HP-SEE
SCL Quantum ESPRESSO extensions

www.hp-see.eu

Dusan Stankovic
Scientific Computing Laboratory
Institute of Physics Belgrade
dusan.stankovic@ipb.ac.rs

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

SCIENTIFIC COMPUTING LABORATORY

INSTITUTE OF PHYSICS BELGRADE

The HP-SEE initiative is co-funded by the European Commission under the FP7 Research Infrastructures contract no. 261499

About Quantum ESPRESSO

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Quantum ESPRESSO – open source code for electronic structure calculations

QUANTUMESPRESSO

- Based on density-functional theory, planewaves and pseudopotentials
- Core packages and modules to expand functionality
  - PW and CP used for testing
- Current version 5.0.1 – development done on 5.0

FFTE

- Fastest Fourier Transform in the East
  - Comes from Japan, Tsukuba University
- Written in Fortran
- Supports 1D, 2D and 3D transforms
  - With complex and real numbers
- Parallelism enabled with
  - MPI
  - OpenMP
- Limitations:
  - Only transforms of size $2^i * 3^j * 5^k$
  - OpenMP problems on tested system

FFTW3 library

- Fastest Fourier Transform in the West
  - Developed at MIT
- Written in C, with Fortran interfaces
- Supports Pthreads, OpenMP, MPI
- Complex and real transforms
  - Sizes can be arbitrary (even prime)
  - Though performance can suffer
- Most commonly used FFT package in the world
  - Performance comparable to vendor-specific libraries
  - Used in MATLAB for FFT calculations

FFTW3 advanced

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Supports transform planning
  - Create plan once, reuse many times
  - Measure performance
    - Run series of benchmarks
    - Finds best configuration for the system
  - Estimate parameters
    - Faster planning
    - Performance might not be optimal
- Serial routines are thread-safe
  - Though plan creation is not
- Can plan multiple (batched transforms)
  - Execute in a single routine call
  - Arbitrary strides in memory

QE code structure 1

- Written in Fortran 90
- Application structure
  - Common modules (e.g FFT or time-logging)
  - Calculation specific (like PW or CP)
- Using external numerical libraries
  - Vendor-specific (MKL, ESSL and so on)
  - Open source (FFTW3)
- Parallelized using MPI and OpenMP
  - Hybrid mode not supported for all libraries
- FFTW version 2 comes with QE
  - Hybrid mode enabled
  - Used for comparison in testing

QE code structure 2

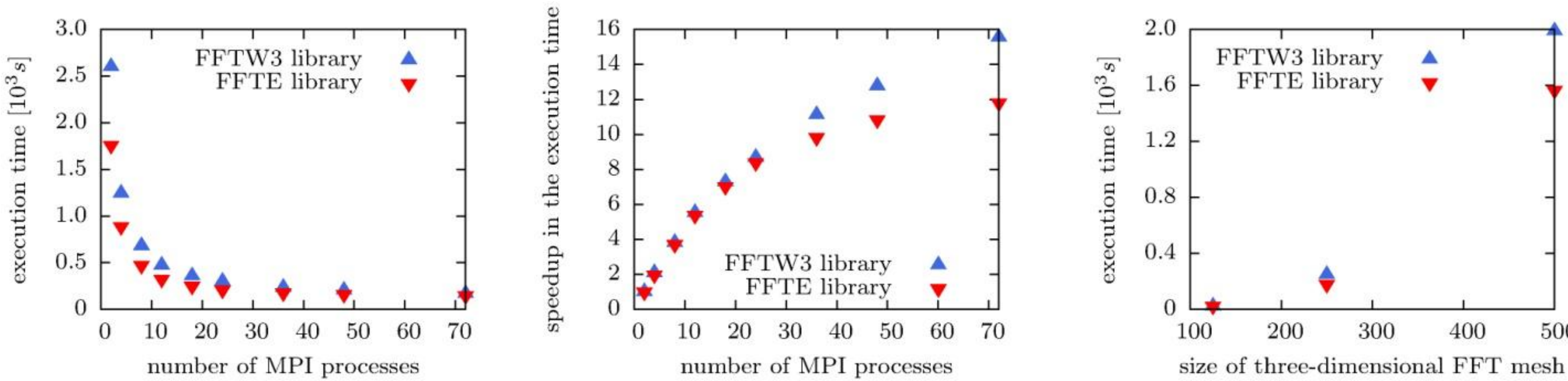
- Conditional compilation to select a library
  - *#ifdef*, *#else* and *#endif* directives
- FFT routine calls in *Modules/fft_scalar.f90* file
- QE uses 3D complex to complex transform
  - Zero-valued elements present in the FFT grid
  - Not using native 3D FFT calls
- Decomposition done manually
  - Instead 1D+2D transforms are used
  - Serial calls on each CPU
  - Data reordering in between using MPI

