Parallel Implementation of the ONETEP Approach

Nicholas D.M. Hine

Thomas Young Centre
Department of Materials and Department of Physics
Imperial College London

15th April 2010

Outline

Outline

Linear Scaling DFT in ONETEP

Goals:

- Construction of sparse Hamiltonian and Overlap matrices in linear-scaling computational effort
- Linear-scaling optimisation of density matrix in insulators
- *In-situ* optimisation of minimal set of localised functions
- Systematic convergence of E_T with respect to size of basis (as for plane-waves)
- Highly efficient parallelisation: Speedup on P processors to remain $\propto P$ up to large P

Nested Loop Optimisation

Outer Loop: Optimise NGWFs

1. Initialise $\{\phi_\alpha(\mathbf{r})\}$ to atomic orbitals
2. Evaluate $\min_{\mathbf{K}} E(\mathbf{K};\{\phi_\alpha\})$

Inner Loop: Optimise Density Kernel

1. Calculate $E = \mathrm{Tr}\mathbf{K}.\mathbf{S}$
2. Calculate $\partial E/\partial K^{\alpha\beta}$ & search direction
3. Take LNV CG step
4. Exit when $\mathbf{K}$ converged, else go to (1)

3. Calculate NGWF gradient $\partial E/\partial\phi_\alpha(\mathbf{r})$ & search direction
4. Take NGWF CG step
5. Exit when NGWFs converged, else go to (2)

Implementation

- Standard F90
- MPI communications
- Libraries required for:
  - FFTs (FFTW, MKL, etc)
  - linear-algebra (LAPACK)
  - Optionally, parallel linear algebra (ScaLAPACK)
- 4 Authors, further 5-10 contributors, $> 100,000$ lines of code
  Hence require:
  - Standardisation of coding style
  - Clear structure & commenting
  - Consistent standards for internal data representation

Internal Data Formats

Sets of functions

NGWFs $\{\phi_\alpha\}$ as psinc function coeffs: Represented by a `FUNCTION_BASIS`, indexing psinc coefficients stored in an array of reals.
Nonlocal Projectors $\{\chi_i\}$ in reciprocal space: Represented by a `FUNCTION_BASIS` and a `PROJECTOR_SET` storing reciprocal space projectors.

Sparse Matrices

eg $S_{\alpha\beta} = \langle\phi_\alpha|\phi_\beta\rangle$ $\quad$ $S_{\alpha\beta} \neq 0$ only if $\phi_\alpha(\mathbf{r})$ and $\phi_\beta(\mathbf{r})$ overlap.
Block-indexed sparse matrices, of pre-determined sparsity patterns (eg K, S, H, KS, KSK...): Represented by a `SPAM3` type.

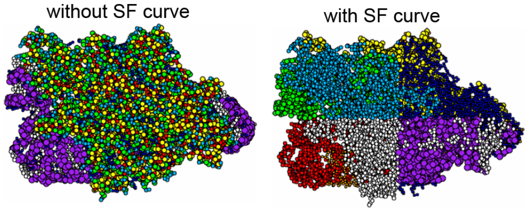
Whole Simulation Cell grid arrays

Used to represent density, potential etc, Distributed over parallel nodes in slabs.

Outline

Space-Filling Curve

Orders atoms according to proximity, for distribution over nodes

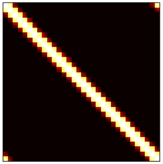


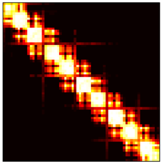
`space_filling_curve:` `T` by default.
Turn it off if you think you can do better by hand (unlikely!)

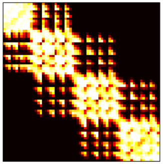
Sparse Matrices

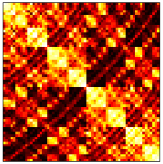
SFC ensures that the nonzero elements of sparse matrices are clustered near the diagonal.

Density of nonzero elements in ~ 4000 atom systems


C Nanotube


DNA Strand


GaAs Nanorod


Si Crystal

Need to make matrix algebra efficient, scalable and flexible over wide range of matrix filling

Sparse Matrices

Sparse algebra system works by dividing matrices by rows and columns into ‘segments’.

