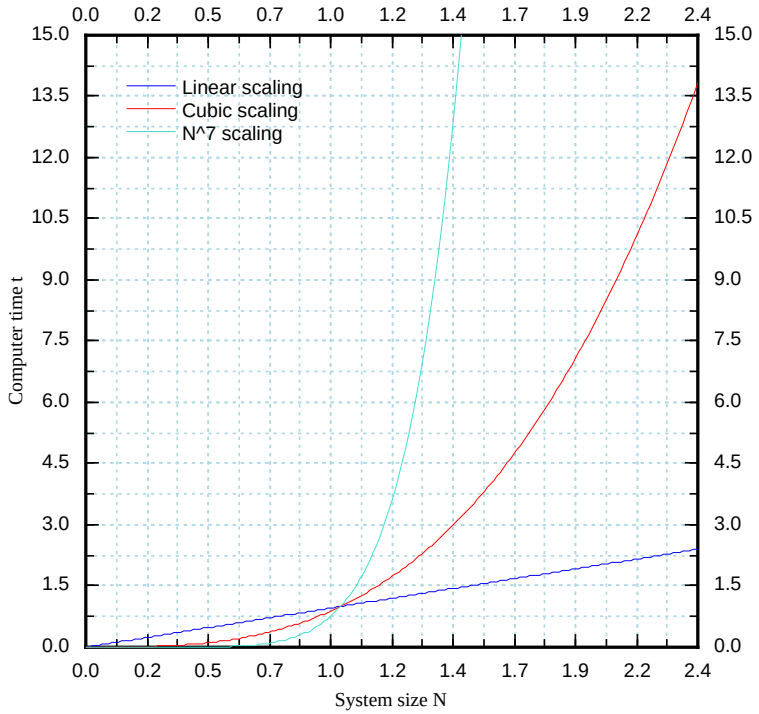
Many processors

- Advent of 'petascale computing' - parallel machines with tens or hundreds of thousands or even millions of processors. Exciting possibilities in this field.

How does QMC CPU time scale with number of processors? Do we need to do to do anything different algorithmically in order to exploit this new hardware fully?

Scaling with system size

Important consideration with all electronic structure methods : how does the computational cost increase as the size of the system N is increased?



method	scaling
Hartree-Fock	N^3
DFT	N^3
CCSD(T)	N^7

- Coupled cluster theory CCSD(T) probably most competitive quantum chemistry correlated wave function method, but the standard algorithm has disastrous scaling! (Recent developments could improve this).
- Great efforts made to write *linear-scaling* DFT codes over the last decade. Very difficult problem, but now more or less solved (e.g. ONETEP).

Basic scaling of QMC

Moving all N electrons **once**, *using delocalized basis (e.g. plane waves)*

- Evaluating orbitals (N orbitals expanded in N basis functions at each of N electron positions): $\mathcal{O}(N^3)$ (RATE DETERMINING STEP)
- Evaluating electron-electron and electron-ion interactions and Jastrow factor : $\mathcal{O}(N^2)$
- Re-evaluating ratio of new to old Slater determinant (requires storing and updating the cofactors of the matrix) : $\epsilon\,\mathcal{O}(N^3)$

Basic scaling of QMC

Moving all N electrons **once**, *using delocalized basis (e.g. plane waves)*

- Evaluating orbitals (N orbitals expanded in N basis functions at each of N electron positions): $\mathcal{O}(N^3)$ (RATE DETERMINING STEP)
- Evaluating electron-electron and electron-ion interactions and Jastrow factor : $\mathcal{O}(N^2)$
- Re-evaluating ratio of new to old Slater determinant (requires storing and updating the cofactors of the matrix) : $\epsilon\,\mathcal{O}(N^3)$

Moving all N electrons **once**, *using localized basis (e.g. Gaussians/blips)*

- Number of non-zero basis functions at random point independent of system size, therefore evaluating orbitals becomes $\mathcal{O}(N^2)$

Basic scaling of QMC

Moving all N electrons **once**, *using delocalized basis (e.g. plane waves)*

- Evaluating orbitals (N orbitals expanded in N basis functions at each of N electron positions): $\mathcal{O}(N^3)$ (RATE DETERMINING STEP)
- Evaluating electron-electron and electron-ion interactions and Jastrow factor : $\mathcal{O}(N^2)$
- Re-evaluating ratio of new to old Slater determinant (requires storing and updating the cofactors of the matrix) : $\epsilon\,\mathcal{O}(N^3)$

Moving all N electrons **once**, *using localized basis (e.g. Gaussians/blips)*

- Number of non-zero basis functions at random point independent of system size, therefore evaluating orbitals becomes $\mathcal{O}(N^2)$

Standard algorithm : $C = AN^2 + \epsilon N^3$

Current simulations :
ϵ is very small and currently $N \leq$ c. 3000 electrons
$\Longrightarrow \mathcal{O}(N^2)$ to move all electrons once

Basic scaling of QMC

Moving all N electrons **once**, *using delocalized basis (e.g. plane waves)*

- Evaluating orbitals (N orbitals expanded in N basis functions at each of N electron positions): $\mathcal{O}(N^3)$ (RATE DETERMINING STEP)
- Evaluating electron-electron and electron-ion interactions and Jastrow factor : $\mathcal{O}(N^2)$
- Re-evaluating ratio of new to old Slater determinant (requires storing and updating the cofactors of the matrix) : $\epsilon\,\mathcal{O}(N^3)$

Moving all N electrons **once**, *using localized basis (e.g. Gaussians/blips)*

- Number of non-zero basis functions at random point independent of system size, therefore evaluating orbitals becomes $\mathcal{O}(N^2)$

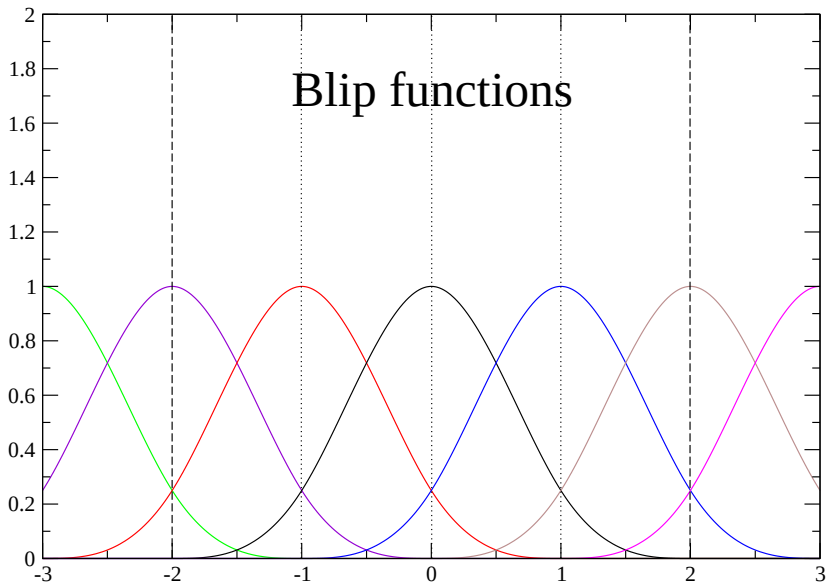
Standard algorithm : $C = AN^2 + \epsilon N^3$

Current simulations :
ϵ is very small and currently $N \leq$ c. 3000 electrons
$\Longrightarrow \mathcal{O}(N^2)$ to move all electrons once

Can we do any better than this?

Example of localized basis functions : blips

Expansion in localized spline functions on a uniform grid



Grid spacing unity

$$\begin{aligned} \phi(x) &= 1 - \frac{3}{2}x^2 + \frac{3}{4}|x|^3 \quad 0 \le |x| \le 1 \\ &= \frac{1}{4}(2 - |x|)^3 \quad 1 < |x| \le 2 \end{aligned}$$

Grid spacing a

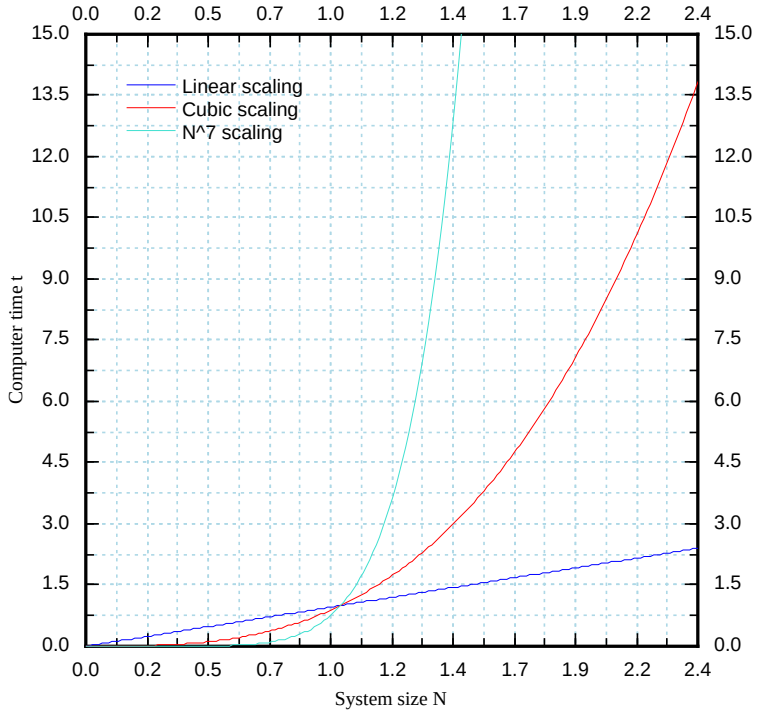
$$\begin{aligned} X_i &= ia \\ \Theta(x - X_i) &= \phi\left(\frac{(x - X_i)}{a}\right) \end{aligned}$$

In 3 dimensions there are only 64 non-zero blips for each position $\mathbf{r}$. With plane waves the number of functions in e.g. silicon is around 100 per atom.

GOOD : Achieved from transformation of wave function expanded in plane waves with accompanying huge efficiency increase. Localized. Universal.

BAD : Somewhat greedy with memory and disk. Extra step required (blip transformation of plane wave data file).

Linear scaling QMC?



method	scaling
Hartree-Fock	$N^3 - N^4$
DFT	N^3
QMC	N^3
CCSD(T)	N^7

Scaling improvement in QMC simply a matter of using localized basis sets and localized orbitals.

- Number of non-zero localized basis functions at random point independent of system size. Not the case with delocalized functions e.g. plane waves.
- Also, if an electron is far enough from the centre of a localized orbital, then we can assume it to be zero, thereby avoiding a great deal of unnecessary calculation.

Localized orbitals

- Non-singular linear transformations of the orbitals leave Slater determinants unchanged. So we can carry out such a transformation to a highly localized set of functions, truncate the functions so they are zero outside a certain radius and smoothly interpolate them to zero at a truncation radius.
- When an electron is moved, only a few functions must be evaluated; the others are zero as the electron is outside their truncation radii. The number of orbitals to be updated does not increase with system size.

Localized orbitals

- Non-singular linear transformations of the orbitals leave Slater determinants unchanged. So we can carry out such a transformation to a highly localized set of functions, truncate the functions so they are zero outside a certain radius and smoothly interpolate them to zero at a truncation radius.
- When an electron is moved, only a few functions must be evaluated; the others are zero as the electron is outside their truncation radii. The number of orbitals to be updated does not increase with system size.

New algorithm : $C = AN + BN^2 + \epsilon N^3$

ϵ is very small ; B is relatively small ; other tricks can improve the N^2 and N^3 terms $\Longrightarrow \mathcal{O}(N)$ to move all electrons once $\Longrightarrow$ linear scaling!

Localized orbitals

- Non-singular linear transformations of the orbitals leave Slater determinants unchanged. So we can carry out such a transformation to a highly localized set of functions, truncate the functions so they are zero outside a certain radius and smoothly interpolate them to zero at a truncation radius.
- When an electron is moved, only a few functions must be evaluated; the others are zero as the electron is outside their truncation radii. The number of orbitals to be updated does not increase with system size.

New algorithm : $C = AN + BN^2 + \epsilon N^3$

ϵ is very small ; B is relatively small ; other tricks can improve the N^2 and N^3 terms $\Longrightarrow \mathcal{O}(N)$ to move all electrons once $\Longrightarrow$ linear scaling!

However, we have forgotten something!

General linear scaling QMC is not possible

General linear scaling QMC is not possible

Stochastic QMC $\Longrightarrow$ mean value $\pm$ desired error bar from M statistically independent samples of local energy.

Variance of mean energy : $\sigma^2_{\rm run} = \frac{\sigma^2}{M}$

Total computer time: $T_{\rm run} = MT_{\rm sample} = \frac{\sigma^2 T_{\rm sample}}{\sigma^2_{\rm run}}$ where $T_{\rm sample} \propto N + \epsilon N^2$

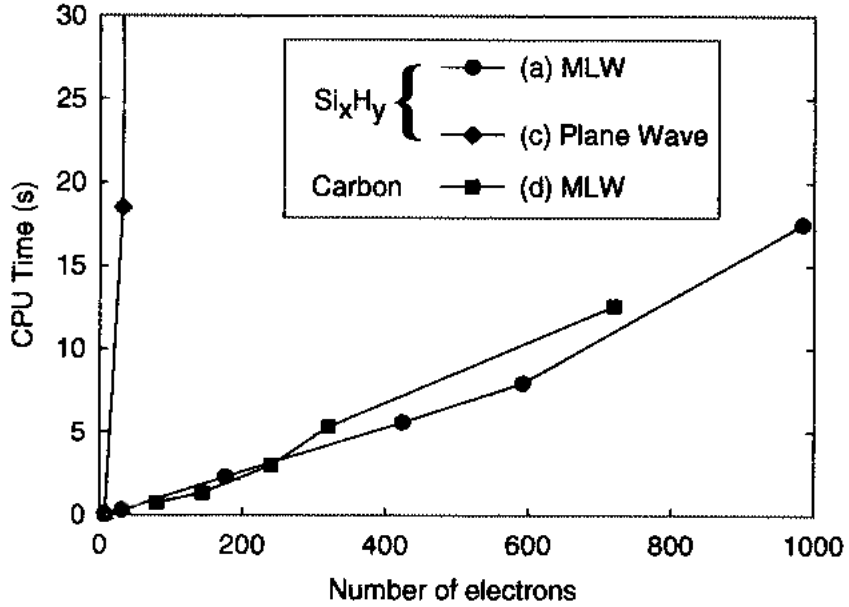
Sample variance σ^2 proportional to number of electrons N (if assume energies of electrons uncorrelated), $\sigma^2_{\rm run}$ is fixed (desired error bar), $T_{\rm sample}$ is proportional to N (say) thus overall $T_{\rm run}$ proportional to N^2 to maintain desired error bar $\Longrightarrow$ quadratic scaling!

Properties of most interest: e.g. defect formation energies, energy barriers, excitation energies i.e. energy differences which become independent of system size when the system is large enough. To perform such a calculation we require a statistical error bar which is independent of system size.

NB : Some properties (e.g. cohesive energies of solids) can be derived from total energies per atom. Then sample variance still increases linearly, but error bar decreased by factor of N and thus number of moves required *decreases* linearly. Hence total cost *independent of the size of the system* (even better than linear scaling!). However, such properties are of limited interest.

‘Linear scaling’ in CASINO

These algorithms have been implemented in CASINO. We form QMC wave functions with significant sparsity in the Slater determinant by using highly localized orbitals instead of the delocalized Bloch orbitals that come out of standard band calculations. Use localized blip or Gaussian basis. That’s it.



Linear scaling QMC fallacy No. 1 - paper titles

Linear scaling quantum Monte Carlo calculations, Williamson *et al.* (2001).

Linear scaling quantum Monte Carlo technique with non-orthogonal localized orbitals, Alfè and Gillan (2004).

Linear scaling for the local energy in quantum Monte Carlo, Manten and Lüchow (2003).

- Memory requirement scales linearly.
- Time to do one MC move and calculate local energy once scales linearly.
- But need to do many MC moves - in fact an *increasing* number of them as the system size increases if you want to keep the error bar the same. Time taken to calculate the local energy to a given error bar thus scales as the square of the system size, so according to all previously established conventions, it should be called *quadratic scaling quantum Monte Carlo*.

Thus only Manten alludes to this in his choice of title.

Linear scaling QMC fallacy No. 1 - paper titles

Linear scaling quantum Monte Carlo calculations, Williamson *et al.* (2001).

Linear scaling quantum Monte Carlo technique with non-orthogonal localized orbitals, Alfè and Gillan (2004).

Linear scaling for the local energy in quantum Monte Carlo, Manten and Lüchow (2003).

- Memory requirement scales linearly.
- Time to do one MC move and calculate local energy once scales linearly.
- But need to do many MC moves - in fact an *increasing* number of them as the system size increases if you want to keep the error bar the same. Time taken to calculate the local energy to a given error bar thus scales as the square of the system size, so according to all previously established conventions, it should be called *quadratic scaling quantum Monte Carlo*.

Thus only Manten alludes to this in his choice of title.