Enabling FFTE

- Distributed as source code files
  - First step to create a library
- FFTE selected with preprocessor directives
  - Compiles and executes if macro __*FFTE* is defined
  - Introduced variables prefixed with *ffte_*
- Overhead related to FFTE
  - Routine call needed for each transform
  - Initialization needed before each transform
    - Compare to FFTW3 plans (needed only once)
- Unable to support FFTE threading
  - Native OpenMP support not working in all cases
  - Serial routines are not thread-safe

FFTE performance 1

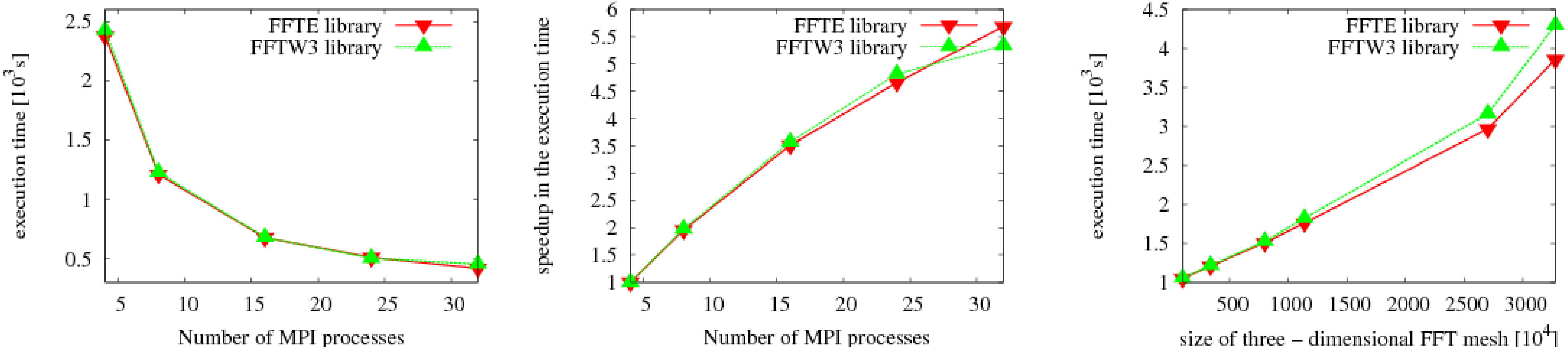
- PW module of QE tested
- Cluster with AMD Magny-Cours Opteron 6174 CPUs



- FFTE was slightly faster
  - More evident with bigger datasets

FFTE performance 2

- CP module of QE tested
- Cluster with Intel Xeon E5345 CPUs



- Similar results as with previous test
- Both tests – serial libraries were compared

Enabling FFTW3 threading

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- FFTW3 library is very widespread
  - Already pre-built at most sites
  - Just need to link to a library
- Fortran interface is present
  - Can be used, or
  - Call directly call C routines
    - Standardized in Fortran 2003
    - Need to reverse dimensions for multidimensional arrays
- Variants of FFTW3 threading
  - Threaded library (implicit)
  - Serial library with OpenMP region