Segment

Rows associated with all atoms of node j in columns associated with atoms of node i. Stored on node i.

Segments are ‘dense’ if they have nonzero element filling fraction $\eta > \eta_c$. Controlled by `dense_threshold - default 0.3`.
Segments are ‘sparse’ if $0 < \eta \leq \eta_c$. Sparse segments divided into ‘blocks’

Block

Rows associated with atom J in columns associated with atom I. Nonzero blocks within segment (j, i) on node i are listed in ‘index’

Segments with $\eta = 0$ are ‘blank’ and are ignored.

Sparse Matrices

Sparse algebra system works by dividing matrices by rows and columns into ‘segments’.

Segment

Rows associated with all atoms of node j in columns associated with atoms of node i. Stored on node i.

Segments are ‘dense’ if they have nonzero element filling fraction $\eta > \eta_c$. Controlled by `dense_threshold - default 0.3`.
Segments are ‘sparse’ if $0 < \eta \leq \eta_c$. Sparse segments divided into ‘blocks’

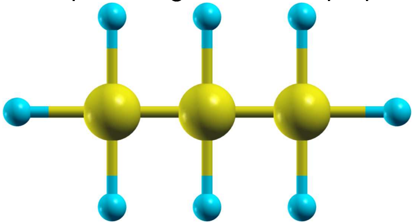
Block

Rows associated with atom J in columns associated with atom I.
Nonzero blocks within segment (j, i) on node i are listed in ‘index’

Segments with $\eta = 0$ are ‘blank’ and are ignored.

Sparse Matrix Algebra

Example: imagine a linear propane-like molecule:

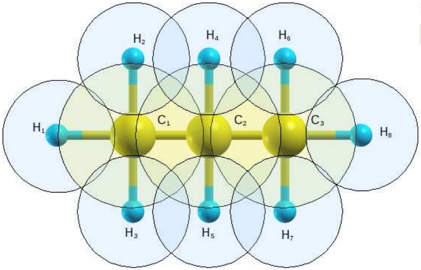


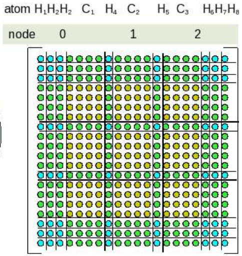
Distribute atoms over 3 cores...

4 NGWFs per C, 1 per H $\Rightarrow$7,6,7 NGWFs on nodes 0,1,2

Sparse Matrix Algebra

Construct (dense) overlap matrix for these NGWFs with $O(N^2)$ nonzero elements.

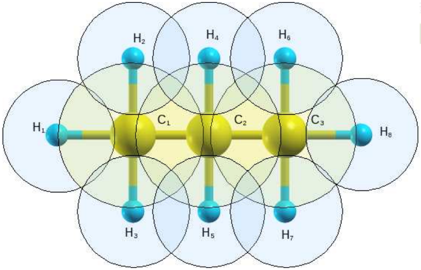


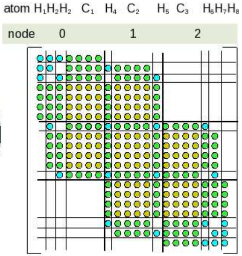


NB: unrepresentatively small NGWF spheres!

Sparse Matrix Algebra

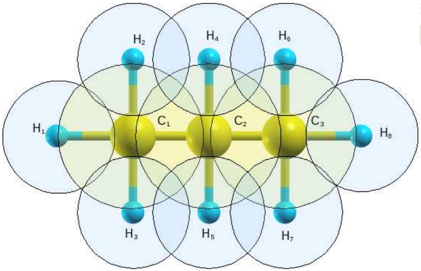
Apply sparsity: remove elements $\alpha\beta$ where ϕ_α and ϕ_β do not overlap