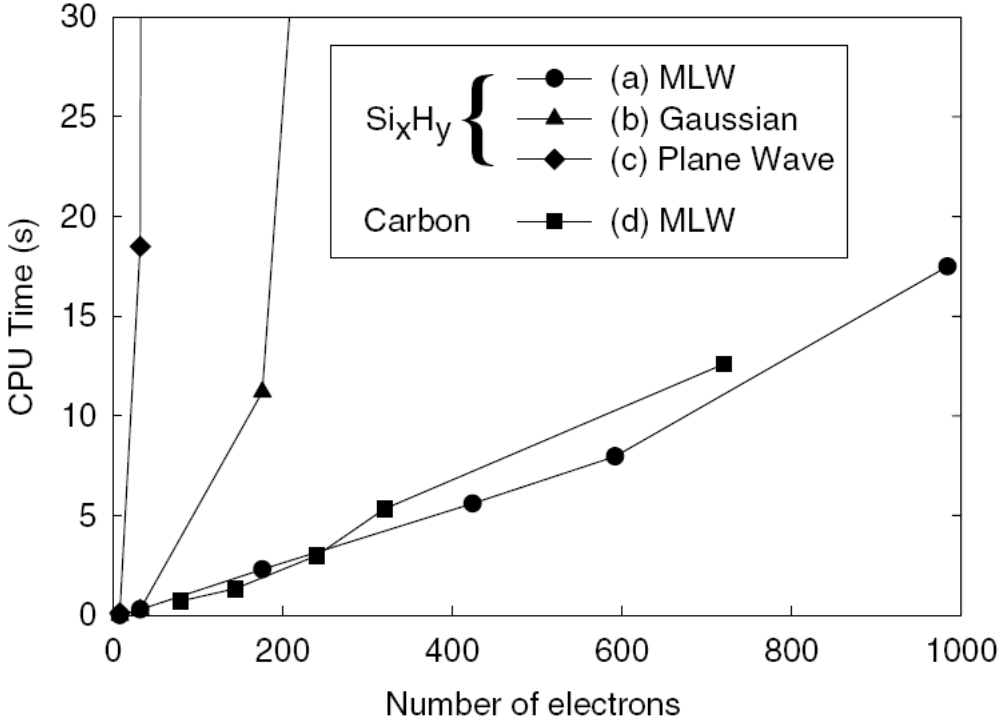
"I think you know the answer to that, Mike. Because it sounds cool."

A.J. Williamson, Leiden conference, 2004

Linear scaling QMC fallacy No. 2

The necessity of splines

Misleading figure from Williamson paper



Fails to mention that curves marked 'Gaussian' and 'Plane Wave' (which are basis sets) are produced with delocalized orbitals whereas his curve marked 'MLW' (Maximally Localized Wannier function - a kind of orbital) is done with localized orbitals in addition to his localized spline basis set.

In fact Gaussians have the potential to be just as effective a representation as splines/blips in 'linear scaling QMC' calculations!

How to produce localized orbitals

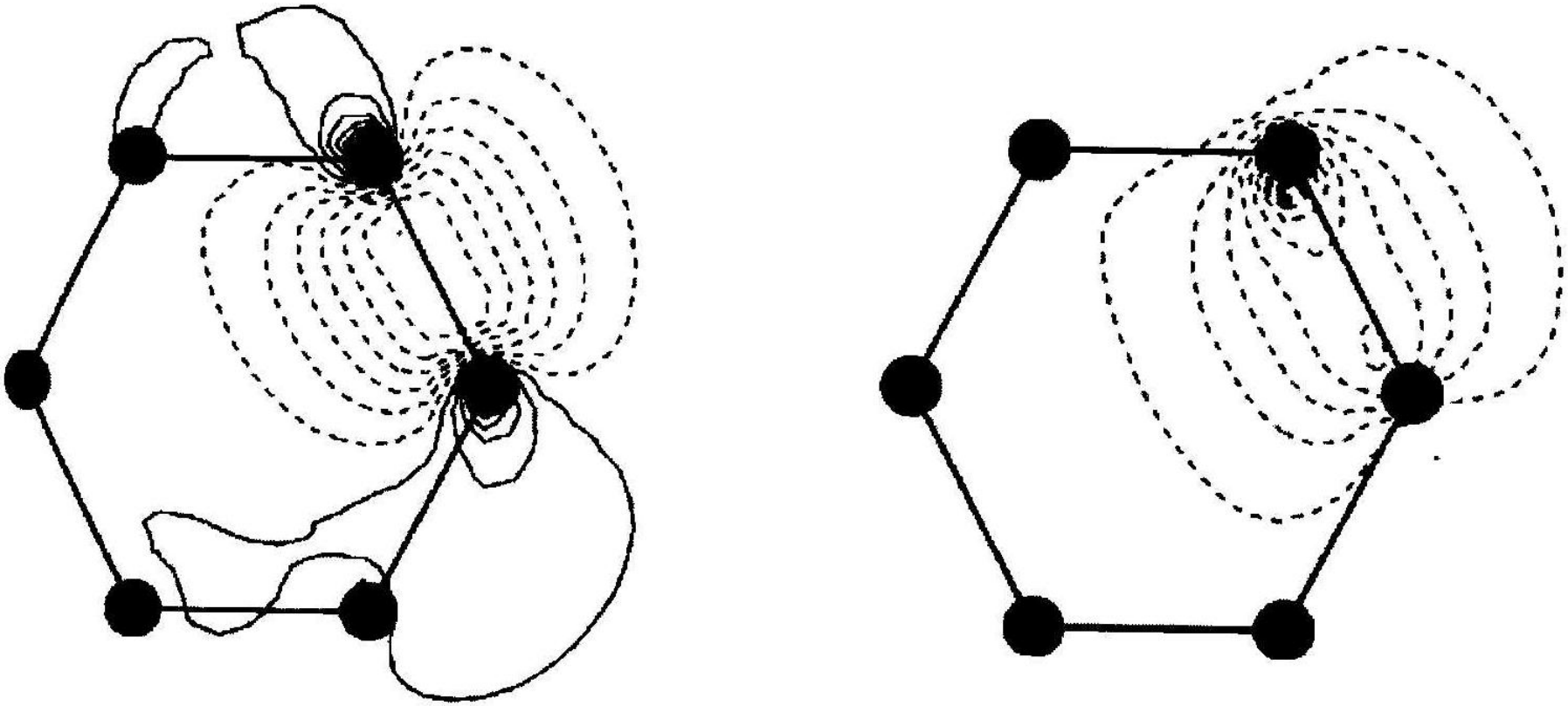
- Calculate orthogonal extended Bloch orbitals in the usual way with your DFT program. Form appropriate linear combinations to produce orbitals localized according to some criterion.

$$\phi_m(\mathbf{r}) = \sum_{n=1}^{M} c_{mn}\psi_n(\mathbf{r}), \qquad m = 1, 2, \ldots, M$$

$$\det|\phi_m(\mathbf{r}_i)| = \det|c_{mn}| \cdot \det|\psi_n(\mathbf{r}_i)|$$

- Determinant unchanged apart from constant factor $\det|c_{mn}|$, therefore total energy unchanged in QMC.
- If the transformation matrix is unitary, then the resulting orbitals remain orthogonal. If it isn't, then they are nonorthogonal. Whichever - the above property of determinants is true. Therefore we can use nonorthogonal orbitals in QMC.
- Additional freedom gained by dispensing with orthogonality can be exploited to improve the localization of the orbitals.

Orthogonal vs. non-orthogonal



Comparison of orthogonal (left) and non-orthogonal (right) maximally localized orbitals for C-C σ bond in benzene C_6H_6. The non-orthogonal orbitals are more localized and more transferable since the extended wiggles in the orthogonal functions depend in detail upon the neighbouring atoms.

Alfè/Gillan localization

- Choose some region of arbitrary shape contained within the unit cell. Want to find the combination $\phi(\mathbf{r}) = \sum_{n=1}^{M} c_n \psi_n(\mathbf{r})$ such that $\phi(\mathbf{r})$ is maximally localized in this region.
- Can vary the c_n to maximize the *localization weight* P :

$$P = \frac{\int_{\text{region}} |\phi(\mathbf{r})|^2 \, d\mathbf{r}}{\int_{\text{cell}} |\phi(\mathbf{r})|^2 \, d\mathbf{r}} = \frac{\sum_{m,n} c_m^* A_{mn}^{\text{region}} c_n}{\sum_{m,n} c_M^* A_{mn}^{\text{cell}} c_n}$$

where

$$A_{mn}^{\Omega} = \int_{\Omega} \psi_m^* \psi_n \, d\mathbf{r}$$

Then P takes its maximum value when the c_n are the components of the eigenvector associated with the largest eigenvalue λ_1 of the generalized eigenvalue equation

$$\sum_n A_{mn}^{\text{region}} c_n = \lambda_\alpha \sum_n A_{mn}^{\text{cell}} c_n$$

and this maximum P is equal to λ_1.

[*J. Phys.: Cond. Mat* **16**, L305 (2004)]

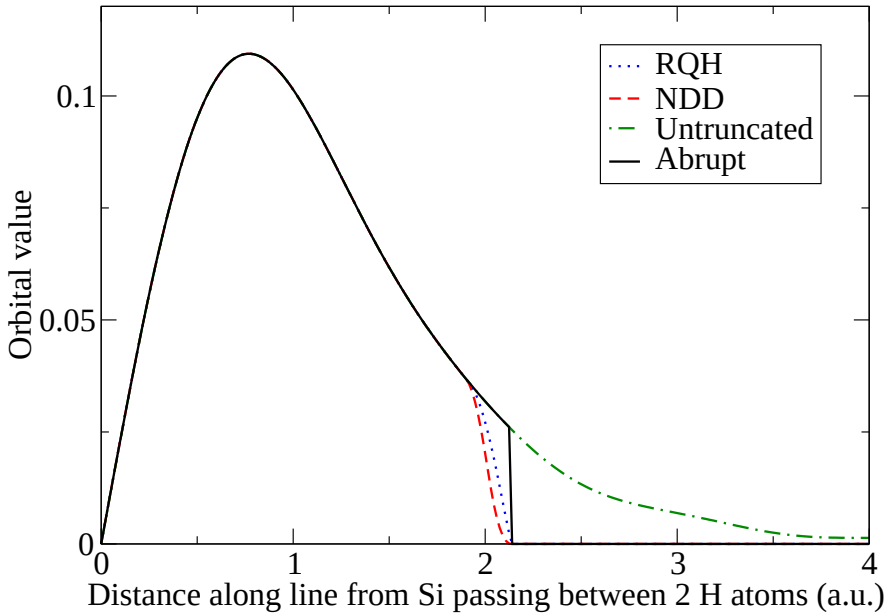
Reboredo-Williamson localization

Optimized nonorthogonal localized orbitals
for linear scaling QMC calculations

Phys. Rev. B **71**, *121105 (2005)*

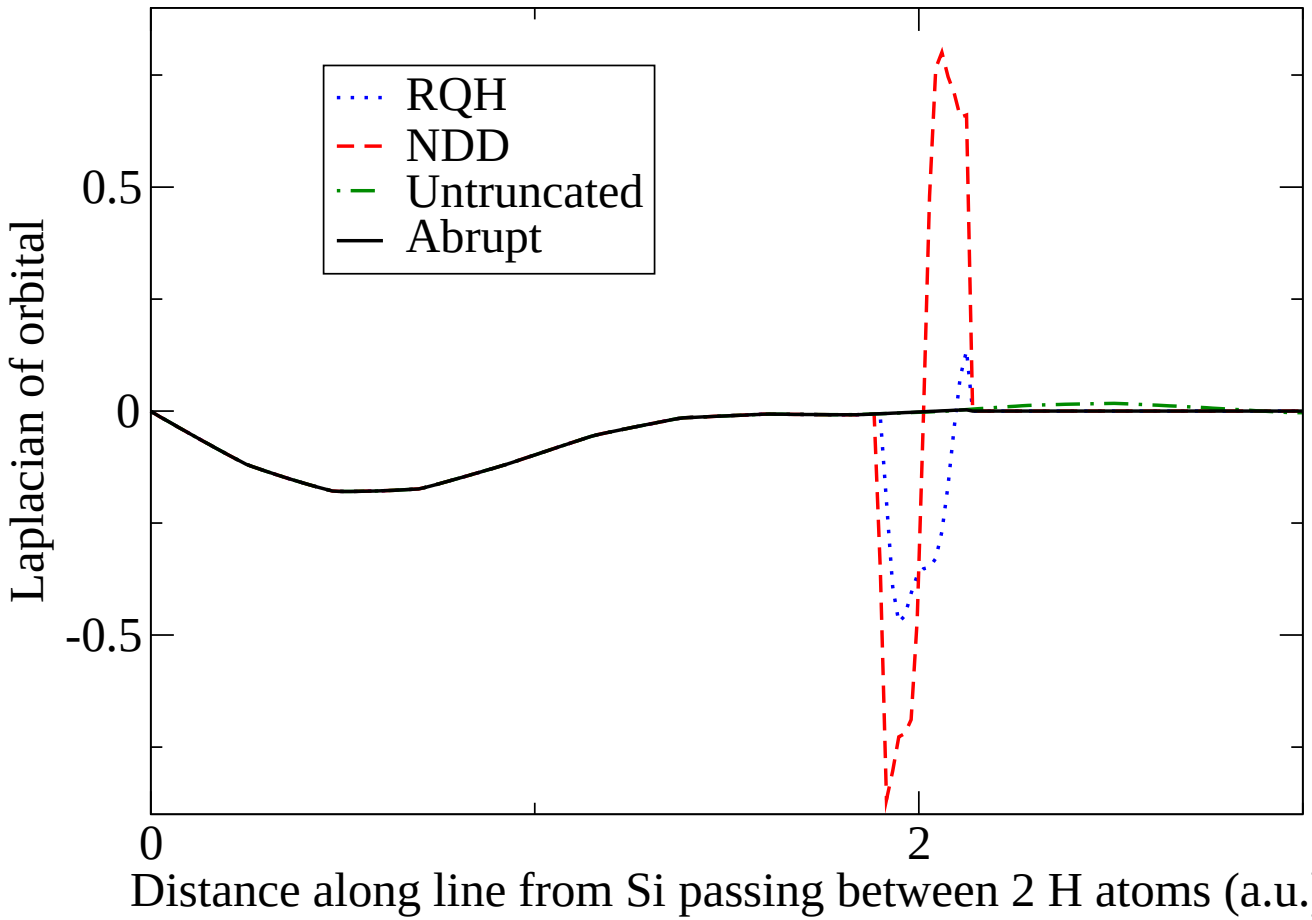
Turns out to be essentially the same thing as Alfè, though possibly less clearly explained.

Cutting things off



- To get any benefit from localized orbitals, need to decide on *truncation radius* outside of which they are taken to be exactly zero. Then we don't need to do any work to calculate the orbital if electron is further away than this.
- Then need to decide how to cut off the orbital. Can cutoff abruptly, or can bring the orbital smoothly to zero over some truncation region by multiplying by an appropriate function. One might initially think the latter is more sensible.

Laplacian of truncated orbitals in silane



Smooth truncation leads to large unphysical peaks in the local kinetic energy in the truncation region!

Cutoff conclusions

- Abrupt truncation gives stable DMC simulations, low variances and energies similar to those of untruncated orbitals. However, it suffers from a theoretical drawback: the gradients and Laplacians of the orbitals contain Dirac delta functions, which are not sampled in QMC, thereby invalidating the variational principles usually satisfied by the VMC and DMC energies. The bias due to this is however extremely small, and can be made arbitrarily small by increasing the cutoff radius.
- Various smooth truncation schemes have been tried, but none perform as well as abrupt truncation. All such schemes produce large, unphysical peaks in the local kinetic energy in the truncation region. And as we have introduced a new small length scale into the problem (the skin thickness) that is much smaller than the physically reasonable timestep, DMC gives a large time step bias and frequent population-explosion catastrophes.

It is therefore recommended that localised orbitals be truncated abruptly.

- If the truncation radii of the orbitals are sufficiently large, the bias due to abrupt truncation is much less than the statistical error. The bias in the kinetic energy is given approximately by the change in the sum of the orbital kinetic energies upon truncation, allowing an estimate to be made of this bias; the bias in the total energy is smaller than this.

Improved scaling calculations in CASINO

How to do this in practice

- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**= blip.

Improved scaling calculations in CASINO

How to do this in practice

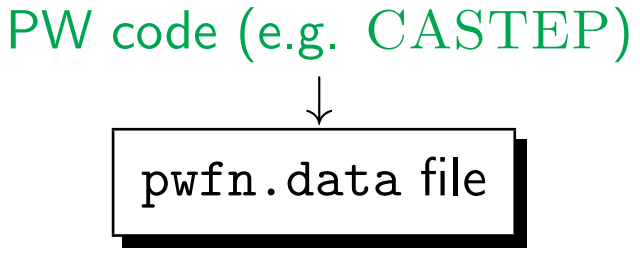
- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**= blip.

PW code (e.g. CASTEP)

Improved scaling calculations in CASINO

How to do this in practice

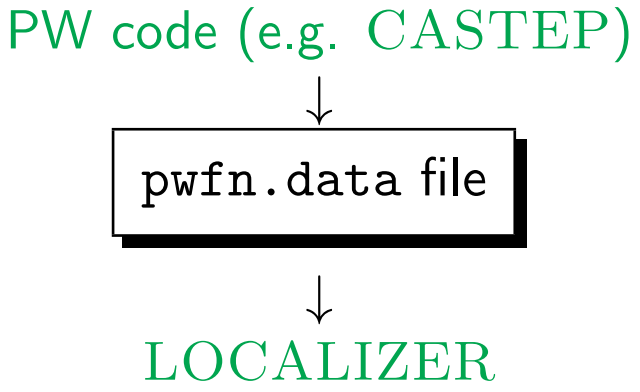
- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**= blip.