FFTW3 threading variants

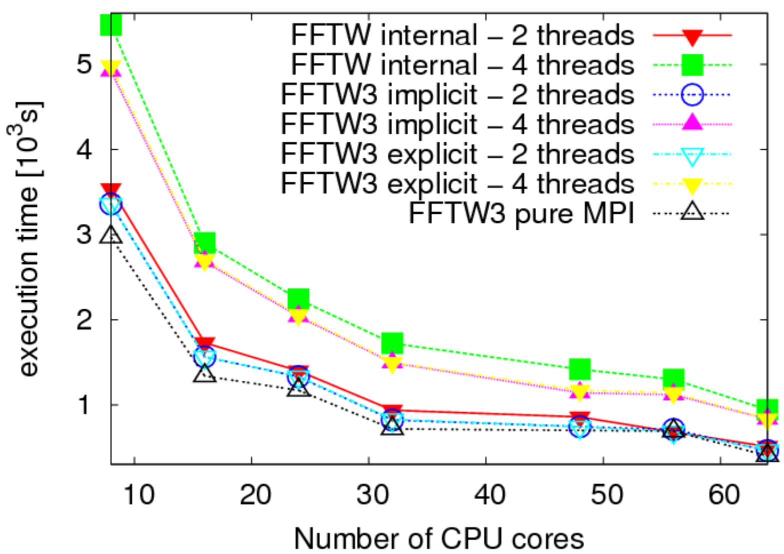
- Implicit
  - New OpenMP parallel region created for each call
  - Can use advanced planning interface
    - Multiple transforms in a single call
  - Code stays mostly the same
    - Added thread init routines
    - Linking with *libfftw3_omp* library
- Explicit
  - Single parallel region with multiple calls inside
  - Code needs to be modified
    - Splitting batched into multiple transforms
    - Linking with standard *libfftw3* library
- Performance should be very close

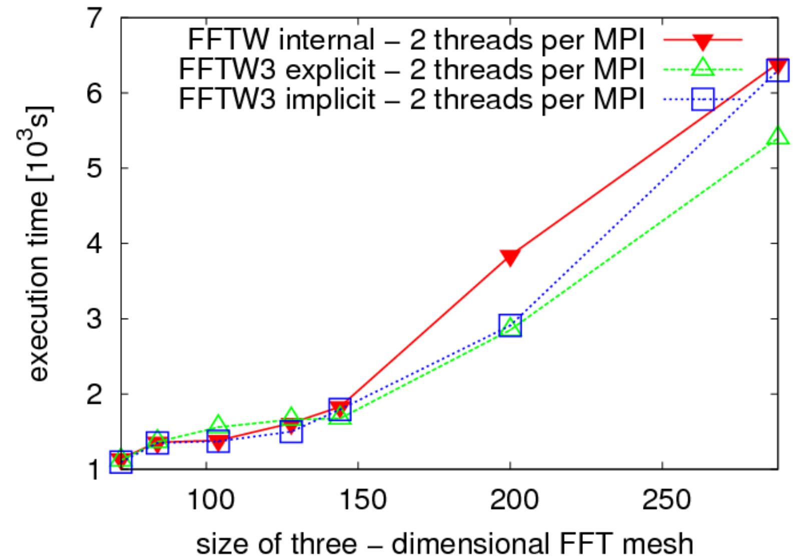
Running QE in hybrid mode

- QE has multiple levels of parallelism
  - MPI, but not just a simple division
  - Domain-specific grouping of processes
    - Different CPU scalability and RAM dependencies
  - OpenMP is the final layer, divide calculations
- A bit harder to tune for optimal performance
  - Depends on the input, cluster configuration etc.
  - Threading changes number of processes
- When testing hybrid QE
  - Find parameters for some test case
  - Modify ratio of OpenMP threads per MPI process
  - Leave other parameters the same