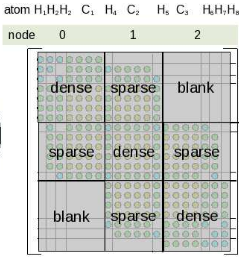




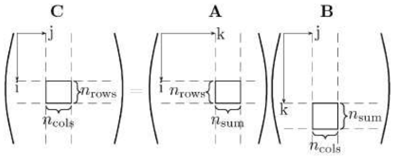
Sparse Matrix Algebra

Apply segmentation: segments with high filling are dense, segments with zero filling are blank





Sparse Matrix Algebra



Product of sparse matrices **A**, **B**:

$$\mathbf{C} = \mathbf{A}.\mathbf{B}$$

By elements:

$$C^{\alpha}{}_{\beta} = \sum_{\gamma} A^{\alpha\gamma} B_{\gamma\beta}$$

Or equivalently by segments:

$$\mathbf{C}_{ij} = \sum_{k} \mathbf{A}_{ik}.\mathbf{B}_{kj}$$

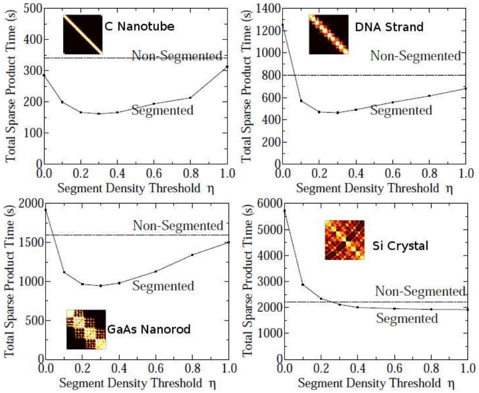
Node j stores columns $\mathbf{C}_{ij}$, so parts of $\mathbf{A}_{ik}$ must be sent from node k to node j. However, if $\mathbf{B}_{kj}$ has no nonzero elements, then $\mathbf{A}_{ik}.\mathbf{B}_{kj}$ does not contribute to $\mathbf{C}_{ij}$, so node k need not send anything to node j.

Principle: Minimise communication! Only send nonzero elements and index entries which contribute to $\mathbf{C}_{ij}$

Sparse Matrix Algebra

Segmentation allows speedup in two ways:

- Minimisation of communication and indexing overhead
- Dense matrix algebra (LAPACK) used for dense segments



(comparing segmented and nonsegmented code, both minimal

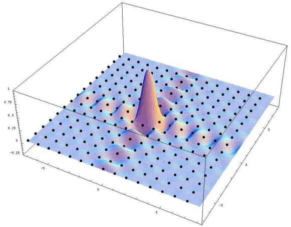
Outline

NGWFs

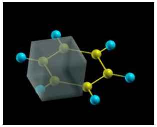
NGWFs $\phi_\alpha(\mathbf{r})$ represented with basis of 'psinc' functions on grid specified by E_{cut}

$$\phi_\alpha(\mathbf{r}) = \sum_\mu D_i(\mathbf{r}) c_{i\alpha}$$

where $$D_i(\mathbf{r}) = \frac{1}{N}\sum_p e^{i\mathbf{k}_p(\mathbf{r}-\mathbf{r}_i)}$$

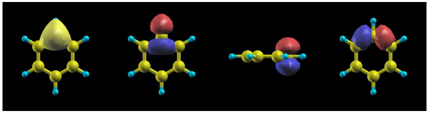


NB: zero at all grid points $\mathbf{r}_{j\neq i}$!



Psinc coefficients stored in 'parallepiped domains' (ppds): little boxes of grid points.