Improved scaling calculations in CASINO

How to do this in practice

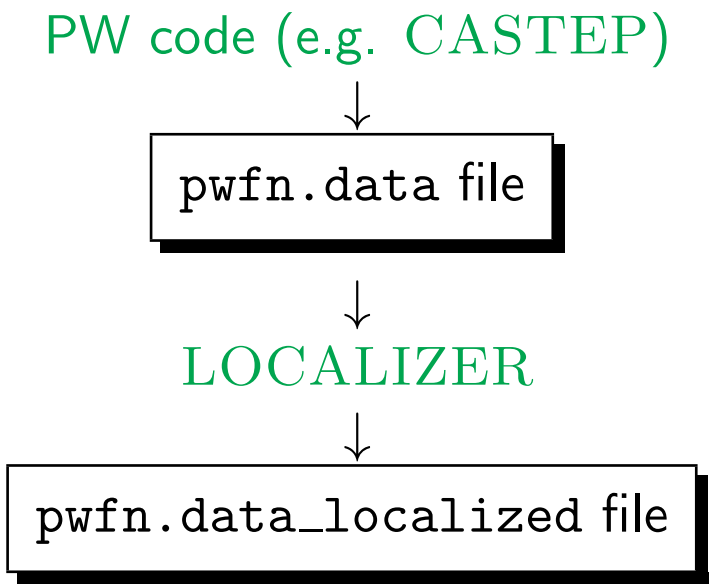
- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**=blip.



Improved scaling calculations in CASINO

How to do this in practice

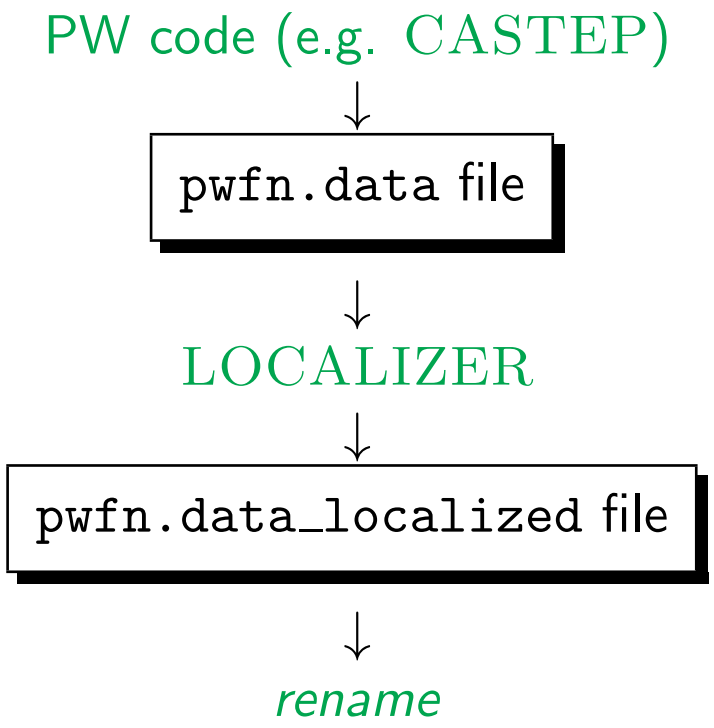
- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**= blip.



Improved scaling calculations in CASINO

How to do this in practice

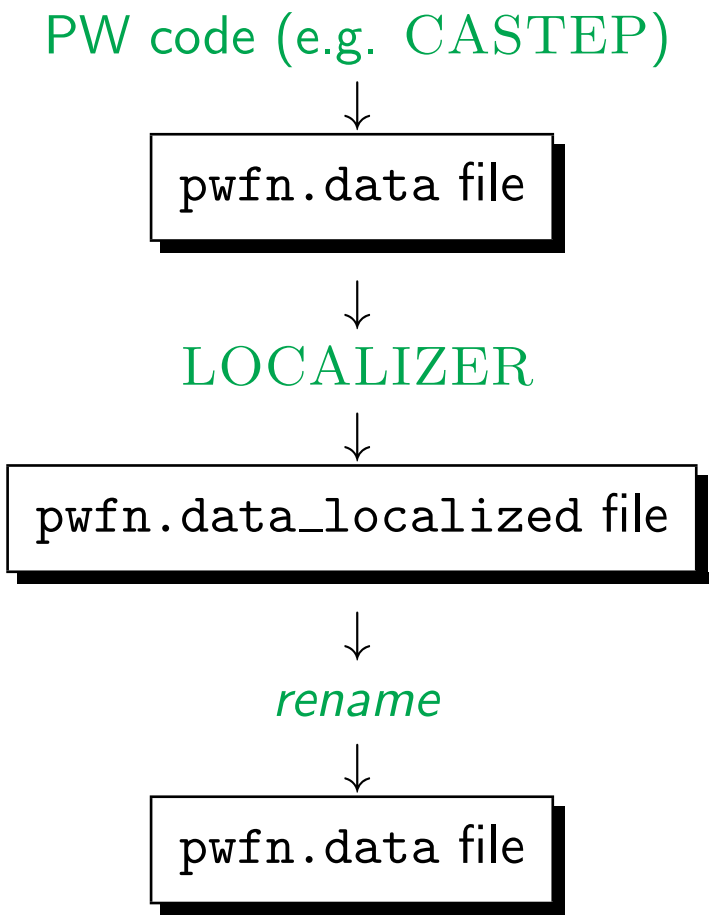
- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**= blip.



Improved scaling calculations in CASINO

How to do this in practice

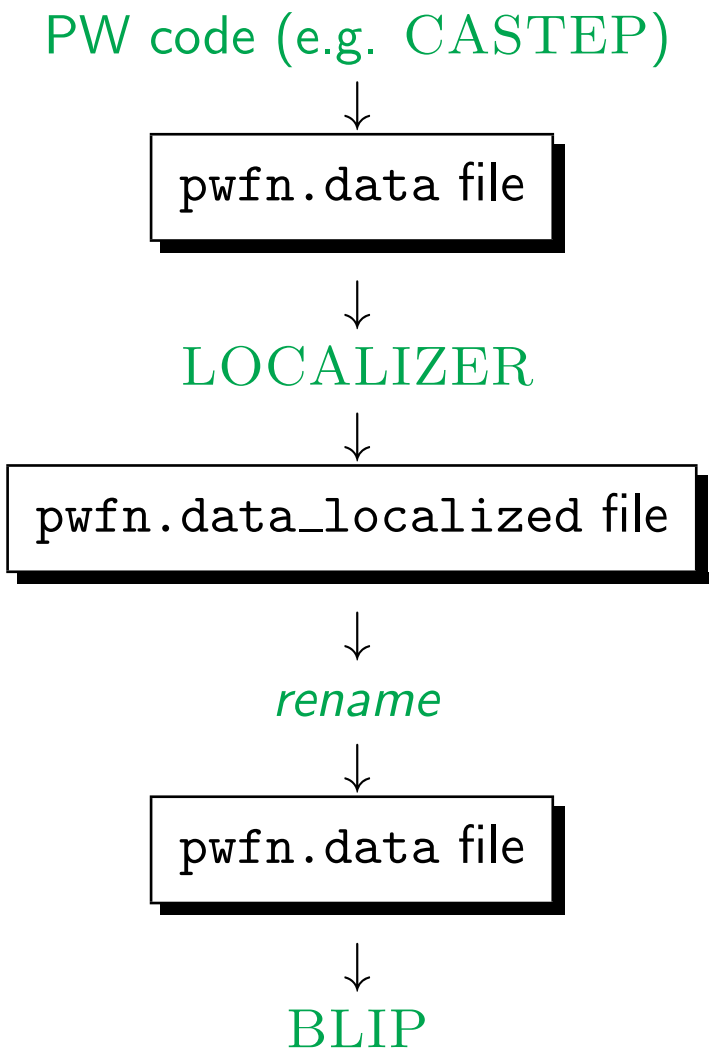
- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**= blip.



Improved scaling calculations in CASINO

How to do this in practice

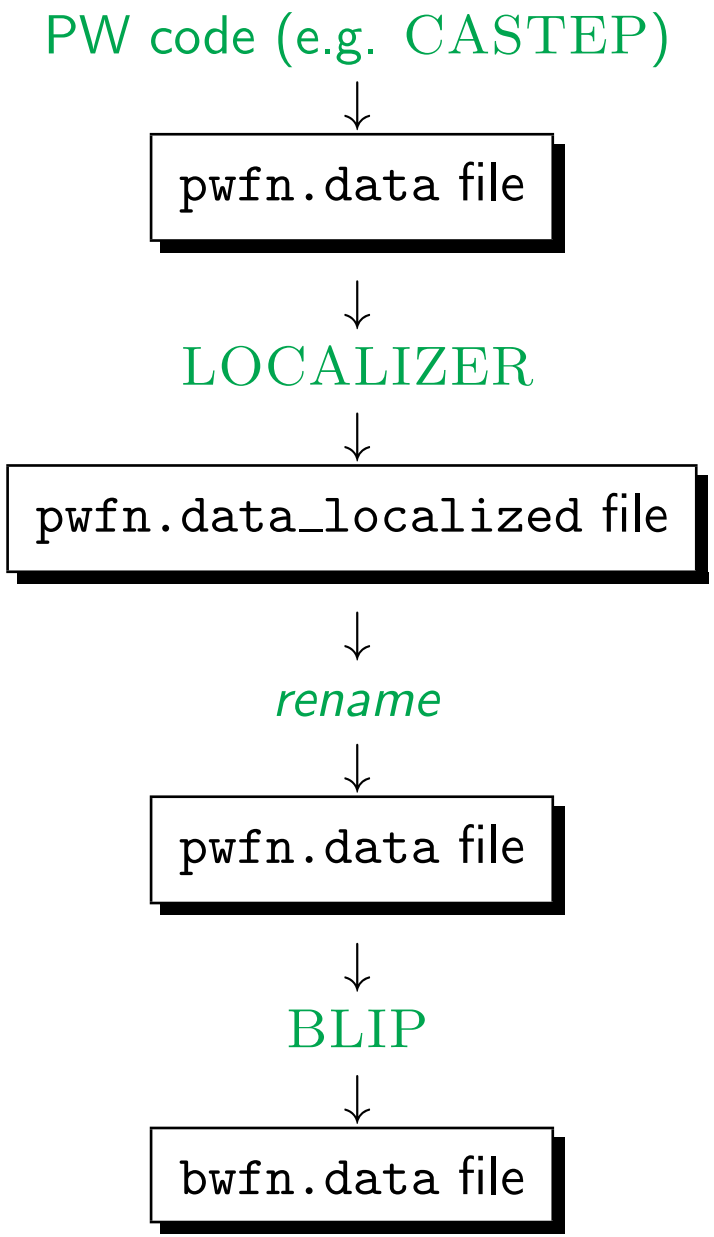
- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**= blip.



Improved scaling calculations in CASINO

How to do this in practice

- Currently must start with plane-wave DFT calculation to generate the trial wave function (Gaussians to be implemented..).
- Linear transformation of Bloch orbitals to localized orbitals with LOCALIZER utility (implements Gillan/Alfè method). Can skip this step if desired.
- Re-expand localized orbitals in blips rather than plane waves using BLIP utility.
- Final `bwfn.data` files contains localized orbital/localized basis representation. Use normally in CASINO with **atom_basis_type**=blip.



Generation of localized orbitals - details

- The LOCALIZER utility requires a `pwfn.data` file holding the Bloch orbitals represented in a plane-wave basis and a `centres.dat` file of format:

```
Number of localization centres
 N
Display coefficients of linear transformation (0=NO; 1=YES)
 0
Use spherical (1) or parallelepiped (2) localization regions
 1
x,y & z coords of centres ; radius ; no. orbs on centre (up & dn spin)
 <pos(1,1)> <pos(2,1)> <pos(3,1)> <radius(1)> <norbs_up(1)> <norbs_dn(1)>
 ...
 <pos(1,N)> <pos(2,N)> <pos(3,N)> <radius(N)> <norbs_up(N)> <norbs_dn(N)>
```

- Choice of localization centres requires some chemical intuition - see discussion in Alfè and Gillan. Optimization of localization centres is possible and will be implemented shortly.
- Orbitals will become linearly dependent as two centres approach one another. In limit that two centres located in same place, the orbitals localized on those centres will be identical, so define instead a single centre and increase number of orbitals localized on that centre.
- LOCALIZER only works for Bloch orbs at Γ (not serious restriction in large systems).

Generation of blip representation - details

- The BLIP utility converts `pwfn.data` (plane-wave) files to `bwfn.data` (blip) files. CASINO should both run faster and scale better with system size with blips than with plane-waves. Blips can require a lot of memory and disk space, however. The gain in speed with respect to plane waves should be of the order of number of plane waves/64. This is clearly a *good thing*.
- Improve quality of blip expansion (i.e. fineness of blip grid) by increasing *grid multiplicity* **xmul**. This gives more blip coeffs and thus needs more memory, but CPU time should not change. For accurate work, one may want to experiment with **xmul** greater than 1.0. However, it might be more efficient to keep the grid multiplicity to 1.0 and increase the plane wave cutoff instead.
- BLIP asks if overlap test is required i.e. sample the wave function, Laplacian and gradient at large no. of random points in simulation cell and compute overlap α of blip orbitals with original plane-wave orbitals:

$$\alpha = \frac{\langle BW|PW\rangle}{\sqrt{\langle BW|BW\rangle \langle PW|PW\rangle}}$$

  The closer α is to 1, the better the blip representation. By increasing **xmul** or the plane-wave cutoff one can make α as close to 1 as desired.
- BLIP will ask you whether you wish to calculate the orbital KE in PW and blip representations. Obviously these should agree closely.

Generation of blip representation of localized orbitals - details

- Can also use BLIP to generate truncated blip representation of localized orbitals. Need transformed `pwfn.data` and same `centers.dat` file that was used as input to LOCALIZER. If latter not present then orbitals will be represented by blip grid that spans the entire simulation cell.
- If you want to truncate the orbitals add the following two lines to the end of `centers.dat`:

```
Minimum skin thickness (a.u.)
 0.d0
```

  For each state CASINO takes requested localization region, adds minimum skin thickness, then choose smallest parallelogram-shaped subgrid of blip grid points that contains this sphere. Then choose skin thickness to be such that the sphere just touches one or more sides of the paralleliped. The requested localization radius (excluding the skin thickness) is referred to as the *inner truncation radius* and the localization radius (including the actual skin thickness) is referred to as the *outer truncation radius*.
- In subsequent QMC calculation, if input keyword **bsmooth** is `T` then the orbital is brought smoothly to zero between these radii. If **bsmooth** is `F` (recommended) then abrupt truncation of the orbital occurs at the outer truncation radius.

All-electron QMC - scaling with atomic number Z

The cost of all-electron QMC calculation increases *rapidly* with atomic number - somewhere between $Z^{5.5}$ (Ceperley 1988) and $Z^{6.5}$ (Hammond 1987) according to best theoretical estimates.

All-electron QMC - scaling with atomic number Z

The cost of all-electron QMC calculation increases *rapidly* with atomic number - somewhere between $Z^{5.5}$ (Ceperley 1988) and $Z^{6.5}$ (Hammond 1987) according to best theoretical estimates.

Why so much?

All-electron QMC - scaling with atomic number Z

The cost of all-electron QMC calculation increases *rapidly* with atomic number - somewhere between $Z^{5.5}$ (Ceperley 1988) and $Z^{6.5}$ (Hammond 1987) according to best theoretical estimates.

Why so much?Very roughly :

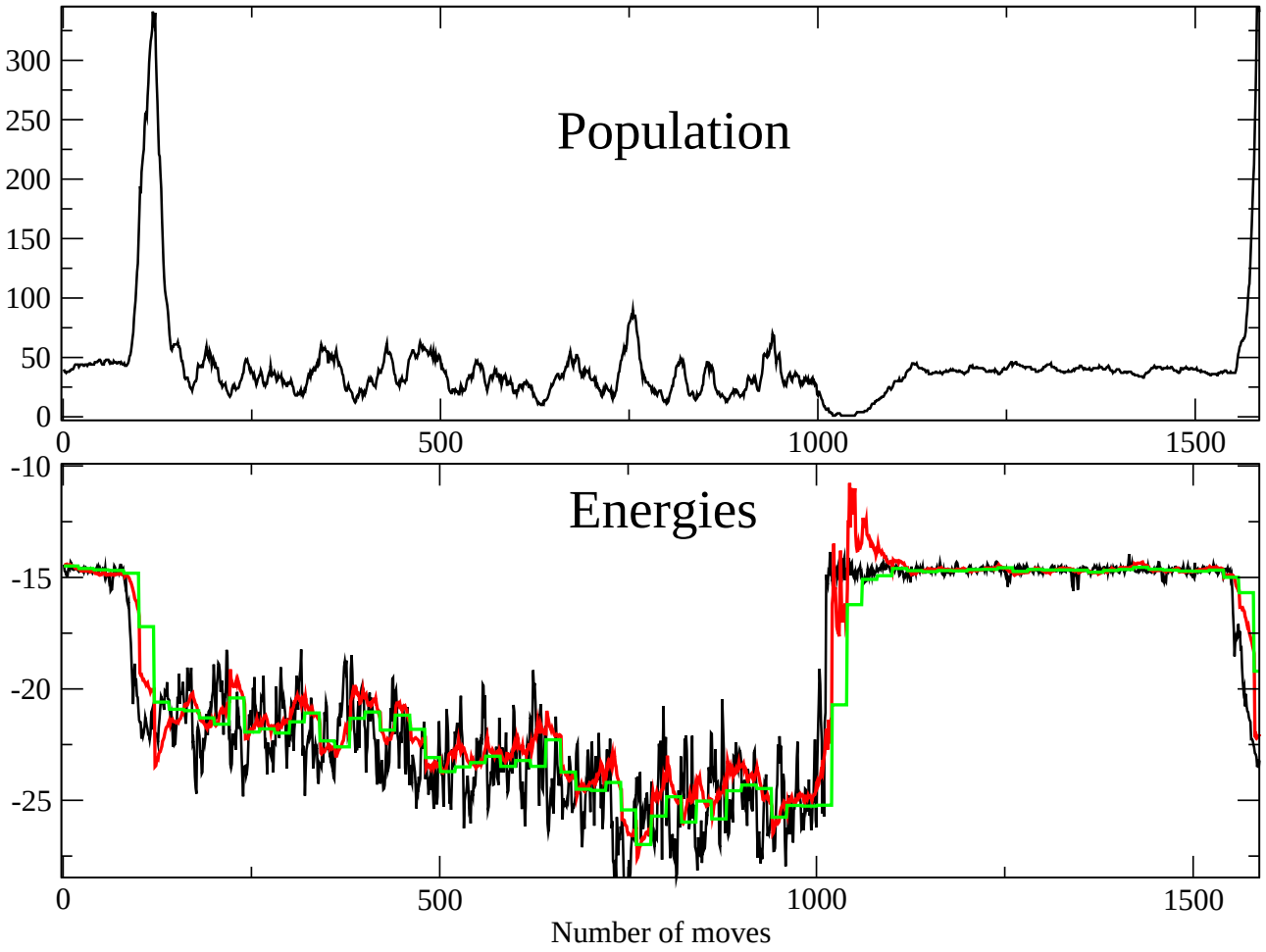
- Usual N^3 scaling with system size ($N = Z$ for neutral atom - see later)
- Increasing Z leads to shorter length scale variations in the wave function in the core region ('narrower orbitals'). The RMS distance diffused by an electron in a single move must be less than this length scale to avoid large timestep errors. The use of shorter time steps results in serial correlation over more (proportional to Z^2) moves - since the length scale for 'valence electrons' is much larger and essentially independent of Z.

