# Acceleration without breaking

## The search for sustainable portable performance in CASTEP

Phil Hasnip, Ed Higgins, Arjen Tamerus and Matt Smith

UNIVERSITY of York

UNIVERSITY OF CAMBRIDGE

OpenACC Summit
August 2020

# Introduction



- Aim: predict materials' behaviour from *first principles* i.e. no knowledge of what they'll do beforehand
- Properties governed by electrons
- Electrons behave according to quantum mechanics
- Solve the Schrödinger equation via density functional theory

The material's behaviour should emerge from the simulation

# CASTEP



- Created late 1980s, rewritten 1999-2001
  - Portable
  - Efficient
  - Parallel
  - User-friendly
- Language: ∼500kLOC of Fortran 2003
- Libraries: BLAS, LAPACK & FFT
- Parallelism: MPI + OpenMP
- Licence: dual academic (free) and commercial (from BIOVIA)

CASTEP's core development is done by researchers at the universities of York, Royal Holloway, Oxford, Durham and Cambridge.

# CASTEP workload



- CASTEP solves the DFT equations iteratively
- Solution written in a Fourier basis
- Needs repeated application of a Hamiltonian matrix

$$\mathrm{H} = \mathrm{T} + \mathrm{V_{loc}} + \mathrm{V_{nl}} + \mathrm{V_{nlxc}}$$

- $\mathrm{T}$ diagonal in Fourier space
- $\mathrm{V_{loc}}$ diagonal in direct space
- $\mathrm{V_{nl}}$ low-rank matrix update
- $\mathrm{V_{nlxc}}$ operations in direct and Fourier space (only present for certain classes of calculation)

# GPU porting project



- Focused on applying $\mathrm{H}$
- OpenACC
  - Generates kernels
  - Manages data transfers
- Libraries: cuBLAS, cuFFT (cuSOLVER, MAGMA)
- CPU handles communication

# GPU porting project



$$H = T + V_{loc} + V_{nl} + V_{nlxc}$$

- $V_{nl}$ low-rank matrix update
- CPU time spent entirely in BLAS
- Use OpenACC to transfer data to device
- cuBLAS for the operation

# Initial performance



Benchmark calculations on cluster with 12-core Ivy Bridge, 2-GPU K20c nodes.

Benchmark 1: solid benzene, 384 atoms, 1 node

| CPU Cores | CPU (s) | GPU (s) | Speedup |
|---|---|---|---|
| 4 | 1212.57 | 718.29 | 1.69 |
| 12 | 564.74 | 391.18 | 1.44 |

Benchmark 2: sapphire surface, 120 atoms, 2 nodes

| CPU Cores | CPU (s) | GPU (s) | Speedup |
|---|---|---|---|
| 8 | 2007.99 | 1165.99 | 1.72 |
| 24 | 954.68 | 591.37 | 1.61 |