FFTW3 performance 1

- PW module of QE tested
- Cluster with Intel Xeon E5345 CPUs

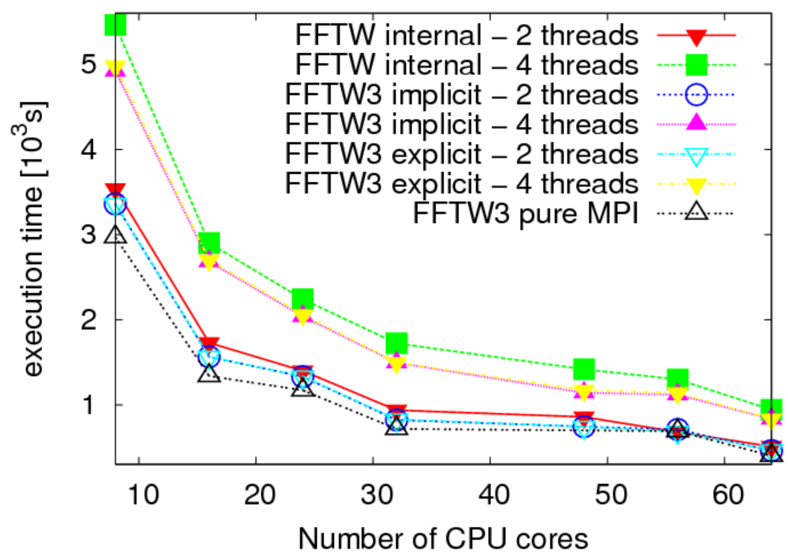


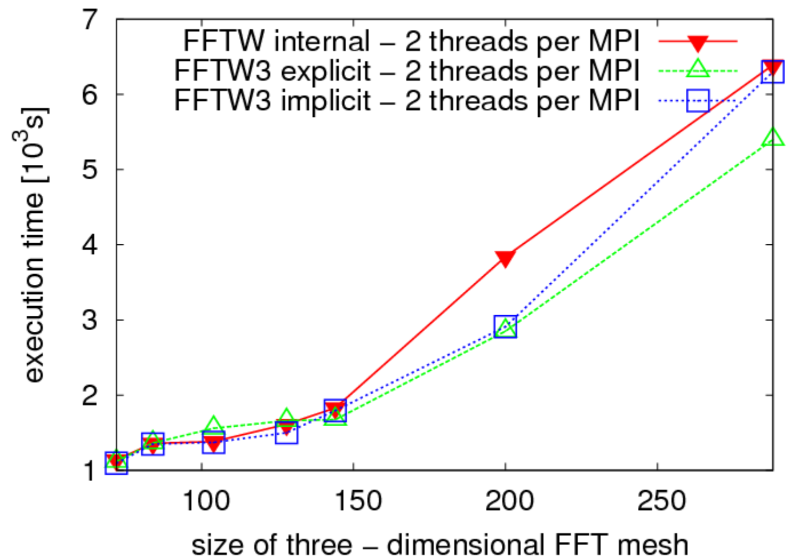


- MPI version fastest
  - Followed by 2-way threaded
- Both variants faster than internal FFTW (version 2)

FFTW3 performance 2

- CP module of QE tested
- Same cluster, Intel Xeon E5345 CPUs





- Similar results as the previous test
- No clear winner between threading variants
  - Different approaches, but similar implementation

Results explanation

- Version without threading was fastest
  - Threading introduces overhead
    - Thread creation
    - Synchronization at the end of parallel region
  - Threading reduces MPI communication
    - Lower number of MPI processes
  - Tradeoff – benefits not large enough to compensate
- Total CPU power the same, with or without threads
  - Not all problems benefit from threading
  - MPI implementations are memory-aware
    - Use shared memory for processes on the same node
    - Equivalent to shared variables in terms of performance

Conclusions

- FFTE extension
  - Comparable to or better results than FFTW3
  - Still has room for improvements
    - Implementation would have to be changed
- FFTW3 threading extension
  - Performance is not improved compared to MPI
  - Thread overhead too large on tested examples
  - Possibly needs larger datasets
    - Greater computation to overhead ratio
  - Possibly needs more processes
    - MPI communicaton reduction will be more significant
  - Faster than internal FFTW (version 2) library in hybrid mode