Other potential problem : fluctuations in the local energy could be large near nucleus because kinetic and potential energy terms terms diverge there (with opposite signs). Clearly worse for heavier atoms.

If time step too large get unacceptable bias in the results, and virtually inevitable 'catastrophic behaviour' in DMC.

DMC stability and catastrophes

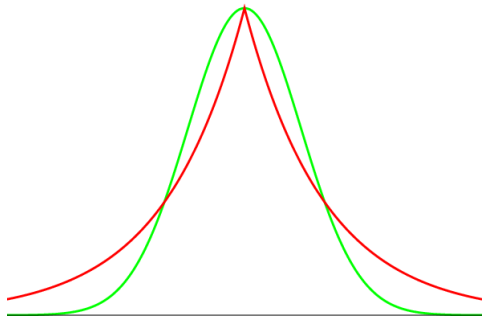
As an example, here is a *persistent electron catastrophe* (increase DMC time step a lot to make it likely to see this. Combination of electron very close to nucleus with divergent local energy, with low probability of acceptance for moving away). Shift in E_T stabilizes it (and any copies). Fixed negative contribution to E_L until random fluctuation removes the persistent electron!



Fix the cusp!

Cusp conditions prescribe the proper derivative discontinuities at the particle collision points, and ensure the divergence in the local potential is cancelled by an opposite divergence in the local kinetic energy.

- Electron-nuclear : e.g. H atom has cusp at origin : $\Psi(r) = \exp(-Zr)$



- Electron-electron : conditions on Slater-Jastrow $\Psi = D\exp(-u)$

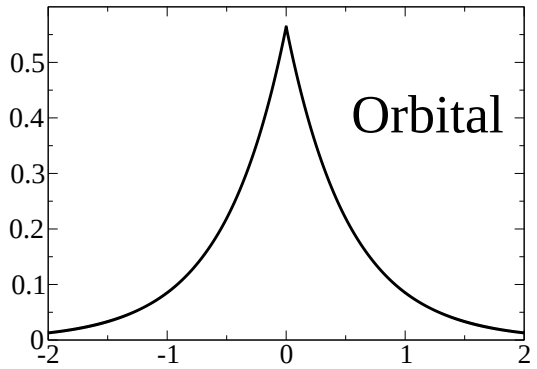
$$\left.\frac{\partial u}{\partial r}\right|_{r=0} = -\frac{1}{2} \quad \text{antiparallel spins}$$

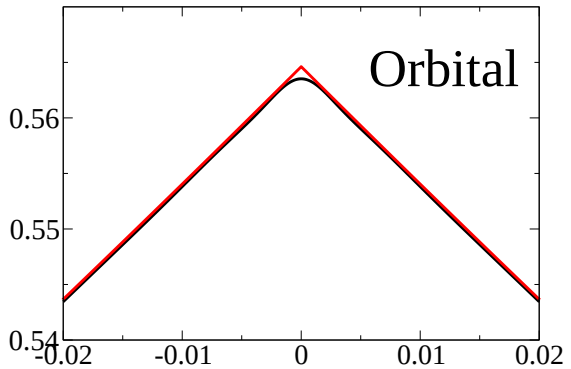
$$\left.\frac{\partial u}{\partial r}\right|_{r=0} = -\frac{1}{4} \quad \text{parallel spins}$$

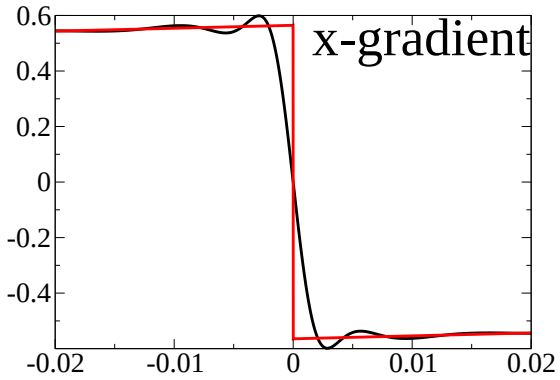
- Can therefore enforce electron-electron cusp conditions by imposing constraints on the Jastrow factor, but what about electron-nuclear?

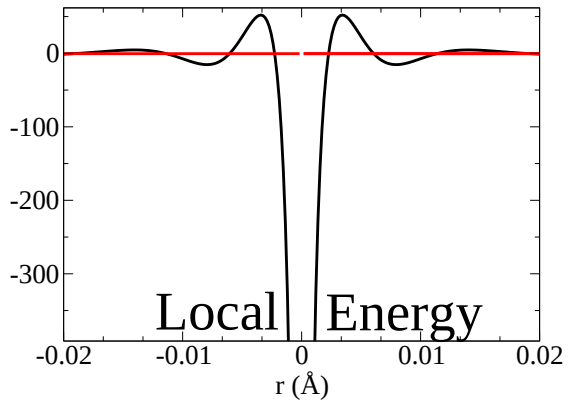
Cusp conditions for the orbitals

- Can imitate cusp by including very narrow Gaussians (i.e. with very high exponents) in the basis sets.
- Behaviour still incorrect in small region around nucleus. Can fix by chopping out part of the wave function in that region, and replacing with a polynomial that obeys suitable constraints (i.e. continuous first three derivatives at the join ; obey cusp condition ; choose $\phi(0)$ to minimize fluctuations in one-electron local energy inside cusp radius).

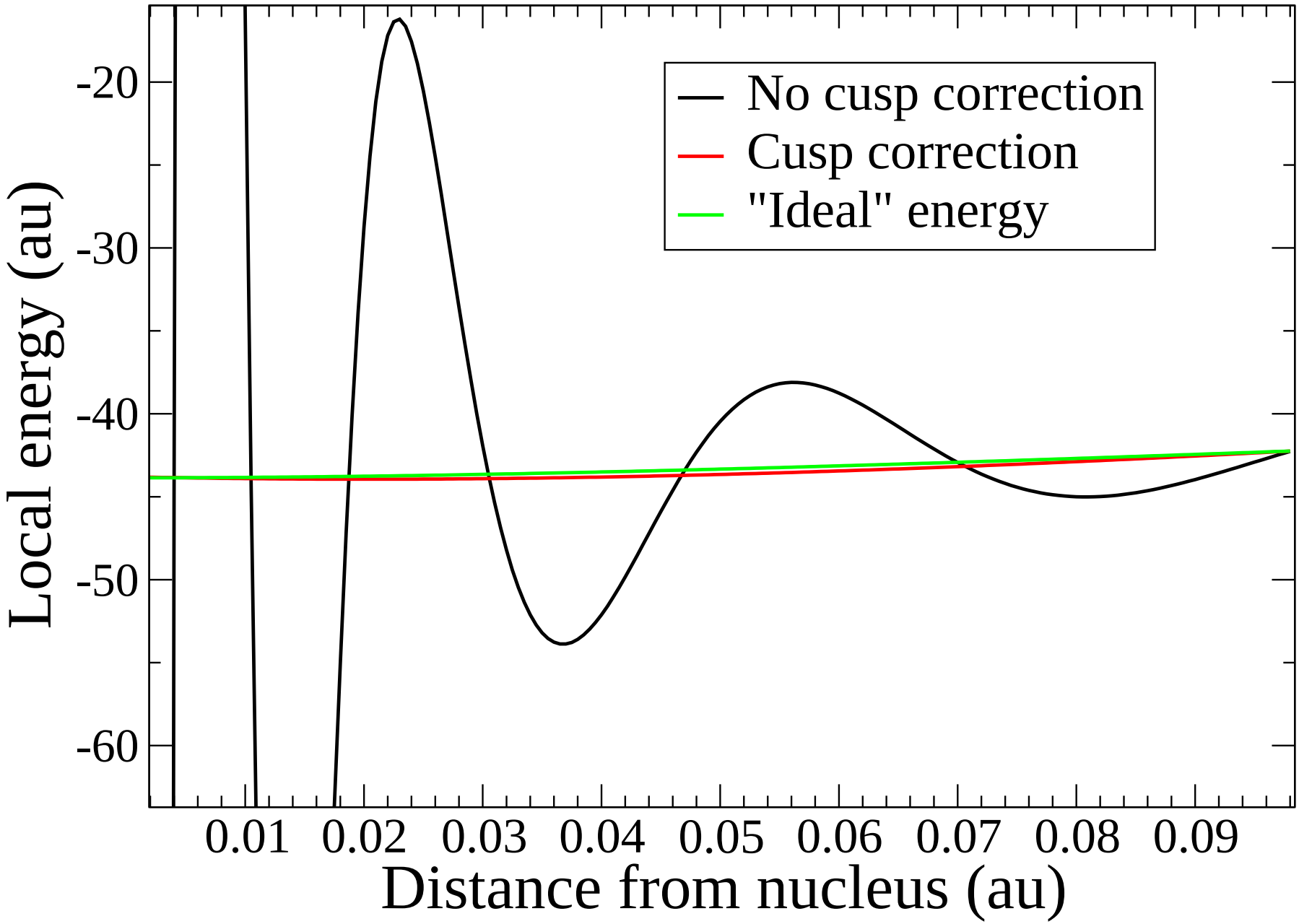






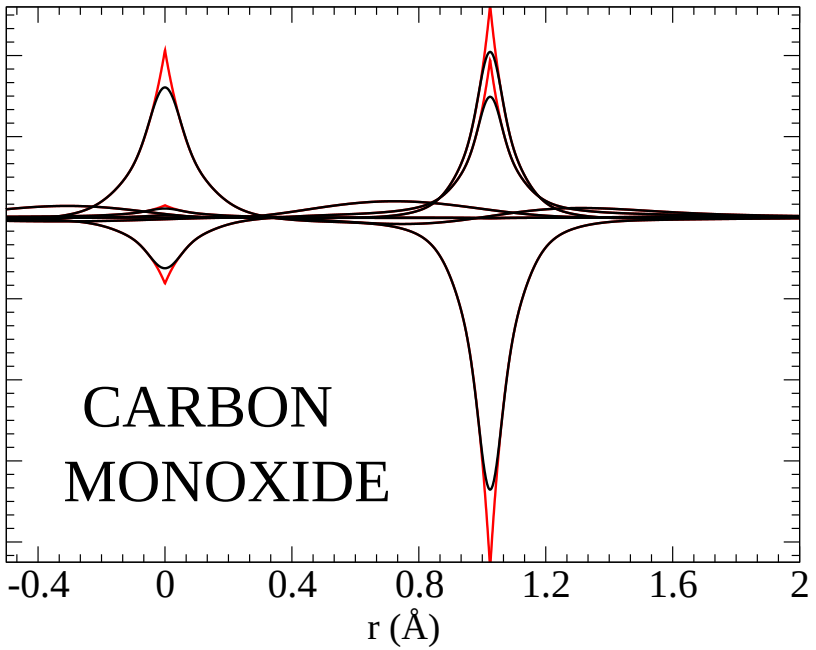


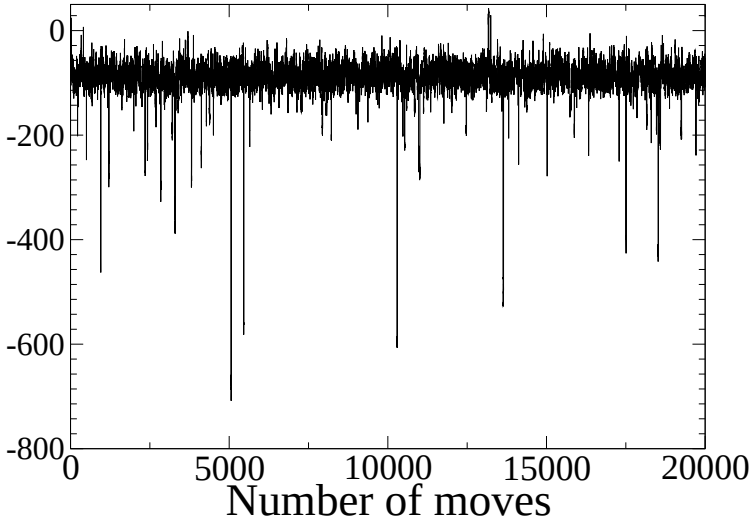
Effective one-electron local energy in CIF

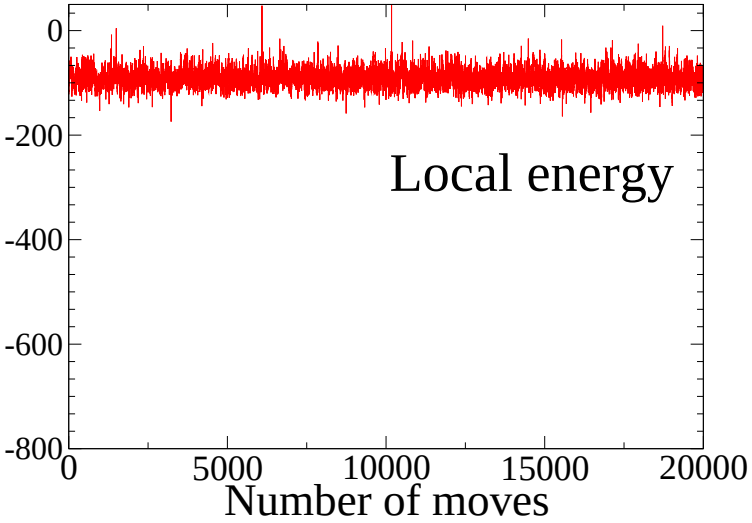


Fluctuations: VMC for CO molecule

- How well do the polynomial cusp corrections work for a bog-standard off-the-shelf quantum chemistry Gaussian basis set which doesn't even try to imitate the cusp?