# Fourier transforms

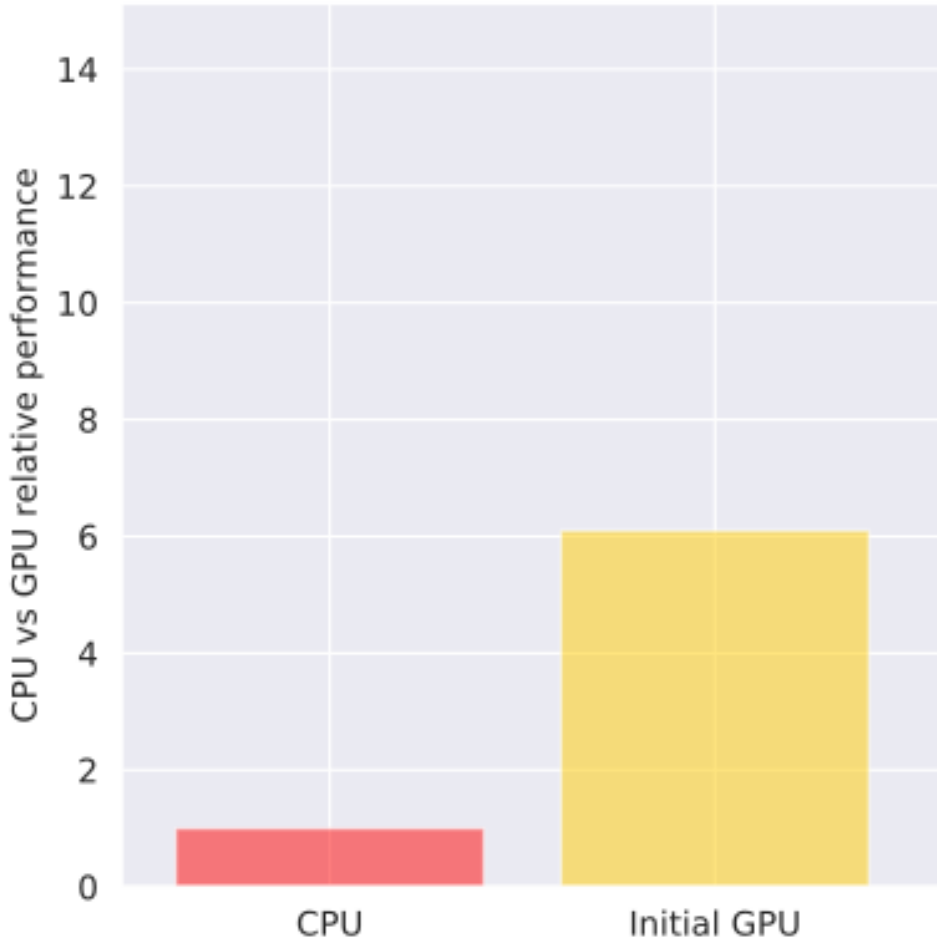


$$H = T + V_{loc} + V_{nl} + V_{nlxc}$$

- $V_{loc}$ and $V_{nlxc}$ both require Fourier transforms
- $V_{nlxc}$ usually FFT-bound
- CPU time spent either in:
  - 3D FFT (if Fourier components not distributed)
  - 1D FFTs and MPI comms (if Fourier components distributed)
- Use OpenACC to transfer data to device
- cuFFT for the operation

# Accelerating NLXC calculations





# FFT performance



- x6 speed
- However, less than half speed-up from memory bandwidth
- Why?

# FFT performance



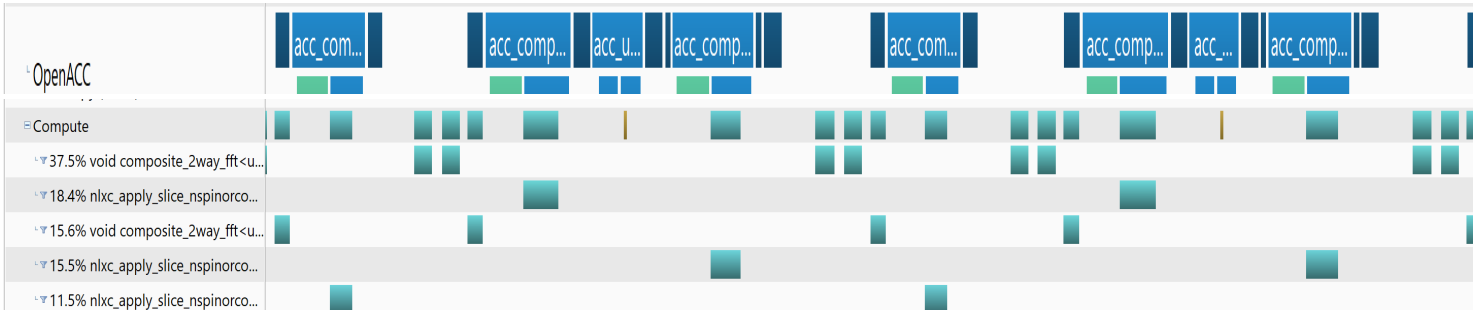
- x6 speed
- However, less than half speed-up from memory bandwidth
- Why?

```
API calls: 61.32% 50.7906s 4028791 ... cudaLaunchKernel
           31.40% 26.0067s 1792327 ... cuStreamSynchronize
```