NGWFs initialised to Atomic Orbital-like form, then optimised



Managing Functions in O(N)

For a general set of functions $\{f_\alpha(\mathbf{r})\}$ distributed over nodes
Almost all operations involving f_α's can be put in one of the following forms:

$$A_\alpha(\mathbf{r}) = \sum_\beta M^{\alpha\beta} f_\beta(\mathbf{r}) \qquad \text{function sum}$$

$$O_{\alpha\beta} = \left\langle f_\alpha | \hat{O} | f_\beta \right\rangle \qquad \text{integral}$$

$$R^\alpha{}_\beta = \sum_\gamma P^{\alpha\gamma} Q_{\gamma\beta} \qquad \text{sparse algebra}$$

Examples (1) NGWFs

For the set of NGWFs $\{\phi_\alpha(\mathbf{r})\}$, examples include

$$n(\mathbf{r}) = \sum_\alpha \phi_\alpha(\mathbf{r}) \sum_\beta K^{\alpha\beta} \phi_\alpha(\mathbf{r}) \qquad \text{density}$$

$$T_{\alpha\beta} = \quad \left\langle \phi_\alpha \middle| -\frac{1}{2}\nabla^2 \middle| \phi_\beta \right\rangle \qquad \text{kinetic energy}$$

$$K^{\alpha\beta} = 3L^{\alpha\gamma} S_{\gamma\delta}\, L^{\delta\beta} - 2L^{\alpha\gamma} S_{\gamma\delta}\, L^{\delta\varepsilon}\, S_{\varepsilon\zeta}\, L^{\zeta\beta} \qquad \text{DM purification}$$

Examples (2) Nonlocal Projectors

For the set of Nonlocal pseudopotential projectors $\{\chi_i(\mathbf{r})\}$, examples of these methods include

$$\frac{\partial E_{nl}}{\partial \phi_\alpha(\mathbf{r})} = \sum_i \sum_\beta \left(\frac{\langle \chi_i | \phi_\beta \rangle K^{\alpha\beta}}{D_i} \right) \chi_i(\mathbf{r}) \qquad \text{nonlocal psp gradient}$$

$$P_{\alpha i} = \quad \langle \phi_\alpha | \chi_i \rangle \ ; \quad R_{j\beta} = \quad \left\langle \chi_j | \phi_\beta \right\rangle \qquad \text{‘s-p’,‘p-s’ overlaps}$$

$$V^{nl}_{\alpha\beta} = \sum_i \frac{P_{\alpha i} R_{i\beta}}{D_i} \qquad \text{nonlocal matrix}$$

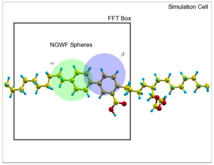
FFT Box

Moving 'FFT box' centred on ϕ_α — allows use of reciprocal-space methods

Integrals and function sums are $O(1)$ per function:

- Functions are strictly localised
  $\Rightarrow$each one overlaps $O(1)$ others
- All calculations performed in 'FFT box' centered onϕ_α
  $\Rightarrow$effort of operation does not grow with system size

Hence whole operation, for $O(N)$ functions, is $O(N)$.



Batch System

However, ϕ_α will overlap ϕ_β's stored on other nodes.

- To avoid recommunicating ϕ_β more than required, calculate a *batch* of FFTboxes, eg $A_\alpha(\mathbf{r})$ for $\alpha = \{11, 12..., 20\}$ at a time

$$A_\alpha(\mathbf{r}) = \sum_\beta M^{\alpha\beta} f_\beta(\mathbf{r}) \qquad \text{[re-uses } f_\beta(\mathbf{r}) \text{ up to } N_{batch} \text{ times]}$$

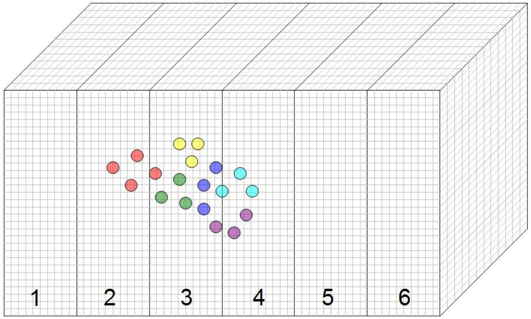
- Batch sizes as large as possible given available memory.
- Routines designed to hide communication behind computation
- *Asynchronous* receive operations avoid synchrony, which emphasises inefficiency due to load balancing