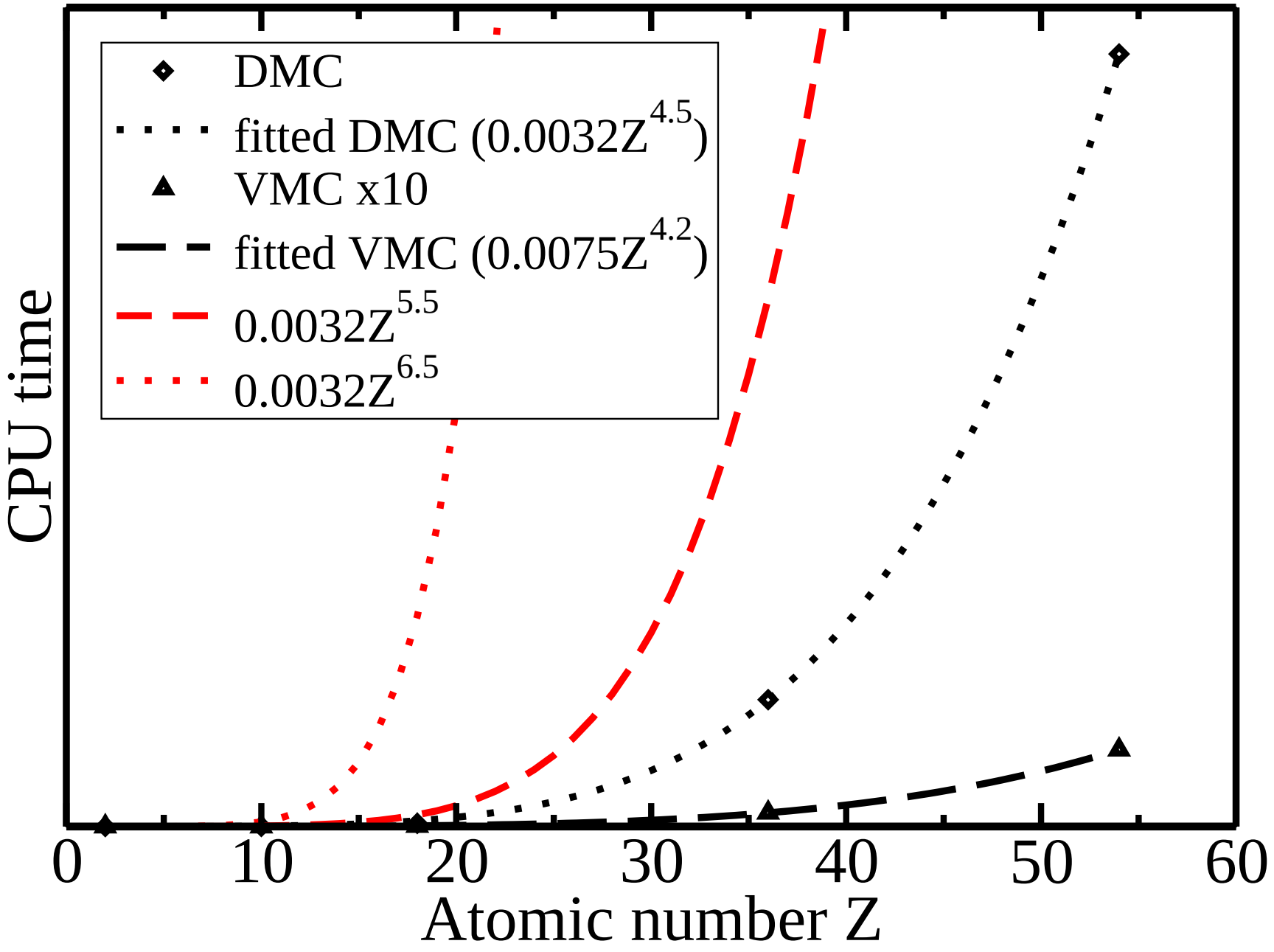




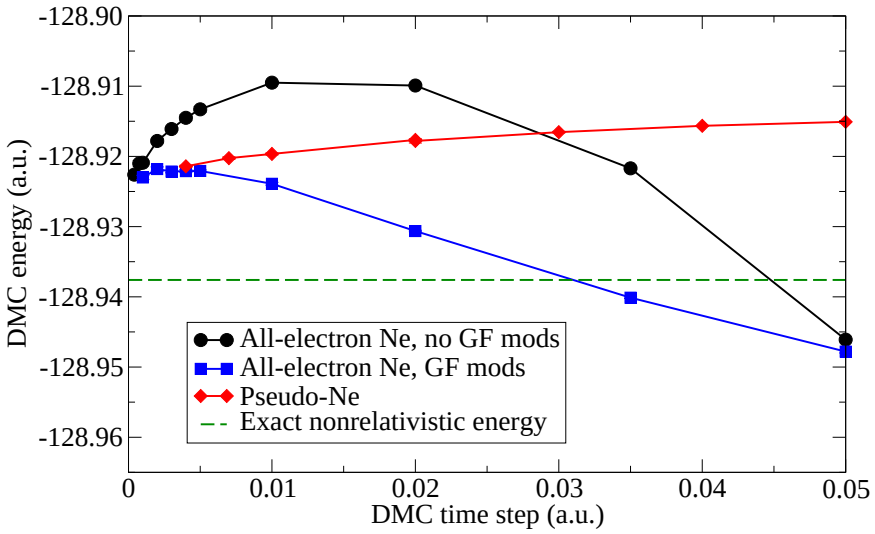
Results for noble gas atoms : He, Ne, Ar, Kr, Xe

atom	DMC total energy (au)	Exact	DMC % corr
helium ($Z = 2$)	-2.903719 $\pm$ 0.000002	-2.903724	100%
neon ($Z = 10$)	-128.9231 $\pm$ 0.0001	-128.939	96%
argon ($Z = 18$)	-527.4840 $\pm$ 0.0002	-527.55	91%
krypton ($Z = 36$)	-2753.7427 $\pm$ 0.0006	-2754.13	82%
xenon ($Z = 54$)	-7234.785 $\pm$ 0.001	-7235.57	77%

Scaling of CPU time with Z : He, Ne, Ar, Kr, Xe

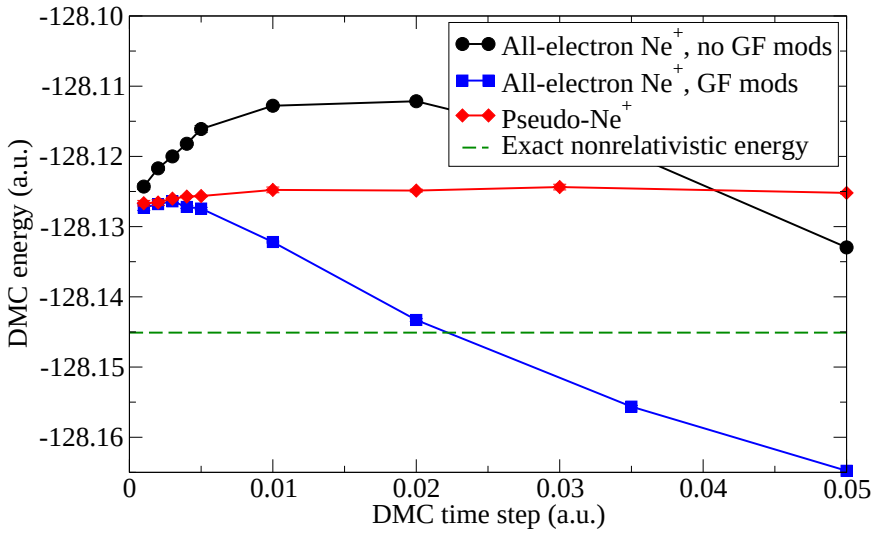


Timestep and nodal errors for Ne and Ne^+



Ne

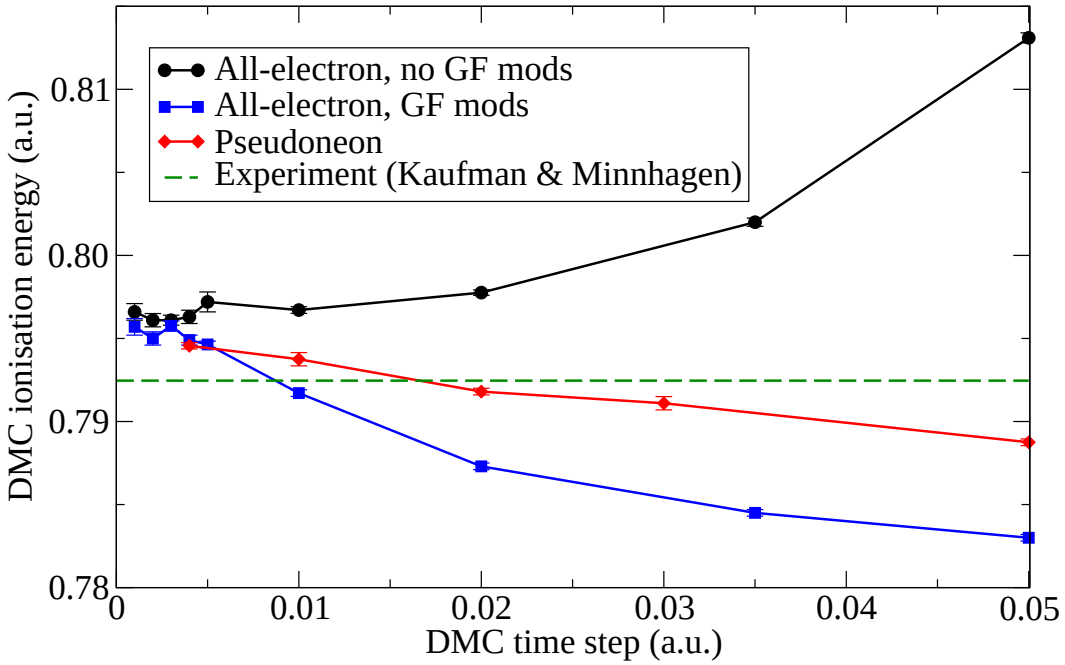
All-electron DMC: 97.1% of E_{corr}



Ne^+

All-electron DMC: 94.4% of E_{corr}

Timestep and nodal errors for Ne 1st ionization potential

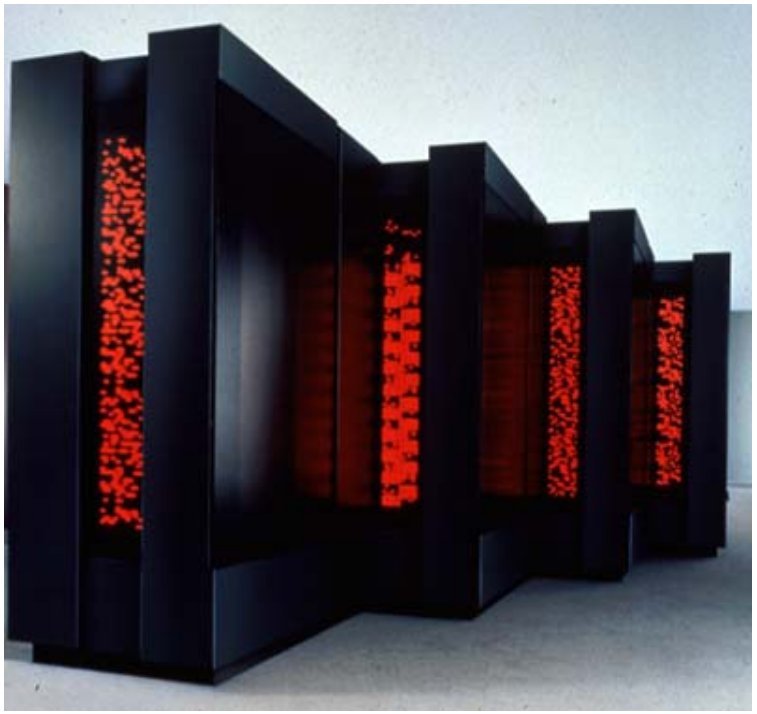


Experiment $= 21.56\,\text{eV}$

PBE-DFT error $= 0.91\,\text{eV}$
Hartree-Fock error $= -1.72\,\text{eV}$
Dirac-Fock error $= -1.74\,\text{eV}$
All-electron DMC error $= 0.11\,\text{eV}$
Pseudo DMC error $= 0.06\,\text{eV}$

Parallel computing

Simultaneous calculations performed by multiple processors



From the mid-1980s until 2004 computers got faster because of *frequency scaling* (more GHz). However, faster chips consume more power, and ever since power consumption (and consequently heat generation) became a significant concern, parallel computing has become the dominant paradigm in computer architecture, particularly with the advent of *multicore processors* - present even in most TTI laptops.

- We do not pretend that QMC is the cheapest technique in the world. Thus the study of anything other than simple systems inevitably requires the use of parallel computers.
- The biggest machines in the world now have more than a million processors. Some techniques (such as DFT) have difficulty exploiting more than a thousand processors because of the large amount of interprocessor communication required. This leads to our third scaling question:

How does QMC scale with the number of processors?

And consequently, how many processors can we successfully exploit?

Parallel computing

Simultaneous calculations performed by multiple processors

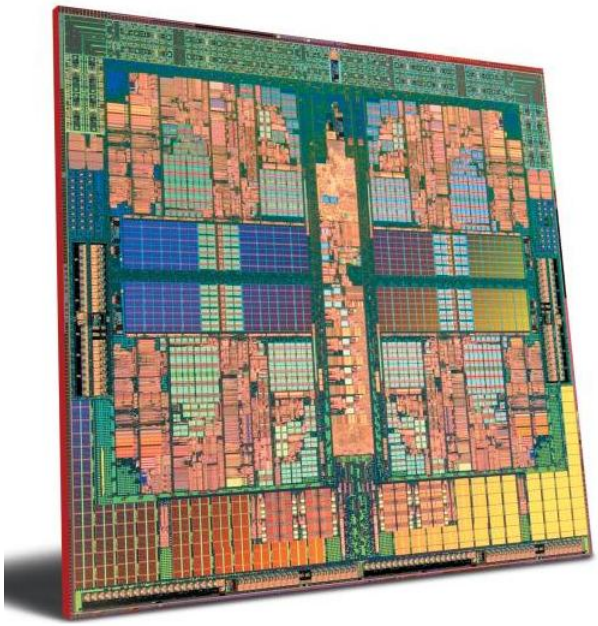
From the mid-1980s until 2004 computers got faster because of *frequency scaling* (more GHz). However, faster chips consume more power, and ever since power consumption (and consequently heat generation) became a significant concern, parallel computing has become the dominant paradigm in computer architecture, particularly with the advent of *multicore processors* - present even in most TTI laptops.

- We do not pretend that QMC is the cheapest technique in the world. Thus the study of anything other than simple systems inevitably requires the use of parallel computers.
- The biggest machines in the world now have more than a million processors. Some techniques (such as DFT) have difficulty exploiting more than a thousand processors because of the large amount of interprocessor communication required. This leads to our third scaling question:

How does QMC scale with the number of processors?

And consequently, how many processors can we successfully exploit?

Increasing complexity and new terminology: CPUs



AMD quad-core processor

In the old days (when we originally wrote CASINO) parallel machines were quite 'simple' things. That is, each computing unit (usually referred to as a 'node' or a 'processor') ran a separate copy of the program, and each had its own local memory.

Nowadays, things are more complex. A computer may have multiple nodes. And those nodes contain multiple sockets. And the processors in those sockets contain multiple (CPU) cores. The memory architecture is also more complex.

Node: a printed circuit board of some type, manufactured with multiple empty *sockets* into which one may plug one of a family of processors.

Processor: this is the object manufactured e.g. by Intel or AMD. Generally there are 'families' of processors whose members have differing core counts, a wide range of frequencies and different memory cache structures. One cannot buy anything smaller than a processor.

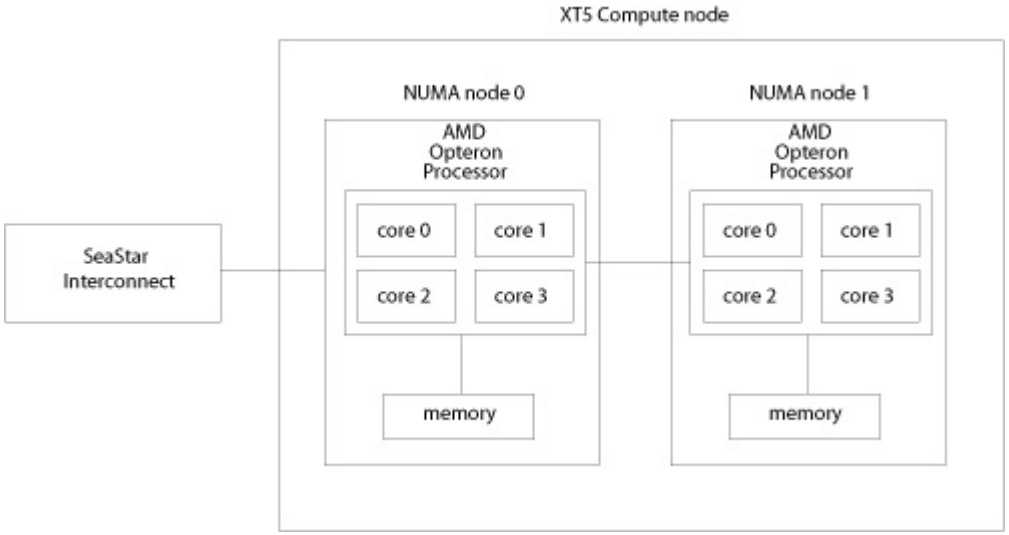
Core: the cores within the processor perform the actual mathematical computations. A core can do a certain number (typically 4) of FLOPs or FLoating-point OPerations every time its internal clock ticks. These clock ticks are called cycles and measured in Hertz (Hz). Thus a 2.5-GHz processor ticking 2.5 billion times per second and capable of performing 4 FLOPs each tick is rated with a theoretical performance of 10 billion FLOPs per second or 10 GFLOPS.

Increasing complexity and new terminology: memory

In complex modern systems we also need to understand how the memory is accessed.

Distributed memory : each processor has its own local private memory.

Shared memory : memory that may be simultaneously accessed by multiple cores with an intent to provide communication among them or avoid redundant copies.



Modern machines containing 'compute nodes' such as this XT5 often have a *non-uniform memory architecture* ('NUMA'). That is a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors.

Typically we might use shared memory on a 'compute node' which is simultaneously and quickly accessible to all processor cores that are plugged into it. Data is sent between nodes using explicit MPI commands and - in this case - the slower SeaStar Interconnect.

With CASINO, shared memory allows one to treat much bigger systems. A particular problem occurs when using a blip basis; the blip coefficients for a large systems can take up many GB of memory (and this may exceed the amount locally available to each core). Thus we may have e.g. a node containing two 6-core processors i.e. 12 cores with a single copy of the blip coefficients in the shared memory available to all cores on that node.

State of the art: petascale computers

- A 'petascale' system is able to make arithmetic calculations at a sustained rate in excess of a sizzling *1,000-trillion operations per second* (a 'petaflop' per second).
- The first computer ever to reach the petascale milestone (in 2008) was the *Roadrunner* at Los Alamos shown above. It contained 122400 cores achieving a peak performance of 1.026 petaflops/s.
- One may consult the 'Top 500 Supercomputers' list at `www.top500.org` to see who and what is currently winning. Current fastest (June 2014) is *Tianhe-2* in China (Intel Xeon cluster, 3120000 cores, 54.9 petaflops/s).