- Lots of kernels launched
- Kernels short
- Lots of waiting

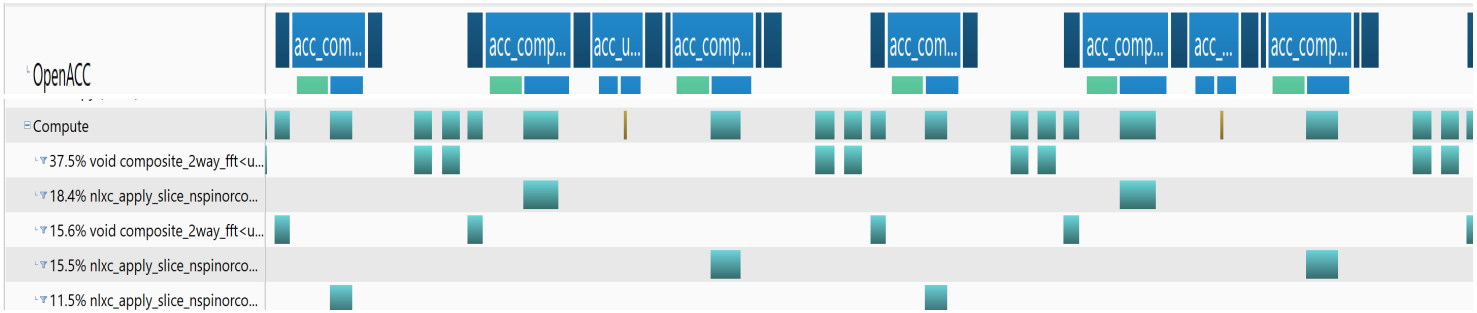
# Timeline analysis





# Timeline analysis

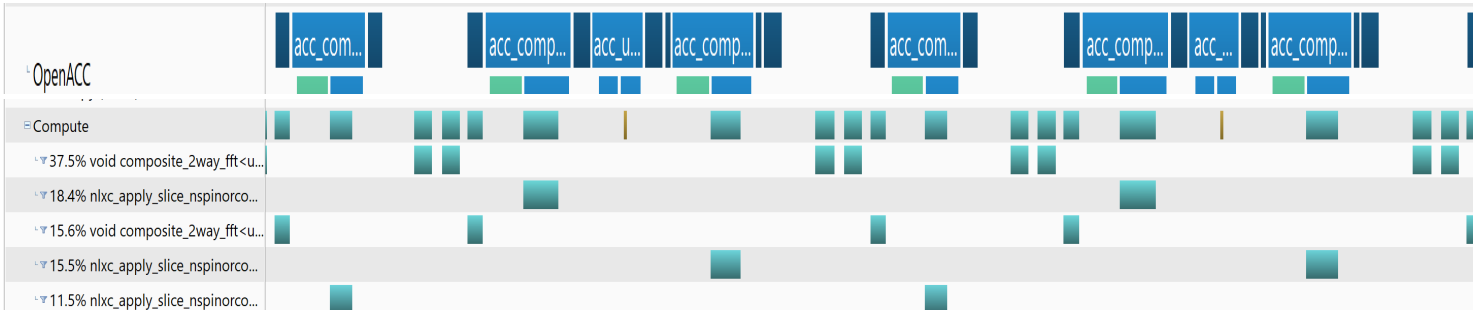




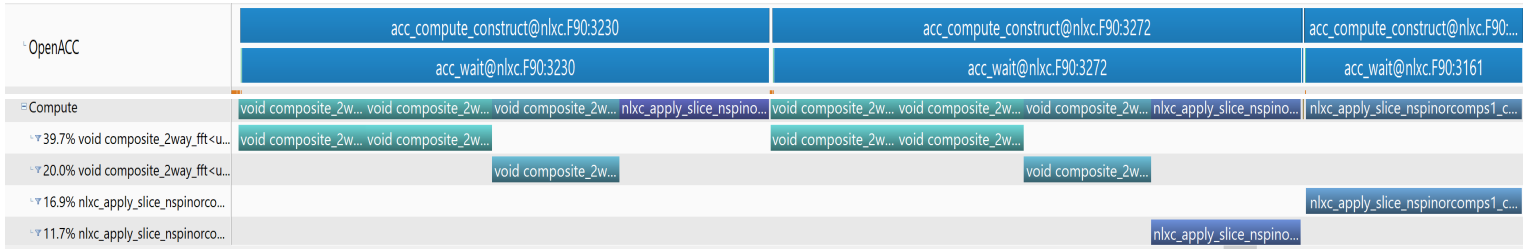

- Refactor to fuse OpenACC kernels
- Use batched FFTs

# Timeline analysis





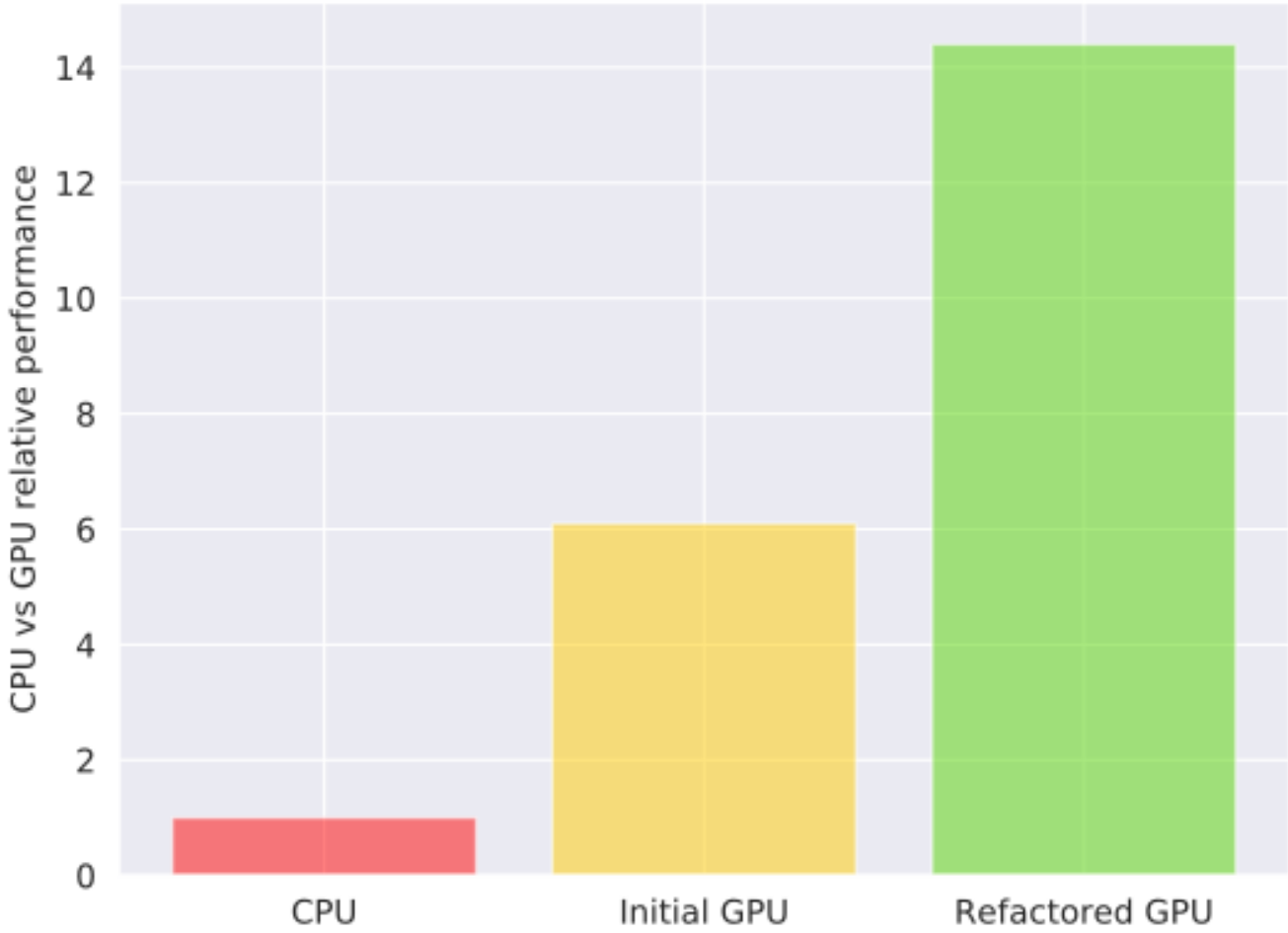

- Refactor to fuse OpenACC kernels
- Use batched FFTs



# Accelerating NLXC calculations



- Use NLXC calculation benchmark ($Fe_2VAl$)



# Thanks



- Matt Smith
- NVIDIA
- Sheffield and ORNL hackathon teams
- EPSRC

# Challenges



- OpenACC is a low-barrier route to GPU acceleration
- Optimising is more work
- Still many challenges!
- Data transfer is an issue
- Encapsulation of data and code makes it hard to optimise data movement
- In MPI-parallel, what is best mode of operation?
  - 1 MPI process per GPU, OpenMP threads for excess cores
    Need to understand interplay of OpenMP and OpenACC
  - Many MPI processes share GPUs
    Optimisations need access to GPU shared memory
- For distributed FFTs, CUDA-aware MPI for data transpositions