Outline

Whole-Cell Arrays

Density $n(\mathbf{r})$, potential $V(\mathbf{r})$ defined everywhere in cell

At worst $O(N)$ memory to store on grid

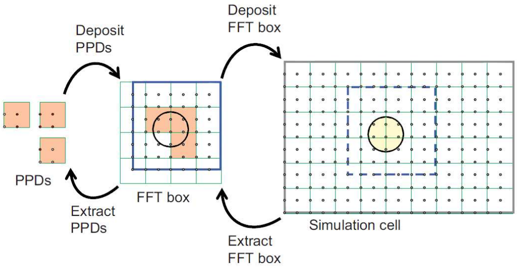


Real space grids eg $n(\mathbf{r})$ parallelised over '12' slabs: $n(:,:,s_3^i : e_3^i)$ on node i.

Recip space grids eg $n(\mathbf{G})$ parallelised over '23' slabs: $n(s_1^i : e_1^i,:,:)$ on node i.

Simplifies whole-cell FFTs.

Whole-Cell Arrays



Least well-behaved aspect when scaling to large P!
Forces synchronisation between nodes and emphasises load balance

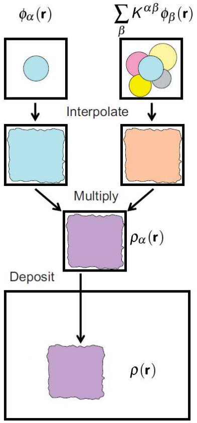
Therefore, minimise use of:

- Extraction of FFT boxes from whole-cell arrays (eg local potential)
- Deposition of FFT boxes to whole-cell arrays (eg density)
- Whole-Cell FFTs ($O(N_{\text{grid}} \ln N_{\text{grid}})$)

Example: Calculating the Charge Density

$$\rho(\mathbf{r}) = \sum_\alpha \rho_\alpha(\mathbf{r}) = \sum_\alpha \phi_\alpha(\mathbf{r}) \sum_\beta K^{\alpha\beta}\phi_\beta(\mathbf{r})$$

- Loop over batches of NGWFs ϕ_α on this node (batch size controlled by `density_batch_size`)
- Loop over all NGWFs ϕ_β for which $S_{\alpha\beta} \neq 0$ in this batch
  - Request ϕ_β ppds from other nodes if not local
  - Respond to incoming requests for ϕ_β ppds
  - Receive ϕ_β ppds from other nodes if not local
  - Accumulate $\sum_\beta K^{\alpha\beta}\phi_\beta(\mathbf{r})$ in *coarse* FFTbox for each ϕ_α
- Loop over ϕ_α in batch
  - Copy $\phi_\alpha(\mathbf{r})$ from PPDs to *coarse* FFTbox
  - Fourier interpolate row and column FFTboxes to *fine* grid
  - Take product $\rho_\alpha(\mathbf{r}) = (\phi_\alpha(\mathbf{r})).\left(\sum_\beta K^{\alpha\beta}\phi_\beta(\mathbf{r})\right)$
  - Deposit $\rho_\alpha(\mathbf{r})$ FFTbox to *fine* grid $\rho(\mathbf{r})$ whole-cell array


$\phi_\alpha(\mathbf{r})$
$\sum_\beta K^{\alpha\beta}\phi_\beta(\mathbf{r})$
Interpolate
Multiply
$\rho_\alpha(\mathbf{r})$
Deposit
$\rho(\mathbf{r})$

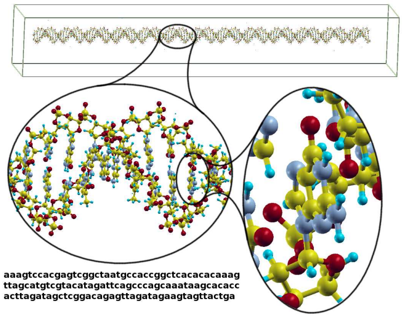
Outline

Scaling with N and P

Several ways of investigating scaling:

- Fixed computational resources... how big can I go?
  - increase N at fixed P
- Fixed problem size... how well does it scale?
  - increase P at fixed N
- Scalable problem, scalable resources... can I simulate an arbitrarily large system in feasible wall-clock time? ('time-to-science')
  - increase P and N at fixed ratio N/P