A usable example: Titan

Titan is a Cray XK7 machine at Oak Ridge National Laboratory in Tennessee, USA. It has a peak performance of around 27.1 petaflops/s, and has 560640 AMD Opteron processor cores, making it the 2nd-fastest computer in the world (June 2014). Like all the best computers, it runs Linux and supports many Fortran compilers.

It is made of 35,040 XK7 compute nodes. Each such node contains one 16-core AMD Opteron processor and 32 GB of memory. Each node also has an NVIDIA Kepler GPU accelerator (codes differ in their ability to take advantage of these). Titan was the fastest computer in the world until very recently.

```
ssh -X mdt26@titan.ccs.ornl.gov

cd CASINO ; make Shm

runqmc -p 224256 --shmem=16 --walltime=6h30m
```

How is CASINO parallelized?

CASINO's parallel capabilities are implemented largely with *MPI* which allows communication between all cores on the system. A second level of parallelization useful under certain circumstances (usually when the number of cores is greater than the number of walkers) is implemented using *OpenMP* constructs, which functions over small groups of e.g. 2-4 cores.

MPI (Message Passing Interface) is a language-independent API (application programming interface) specification that allows processes to communicate with one another by sending and receiving messages. It is a *de facto* standard for parallel programs running on computer clusters and supercomputers, where the cost of accessing non-local memory is high.

Example: `call MPI_Reduce([input_data], [output_result], [input_count], [input_datatype],[reduce_function], ROOT, [User_communication_set], [error_code])`

By setting the 'reduce function' to 'sum', such a command may be used - for example - to sum a vector over all cores, which is required when computing averages.

OpenMP is an API that supports shared-memory multiprocessing. It implements *multithreading*, where the master 'thread' (a series of instructions executed consecutively) forks a specified number of slave threads and a task is divided among them. The threads run concurrently, with the runtime environment allocating threads to different cores. The section of code meant to run in parallel is marked with a preprocessor directive that causes the threads to form before the section is executed:

```
!$OMP parallel
...
!$OMP end_parallel
```

Scaling of QMC with number of processors : VMC

- Perfectly parallel - no interprocessor communication required during simulation.
- Each processor does independent random walk using different random number sequence - results are averaged at end of each block. Running for time T on P processors generates same amount of data as running for time TP on one processor.
- VMC should therefore scale to arbitrarily large number of processors.

Scaling of QMC with number of processors : optimization

Standard variance minimization

VMC stages perfectly parallel. In optimization, config set distributed between CPUs. Master broadcasts current set of optimizable parameters. Each CPU calculates local energy of each of its configs - reports energies (and weights, if required) to master. CPU time to evaluate local energies usually far exceeds comms time (reporting 1 or 2 numbers per config to master and receiving a few parameters at each iteration). Time to evaluate local energies increases with system size, whereas comms time independent of system size. Standard varmin essentially perfectly parallel.

Fast variance minimization for linear Jastrow parameters

VMC stages perfectly parallel. Optimization done on master - typically takes fraction of a second and is independent of system size. Varmin_linjas perfectly parallel.

Energy minimization

VMC stages perfectly parallel. For matrix algebra stages, configs divided evenly between CPUs, each of which separately generates one section of the full matrices. Full matrices then gathered on master, which does matrix algebra. Time for matrix algebra usually insignificant compared to time for VMC and matrix gen. Comms time typically at most a few percent of time per iteration. Overall, energy minimization very nearly perfectly parallel.

Scaling of QMC with number of processors : DMC

- In DMC, config population initially divided evenly between cores. Algorithm not perfectly parallel since populations fluctuates on each core; iteration time determined by the core with the largest population. Necessary to even up config population between cores *occasionally* ('load balancing').
- The best definition of '*occasionally*' turns out to be 'after every move', since this minimizes the time taken by the core with the largest number of configurations to finish propagating its excess population.
- From the CASINO perspective, what is a 'config' and how big is it? It is a list of electron positions, together with some associated wave function- and energy-related quantites. For the relatively big systems of interest, a config might be from 1-10kb in size, and up to around five of them might need to be sent from one processor to another. Thus messages can be up to 50kb in size (though usually they are much smaller).
- Transferring configs between cores is thus likely to be time-consuming, particularly for large numbers of cores. Thus there is a trade-off between balancing the load on each processor and reducing the number of config transfers.

Formal parallel efficiency

- Cost of propagating all configs in one iteration : $T_{\mathrm{CPU}} \approx A\frac{N^{\alpha}N_C}{P}$

  Here P is number of CPU cores, N_C is number of configs, N is number of particles, and $\alpha = 1$ (localized orbs and basis) or 2 (delocalized orbs, local basis). Add 1 to α for trivial orbs/large systems where determinant update dominates.

- Cost of load balancing : $T_{\mathrm{comm}} \approx B\sqrt{N_C P N^3}$

  Require $T_{\mathrm{CPU}} \gg T_{\mathrm{comm}}$ as DMC algorithm perfectly parallel in this limit.

- Ratio of load balancing to config propagation time :

$$\frac{T_{\mathrm{comm}}}{T_{\mathrm{CPU}}} = \frac{A}{B}\frac{P^{\frac{3}{2}}N^{\frac{3}{2}-\alpha}}{\sqrt{N_C}}$$

- For $\alpha > 3/2$ (which is true unless time to evaluate localized orbitals dominates), the fraction of time spent on comms falls off with system size.

- By increasing N_C fraction of time spent on comms can be made arbitrarily small, but, in practice number of configs per core limited by available MEMORY.

- Memory issue is the main problem for very large systems or very large number of cores, particularly when using a blip basis set.

Obvious ways to improve load balancing in CASINO

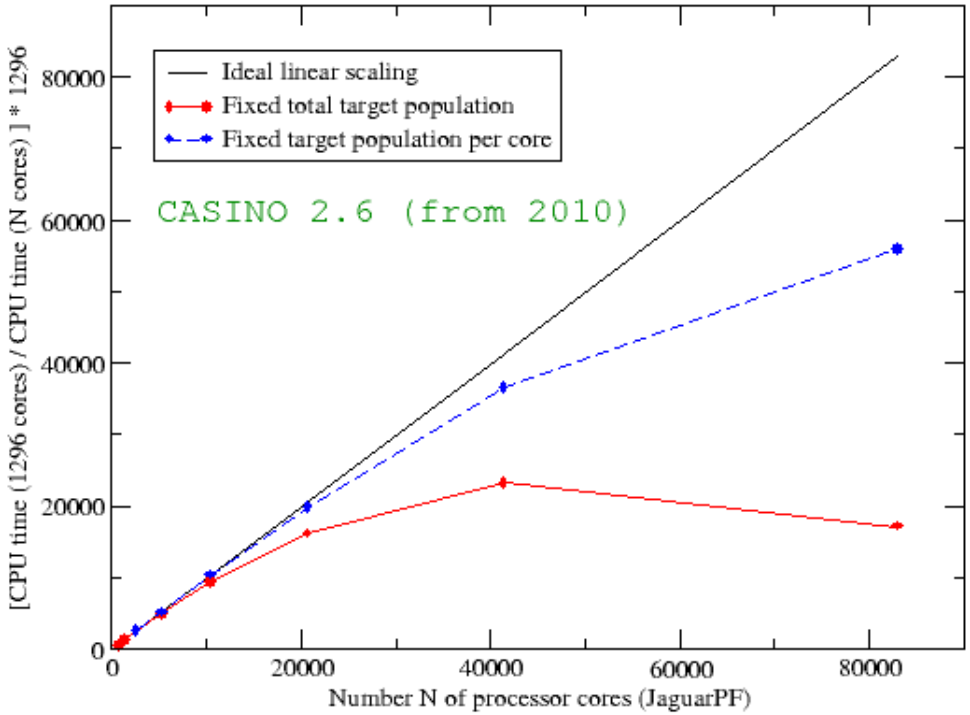
- Increase number of configs per core (without blowing the memory).
- Use weighted DMC (**lwdmc** keyword) to reduce branching (with the default weight limits of 0.5 and 2.0) and disable transfer or large arrays (such as inverse Slater matrices) between cores by using the **small_transfer** keyword.

Obvious ways to avoid blowing the memory in CASINO

- On architectures made up of shared memory nodes with multiple cores: allocate blips on these nodes instead of on each core (`make Shm` to enable this, then `runqmc --shmem`).
- Use OpenMP - extra level of parallelization for loops scaling with number of electrons. Define 'pools' of small numbers of cores (typically 2-4). Parallelisation over configs maintained over pools, but inside each pool work for each config is parallelized by splitting the orbitals over pools (this reduces necessary memory per core). Then, each core in the pool only evaluates the value of a subset of orbitals. That done, all cores within the pool communicate to construct the Slater determinants, which are evaluated again in parallel using the cores in the pool. Gives $\sim$1.5$\times$ speedup on 2 cores, $\sim$2$\times$ speedup on 4 cores.

  To use with CASINO, compile with '`make OpenMP`', then run with e.g. on a 4-core machine '`runqmc --nproc=2 --tpp=2`' where `tpp` means 'threads per process'. Can also run with both Shm and OpenMP (`make OpenmpShm` etc.).
- Use **single_precision_blips** keyword, the blip coefficients using single precision real/complex numbers, which will halve the memory required.

How does CASINO scale with old standard DMC algorithm?



Scaled ratio of CPU times in DMC statistics accumulation for various numbers of cores on Jaguar (Titan's predecessor) using the September 2010 version of CASINO 2.6. System: one H_2O molecule adsorbed on a 2D-periodic graphene sheet containing fifty C atoms per cell. For comparative purposes 'ideal linear scaling' (halving of CPU time for double the number of cores) is shown by the solid black line. Both blue and red lines show results for fixed sample size i.e. number of configs $\times$ number of moves [fixed problem size = 'strong scaling']. However, blue line has fixed target population of 100 configs per core (with an appropriately varying number of moves). Red line has fixed target population of 486000 (and constant number of moves) i.e. the number of configs per core falls with increasing number of cores (from 750 to around 5).

New tricks to effectively reduce T_{comm} to zero

Rendering the earlier formal analysis somewhat redundant, I discovered last year that with a few tricks one can effectively eliminate all overhead due to config transfers, and hence hugely improve the scaling (this is described in *Petascale computing opens new vistas for quantum Monte Carlo'*, by me, Mike Gillan and Dario Alfè, Psi-k Newsletter 'Scientific Highlight of the Month' Feb 2011).

The new algorithm involved:

(1) Analysis and modification of the procedure for deciding which configs to send between which pairs of cores when doing load balancing (the original CASINO algorithm for this originally scaled linearly with the number of cores – when you need it to be constant – yet this was never mentioned in formal analyses!).

(2) The use of *asynchronous, non-blocking* MPI communications.

- To send a message from one processor to another, one normally calls blocking `MPI_SEND` and `MPI_RECV` routines on a pair of communicating cores. 'Blocking' means that all other work will halt until the transfer completes.
- However, one may also use *non-blocking* MPI calls, which allow cores to continue doing computations while communication with another core is still pending. On calling the non-blocking `MPI_ISEND` routine, for example, the function will return immediately, usually before the data has finished being sent.

Decisions about config transfers: the redistribution problem

- At the end of every move we have a vector (of length equal to the number of cores) containing the current population of configs on each core.
- Relative to a 'target' population, some cores will have an excess of configs, some will have the right amount, and some will have a deficit.
- The problem is to arrange for a series of transfers between pairs of cores in the most efficient way such that each core has as close to the target population as possible. Here 'efficient' means the total number of necessary transfers and the size of those transfers is to be minimized.

OLD ALGORITHM: Requires repeated operations on the entire population vector, asking things like 'what is the location of the current largest element?' [Fortran: `maxloc(popvector)`]. This scales linearly with the number of cores, and if you're asking to find the largest element of a vector of length 1 million and you do it a million times it starts to take some serious time. Any benefit from obtaining the optimum list of transfers is swamped by the process of finding that optimum list.

MORAL: *the algorithm is perfectly reasonable for a routine written in the years when no-one could run on more than 512 cores; however, such things can come back and bite you in the petascale era.*

SOLUTION

- Partition the cores into 'redist groups' of default size 500 and contemplate transfers only within these groups. If e.g. one core has a deficit of 1,2,3,4. . . configs, then in a group of that size it is highly likely that some other core will have a surfeit of 1,2,3,4.. configs, etc. Thus efficiency in config transfers will hardly be affected by not considering the full population vector.
- To avoid imbalances developing in the group populations, the list of cores that belong to each group is changed at every iteration ('`redist_shuffle`').

Non-blocking asynchronous communication

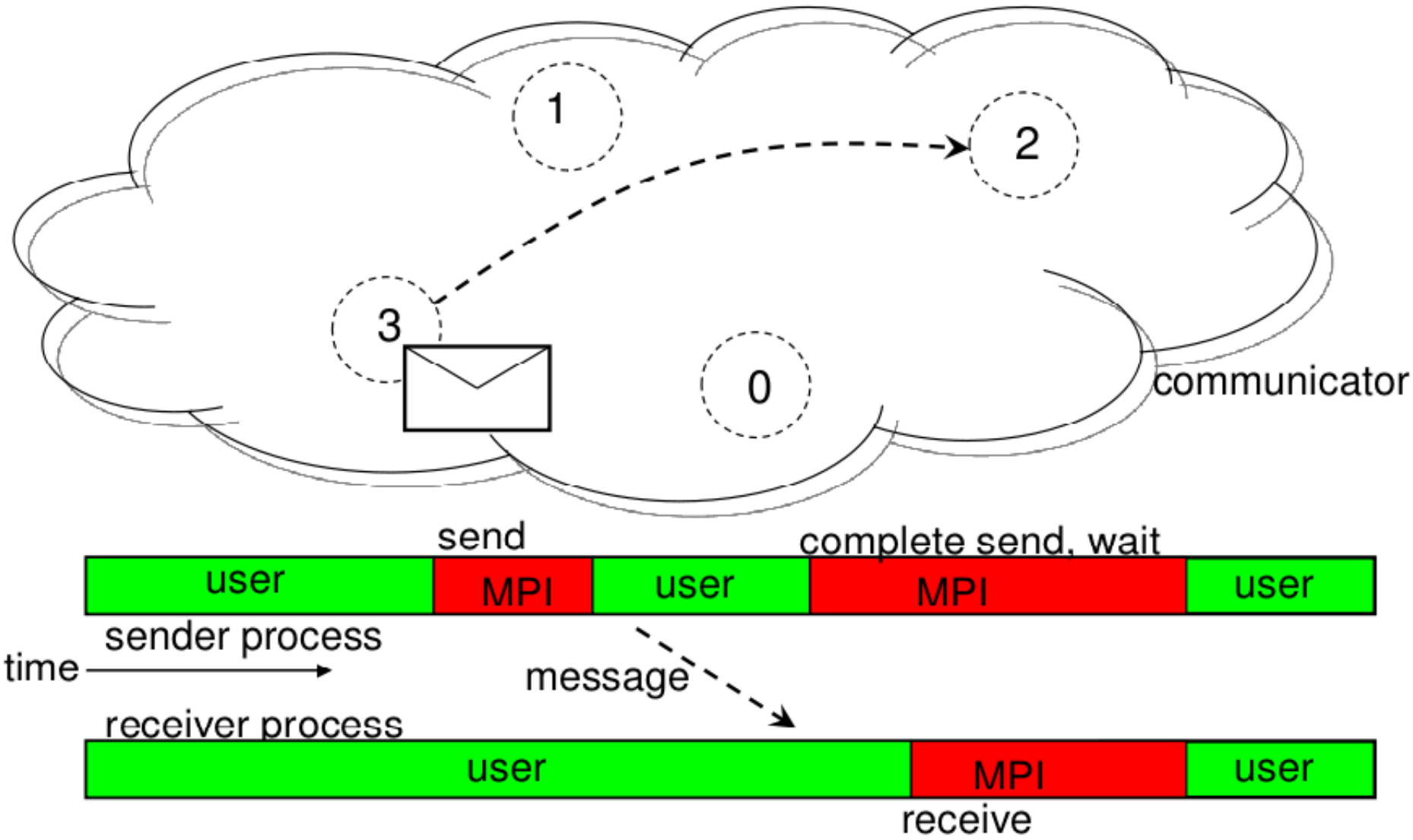
A communication call is said to be non-blocking if it may return before the operation completes (a *local* concept on either sender or receiver). A communication is said to be asynchronous if its execution proceeds at the same time as the execution of the program (a *non-local* concept).

Mode	Command	Notes	synchronous?
synchronous send	MPI_SSEND	Message goes directly to receiver. Only completes when receive begins.	synchronous
buffered send	MPI_BSEND	Message copied to a 'buffer'. Always completes regardless of receiver.	asynchronous
standard send	MPI_SEND	Either synchronous or buffered	both/hybrid
ready send	MPI_RSEND	Assumes the receiver is ready. Always completes regardless.	neither
receive	MPI_RECV	Completes when a message has arrived	

MPI also provides *non-blocking* send (MPI_ISEND) and receive (MPI_IRECV) routines. They return immediately, at the cost of you not being allowed to modify the sent vector/receiving vector until you execute a later MPI_TEST or MPI_WAIT call (or MPI_TESTALL/MPI_WAITALL for multiple communications) to check completion. In the meantime, the code can do some other work.

- Non-blocking routines allow separation of initiation and completion, and allow for the *possibility* of comms and computation overlap. Normally only one comm allowed at a time; non-blocking functions allow initiation of multiple comms, enabling MPI to progress them simultaneously.
- Non-blocking comms, when used properly, can provide a tremendous performance boost to parallel applications.

Non-blocking send operation



New DMC algorithm

MOVE 1

- Move all currently existing configs forward by one time step
- Compute the multiplicities for each config (the number of copies of each config to continue in the next move).
- Looking at the current populations of config on each processor, and at the current multiplicities, decide which configs to send between which pairs of cores, and how many copies of each are to be created when they reach their destination.
- Sending cores initiate the sends using non-blocking MPI_ISENDs; receiving cores initiate the receives using non-blocking MPI_IRECVs. All continue without waiting for the operations to complete.
- Perform on-site branching (kill or duplicate configs which require it on any given processor).

MOVE 2 AND SUBSEQUENT MOVES

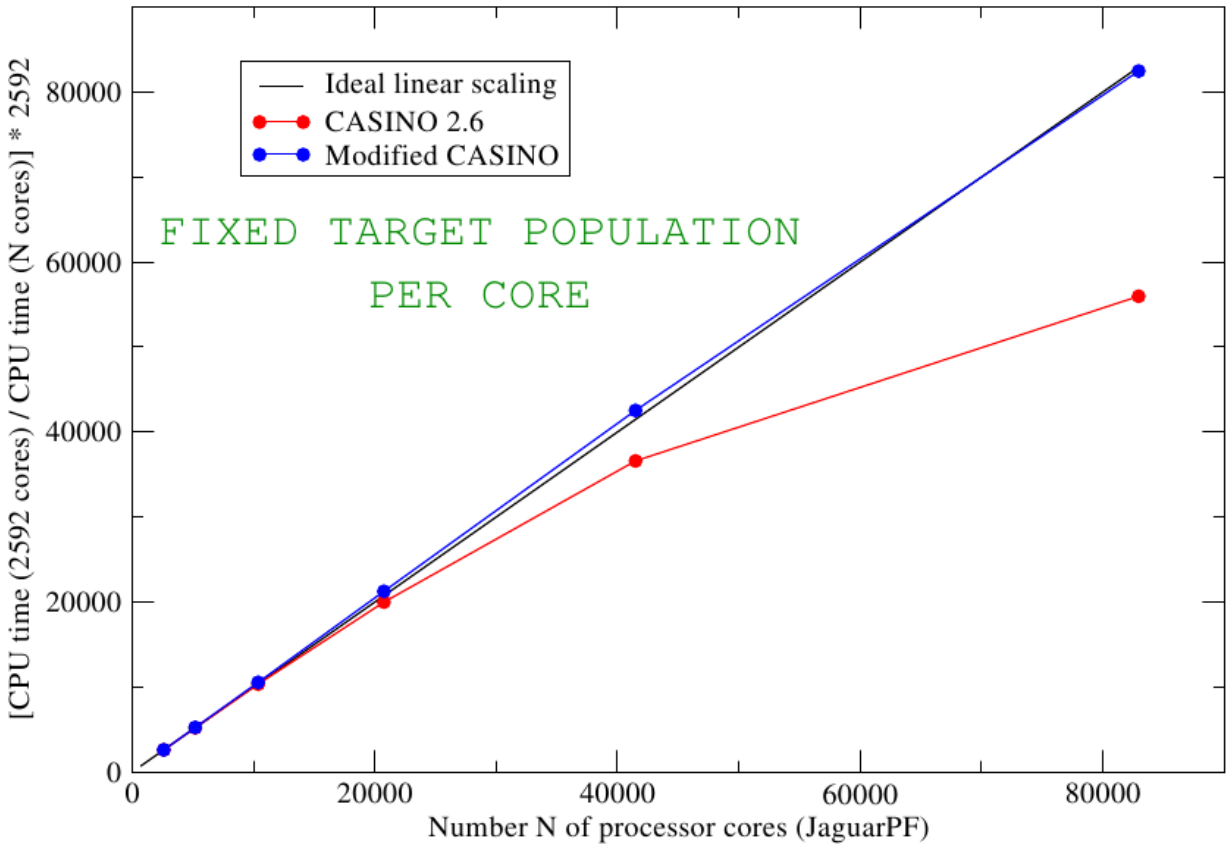
- Move all currently existing configs on a given processor by one time step (not including configs which may have been sent to this processor at the end of the previous move).
- Check that the non-blocking sends and receives have completed (they will almost certainly have done so) using MPI_WAITALL. When they have, duplicate newly-arrived configs according to their multiplicities and move by one time step.
- Compute the multiplicities for each moved config.
- Continue as before

Any improvement in the load-balancing time?

Number of cores	Time, CASINO 2.6 (s.)	Time, Modified CASINO (s.)
648	1.00	1.05
1296	3.61	1.27
2592	7.02	1.52
5184	18.80	3.06
10368	37.19	3.79
20736	75.32	1.32
41472	138.96	3.62
82944	283.77	1.04

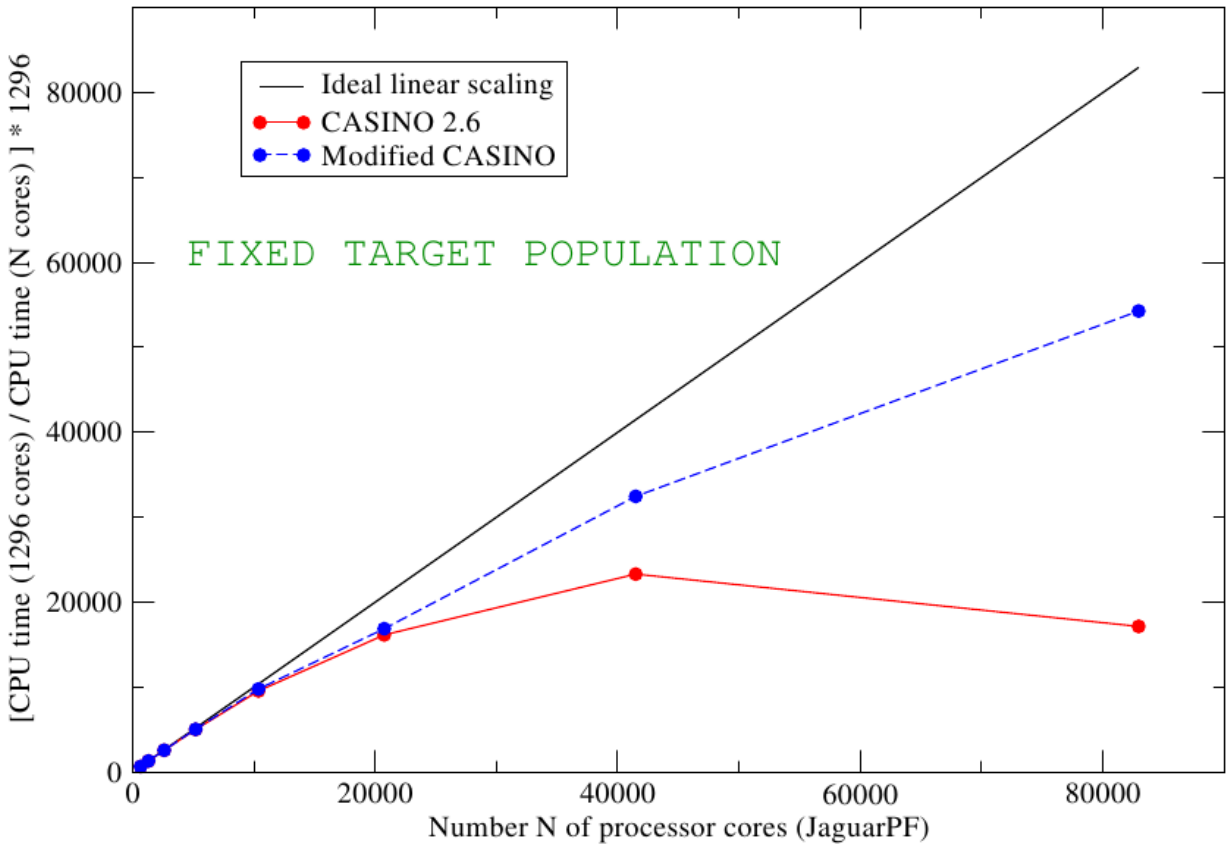
Table 1: *CPU time taken to carry out operations associated with redistribution of configs between cores in CASINO 2.6 (2010) and in my modified version, during one twenty-move DMC block for a water molecule adsorbed on a 2d graphene sheet.*

Perfect parallel efficiency..



Scaled ratio of CPU times in DMC statistics accumulation for various numbers of cores on Jaguar (Titan's predecessor) using both the September 2010 version of CASINO 2.6 (red line) and a later public release CASINO 2.8 (blue line). System: one H_2O molecule adsorbed on a 2D-periodic graphene sheet containing fifty C atoms per cell. For comparative purposes 'ideal linear scaling' (halving of CPU time for double the number of cores) is shown by the solid black line. In both cases there is a fixed target population of 100 configs per core (with an appropriately varying number of moves to maintain constant number of configuration space samples).

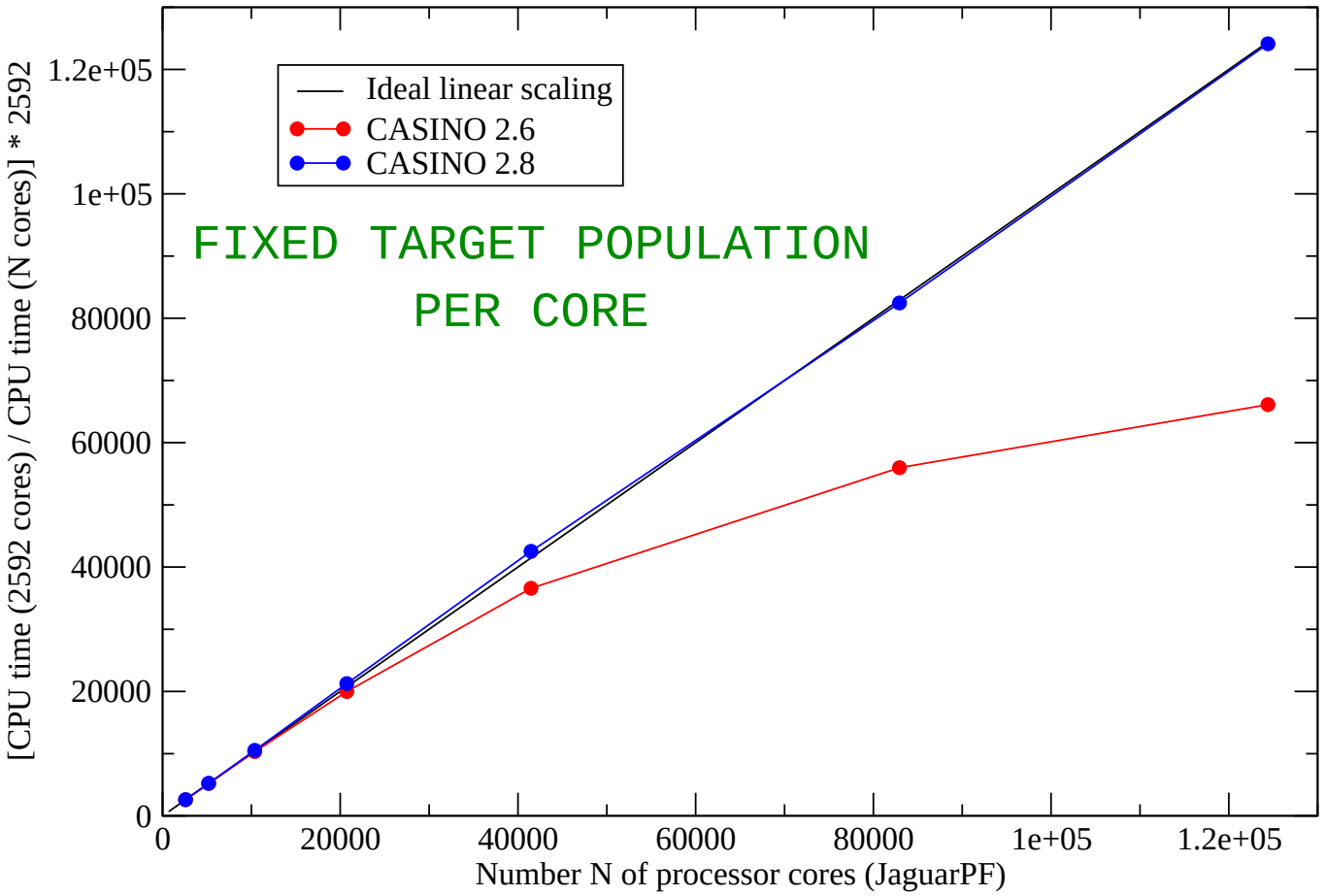
..if you give the processors enough to do



Similar graph for the same number of configuration space samples, but using a fixed target of 486000 for total config population and a fixed number of moves, rather than a fixed target per core.

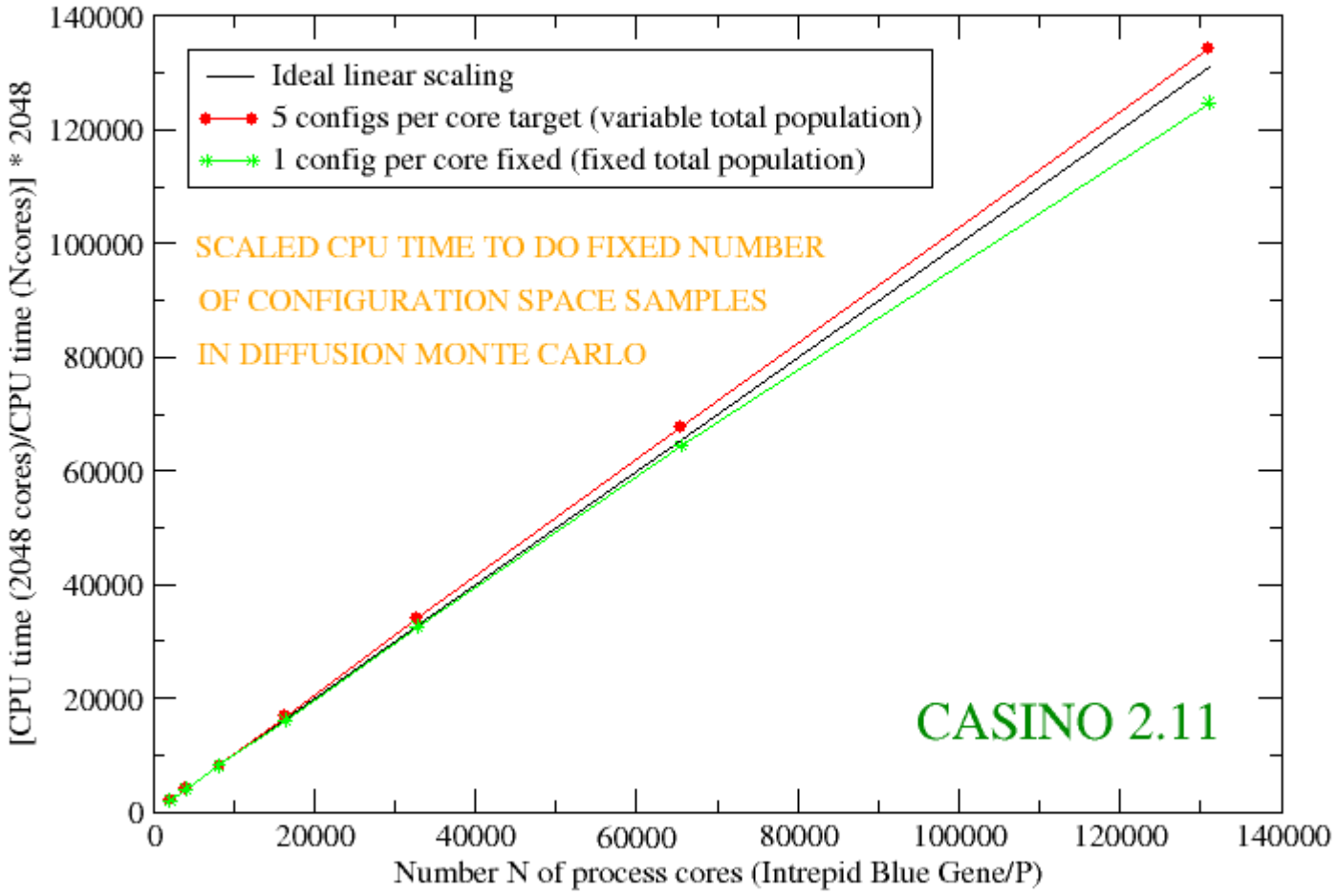
Note that fixing the total target population can introduce considerable inefficiency at higher core counts (since cores end up without enough work to do as the number of configs per node decreases). This graph should not be looked on as representing CASINO's general scaling behaviour. The inefficiency can generally be decreased by increasing the number of configs per core.

Can we push it to more than 100000 cores?