Random B-DNA: Scalable, non-periodic


aaagtccacgagtcggctaatgccaccggctcacacacaaag
ttagcatgtcgtacatagattcagcccagcaaataagcacacc
acttagatagctcggacagagttagatagaagtagttactga

Scaling with System Size

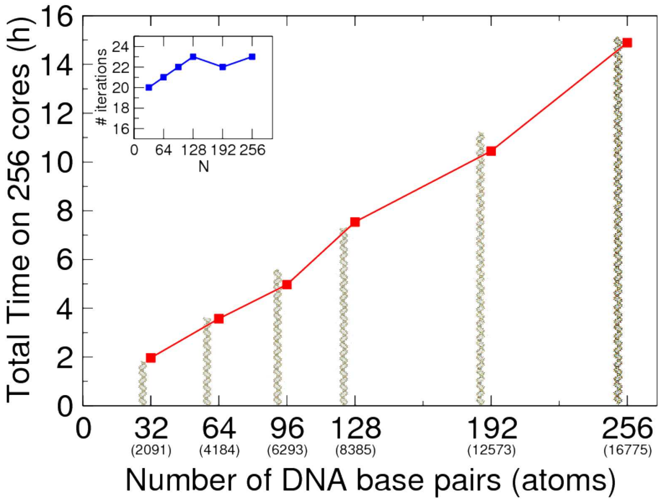


Figure: Wall clock time for total energy calculation on random DNA sequences. Inset: number of iterations for NGWF convergence.

Scaling with Number of Processors

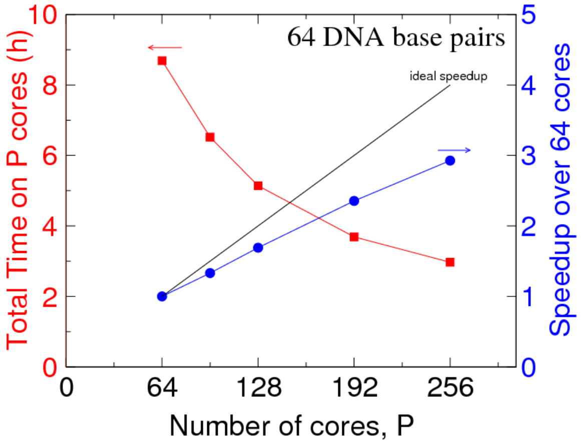


Figure: Wall clock time (red, left scale) and speedup over 64 cores (blue,right scale) for a total energy calculation of 64 base-pairs of DNA (4182 atoms) on varying number of cores.

Scaling at Constant Atoms per Processor

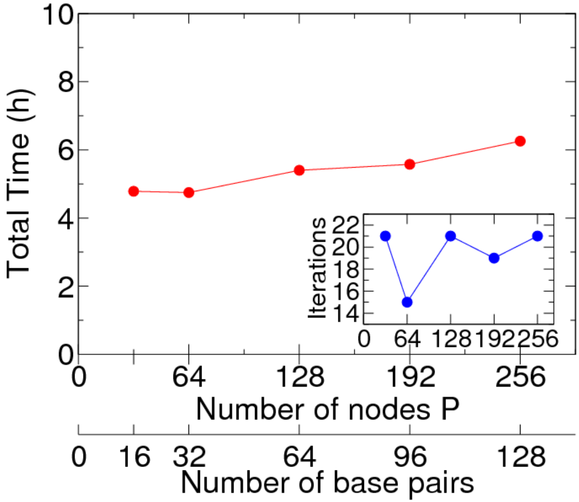


Figure: Wall clock time for 16-128bp DNA on 32-256 cores, keeping N/P constant. Inset: number of iterations for NGWF convergence.

Conclusion

- Highly Efficient Parallelisation - avoid synchronisation, hide communication behind calculation
- Re-usable, extensible algorithms for accumulation of function sums, integrals, etc in NGWF basis
- Scaling of calculations of tens of thousands of atoms on up to thousands of nodes