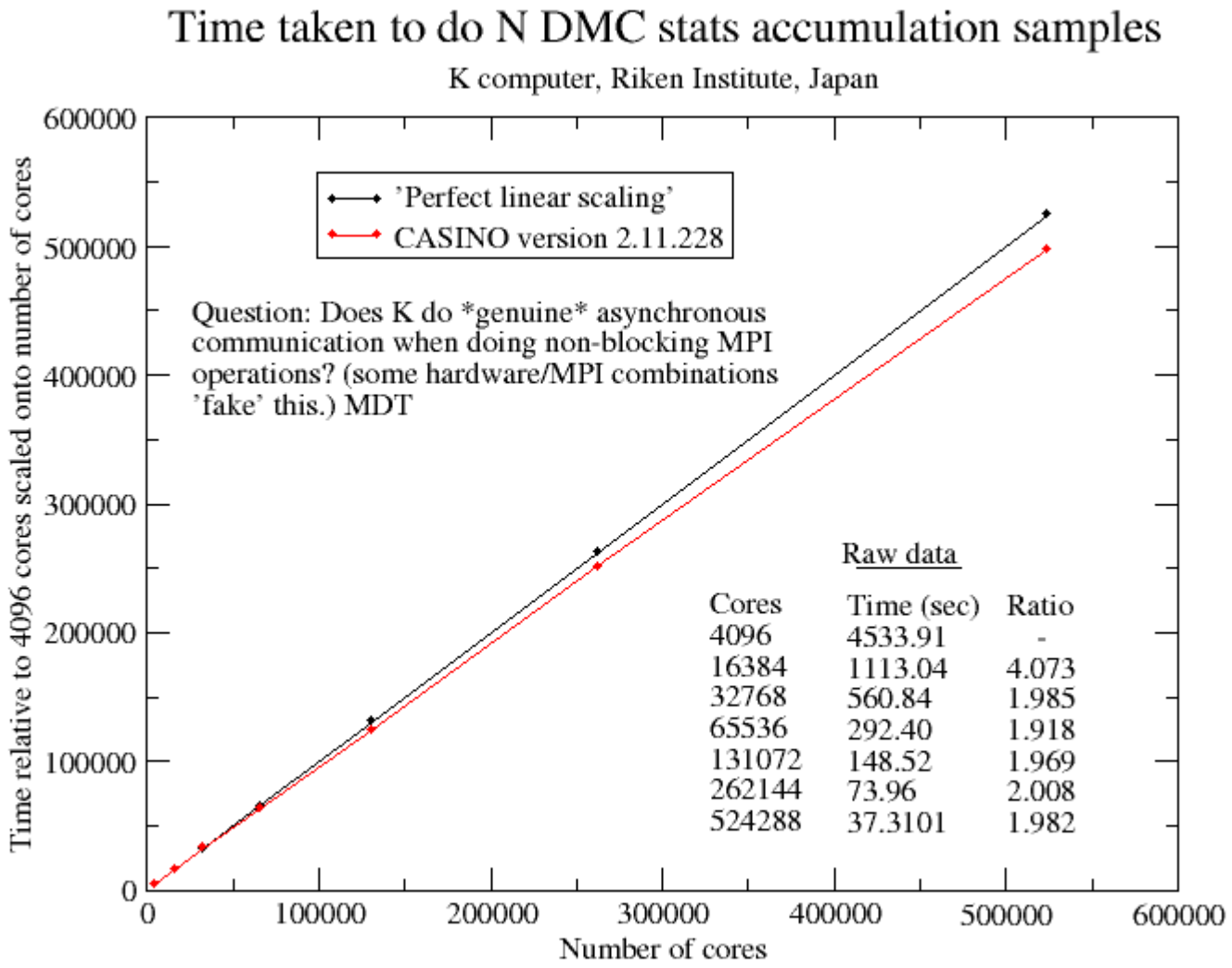


Yes! Not even the hint of a slowdown on 124416 cores.. Reasonable to assume we could use all 299008 cores of the Jaguar machine, if we could be bothered to sit through the queueing time.

..and up to 131072 cores on an IBM Blue Gene/P



..and up to 524288 cores on the Japanese K computer



How many cores can we exploit?

- Because QMC is a sampling technique then, for any given system, there is a maximum number of cores you can exploit if you insist that your answer has no less than some required error bar and that it has a minimum number of moves (so we can reblock the data).
- E.g. we require 1000000 random samples of the wave function configuration space to get the required error bar ϵ. Let's say we need at least 1000 sampling moves to accurately reblock the results. And let's say we have a 1000 processor computer. In that case only one config per node is required to get the error bar ϵ (even though the available memory may be able to accommodate many more than this).
- We now buy a 2000 processor machine. How do we exploit it to speedup the calculation? We can't decrease the number of moves, since then we can't reblock. It is wasteful to just run the calculation anyway, since then the error bar will become smaller than we require. We can split each config over two nodes, and use OpenMP to halve the time taken to propagate the configs, but let's say we find that OpenMP doesn't really work very well over more than two cores.
- How then do we exploit a 4000 processor machine? Answer - we can't. The computer is simply too big for the problem if you don't need the error bar to be any smaller.

A problem: including the effects of equilibration time

Important point that we have ignored so far: *the DMC equilibration time cannot be reduced towards zero by using more cores*. And when the equilibration time becomes comparable to the statistics accumulation time (which *is* reduced by using more cores) our scaling analysis will be affected. Thus:

- In equilibration, the RMS distance diffused by each electron needs to be greater than some characteristic length. This translates into a requirement for a *minimum number of moves* (which obviously depends on the DMC time step τ). In fact, with valence electron density parameter r_s:

$$N_{\mathrm{equil}}^{\mathrm{min}} > \frac{N_{\mathrm{elec}}^{\frac{2}{3}} r_s^2}{3\tau}$$

- If you use more processors, with a fixed number of configs per core, the equilibration time will be independent of processor count (and will be smaller the fewer configs per core you use).
- The time taken to accumulate the data with the required error bar (through M samples of the configuration space) will go down with increasing core count, with 'perfect linear scaling'.

Computers like Titan have maximum job times (typically 12 or 24 hours) and a time that you have to sit in the queue before a job starts. So you can define a (somehwat arbitrary) maximum time that you are prepared to wait for DMC equilibration to complete. Neil has a nice internal paper, where he defines this as 4 days, and thus concludes (depressingly) that

(1) It will be difficult to do DMC calculations with more than around 800 electrons (since one has to equilibrate for days regardless of the size of the computer).

(2) For an 800 electron system, the maximum number of cores worth using is 14400 (when calculating the total energy of the simulation cell) or er.. 36 (when calculating the energy per atom).

Cheaper equilibration: the preliminary DMC scheme

With a small config population, equilibration is a small fraction of the total DMC run time. When running on a large number of cores however, the configuration population is necessarily large, since each core must have at least one configuration on average; the equilibration time will likely be large.

An alternative procedure which can significantly reduce the expense of equilibration is Neil's preliminary DMC scheme (see section 13.10 of the CASINO manual):

- Instead of using VMC to generate all the N_{conf} required configs for the DMC calculation, use VMC to generate a much smaller number of configs $N_{\mathrm{conf}}^{\mathrm{small}}$ (at least 1000 to be large enough to avoid population control bias).
- Equilibrate the $N_{\mathrm{conf}}^{\mathrm{small}}$ configs on some small system without a delay-inducing queueing system.
- Run enough statistics accumulation on the equilibrated $N_{\mathrm{conf}}^{\mathrm{small}}$ in order to generate N_{conf} independent configs for use in the full DMC calculation. (Can also use the energies of these configs in the accumulation data..).

Example

Want to do 50000 core calculation, with 2 configs per core - require 100000 equilibrated configs.

Normally use 100000 samples of VMC wave function to provide initial configs to be equilibrated, then in order to equilibrate DMC we need to run - say - 750 steps on each of the 100000 configs (2 configs per core).

Instead use 1000 samples of VMC wave function on (say) 500 cores (2 configs per core), then run each of those for 750 steps to achieve DMC equil - 100 times faster than before because I have far fewer configs.Turn on stats accumulation. Run each of the 1000 samples for $100 \times T_{\mathrm{corr}}$ moves to get 100000 independent samples in total. Takes roughly the same time as equilibration. Thus overall around 50 times faster than equilibration done on 100000 configs.

Drawback: more costly in human time as one has to run equilibration on small computer then transfer to Titan or whatever..

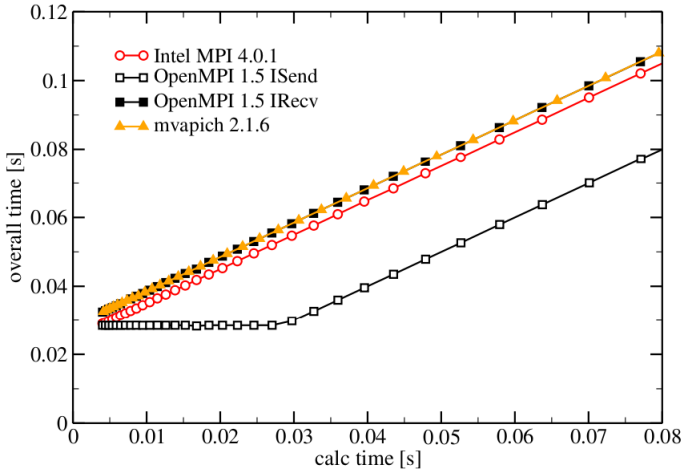
Is non-blocking communication really asynchronous?

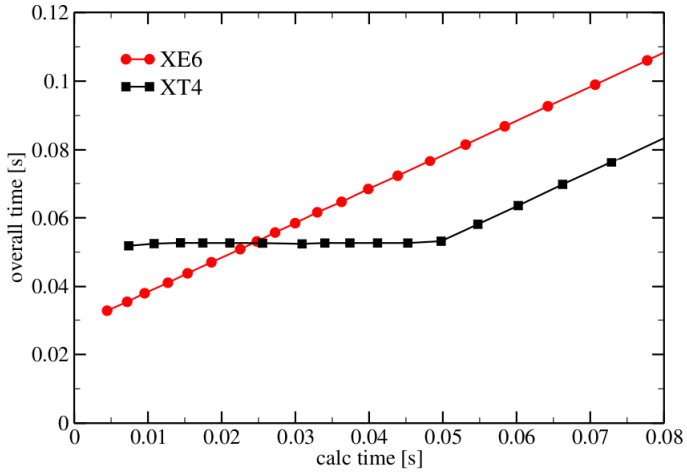
Not necessarily! MPI standard doesn't *require* non-blocking calls to be asynchronous. Two problems:

(1) Hardware may not support asynchronous communication. Some networks provide communication co-processors that progress message passing regardless of what application program does (e.g. Infiniband, Quadrics, Myrinet, Seastar and some forms of TCP that have offload engines on the NIC). Then communication can be started by the computation processor which in turn gives task of sending data over the network to the communication processor.

(2) Unfortunately, even if the hardware supports it, people implementing MPI libraries may not bother to code up truly asynchronous transfers (since the standard allows them not to!). MPI progress is actually performed within the MPI_TEST or MPI_WAIT functions. This is cheating!

Test: initiate MPI_IRECV with large 80Mb message, then do some computation for a variable amount of time. If comms really do overlap with computation then total runtime will be constant so long as computation time is smaller than comms time.
Ref: Hager *et al*. `http://blogs.fau.de/hager/files/2011/05/Hager-Paper-CUG11.pdf`





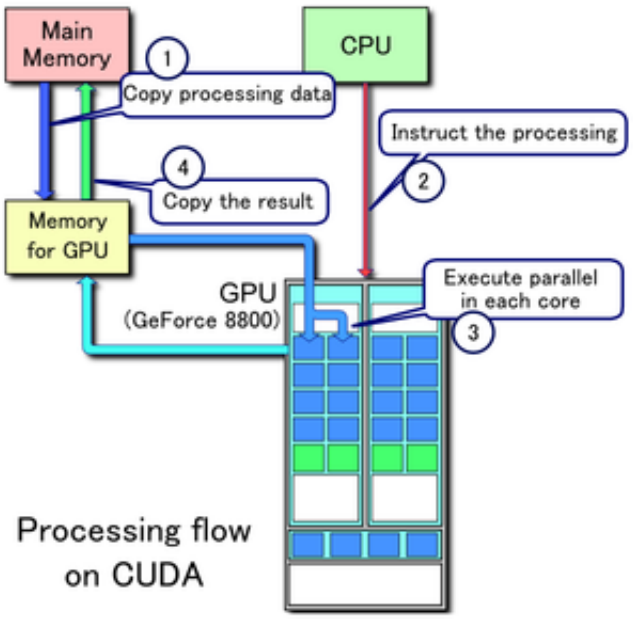
If your MPI doesn't provide true asynchronous progress, then some form of periodic poll through a MPI_TESTALL operation may be required to achive optimal performance. Can also overlap computation and comms via mixed-mode OpenMP/MPI - use dedicated communication thread.

The future? GPUs

- A GPU (graphics processing unit) is a specialized processor designed to answer the demands of real-time high-resolution 3D-graphics compute-intensive tasks (whose development was driven by rich nerds demanding better games). They are produced by big companies such as Nvidia and ATI.
- Decent modern GPUs in machines with only a few CPU-cores are engineered to perform *hundreds* of computations in parallel. In recent years there has been a trend to use this additional processing power to perform computations in applications traditionally handled by the CPU.
- Modern GPUs have evolved into highly parallel multicore systems allowing very efficient manipulation of large blocks of data. This design is more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel.
- To do general purpose computing on GPUs, people originally had to 'pretend' to be doing graphics operations and learn things like OpenGL or DirectX (and so very few people bothered). Nowadays, new architectures such as CUDA allow people to operate GPUs using more familiar programming languages, and their use is booming.
- In fact, it might be said, that computing is evolving from 'central processing' on the CPU to 'co-processing' on the CPU and GPU. Of of the top 500 supercomputers in 2014, a total of 62 systems on the list are using accelerator/co-processor technology, up from 53 in 2013, 57 in 2012, and 39 in 2011). As we have seen, the Titan system is one of them.

The highly parallel nature of Monte Carlo algorithms suggest that CASINO might benefit considerably from GPU co-processing. However, whenever we have tried it, we have only succeeded in reducing the speed of the code by an order of magnitude. It only appears to be possible to get any benefit by doing some sort of ground-up rewrite, which none of us seem prepared to do.

Programming for Nvidia GPUs: CUDA



CUDA (Compute Unified Device Architecture) is a parallel computing architecture developed by Nvidia. It is the computing engine in Nvidia GPUs that is accessible to software developers through variants of industry-standard programming languages. Programmers typically use 'C for CUDA' (C with Nvidia extensions and certain restrictions) to code algorithms for execution on the GPU.

CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs so that they become accessible for computation like CPUs. Unlike CPUs however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly.

CASINO is written in Fortran95, so we would like to code in Fortran directly, rather than the officially-supported C. Fortunately (see e.g. `www.pgroup.com/resources/cudafortran.htm`) there are available third-party solutions such as PGI CUDA Fortran, so one can do things like this:

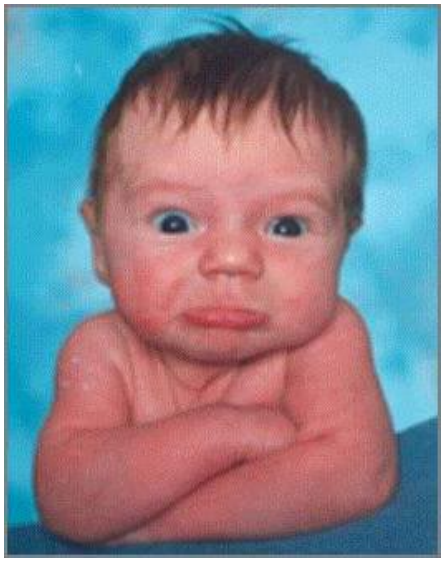
```
REAL :: a(m,n)                ! a instantiated in host memory
REAL,DEVICE :: adev(m,n)      ! adev instantiated in GPU memory
adev = a                      ! Copy data from a (host) to adev (GPU)
a = adev                      ! Copy data from adev (GPU) to a (host)
```

Consider also 'Accelerator compilers' which allow you to program GPUs simply by adding compiler directives to existing Fortran and recompiling with special flags (`www.pgroup.com/resources/accel.htm`). Doesn't work very well yet, in my experience!

Conclusions

- In general it seems to be the case that, following my modifications, CASINO is now linear scaling with the number of cores providing the problem is large enough to give each core enough work to do.
- This should normally be easy enough to arrange, and if you find yourself unable to do this, then you don't need a computer that big.
- One must pay attention to issues related to equilibration time if you want to use CASINO for big systems on very large numbers of cores.. Techniques have been developed to address this question, but further work is required.
- On typical machines like Titan, very large priority is given to jobs using large numbers of cores (where 'large' means greater than around 40000). Being allowed to use the machine in the first place increasingly means being able to demonstrate appropriate scaling of the code beforehand. CASINO can do this; many, even most, other techniques cannot.
- People need to start rewriting their codes to use GPUs, if they haven't already.
- Massively parallel machines are now increasingly capable of performing highly accurate QMC simulations of the properties of materials that are of the greatest interest scientifically and technologically.

I don't have access to a petascale computer (sulk..)



So you have three options:

(1) Don't do QMC calculations on very big systems.

(2) Wait for 10 years until everyone has a petascale computer under their desk.

(3) Unless you happen to be North Korean or Iranian or otherwise associated with the Axis of Evil, apply for some time on one. I did. You might consider, for example:

The INCITE program

`www.doeleadershipcomputing.org/guide-to-hpc/`

The European DEISA program

`www.deisa.eu`

Scaling problems : general conclusions

Many atoms

- 'Linear scaling' QMC algorithms have been implemented (which scale as the square of the system size to get the total energy per cell to within a given error bar, or independently of the system size to get the total energy per atom, but never linearly..). One should therefore describe QMC as 'quadratic scaling'.

Many protons

- All-electron DMC calculations are stable up to arbitrary atomic number given patience and a large computer.
- CPU time for fixed error bar seems to scale as $\approx Z^{5.5}$ for low atomic numbers as predicted, but appears to improve to $\approx Z^{4.5}$ for heavier atoms. Probable residual systematic timestep error for core electrons in heavier atoms but this should cancel in energy differences.

Many processors

- Only DMC not trivially parallelizable. However, recent algorithmic improvements mean that even DMC now has a perfect linear scaling with the number of cores, subject to the caveats already mentioned.