Quantum Natural Language Processing

All-hands meeting
04-Feb-2019

Intel Ireland, Leixlip

Agenda

1. Status of action points
2. Progress of active tasks (T1.1 and T1.2)
3. Next steps

Action points completed

- Steering committee meeting on April 5th 2019, 4 pm Irish time
  - Objective: outline of project objectives, deliverables and expected impacts; input from steering committee on desired outcomes, connections to related working groups; …?
  - TBD: ICHEC to internally circulate agenda and slides on April 1st 2019; venue (VC / ICHEC / Leixlip)
- Milestones (M4.3 and M4.4) added for EI reviews (mid-term and final)
  - https://git.ichec.ie/intel-qnlp/intel-qnlp/blob/master/README.md
- Presentation at All Ireland Conference on Quantum Technologies at Maynooth on Jan 21st 2019
- VK shared details about EQTC'19 with JK and FB
- Access to ICHEC resources
  - Restricted access to Git repository
  - Git repository access given to FB and JK; BQ to register
  - Kay access given to FB

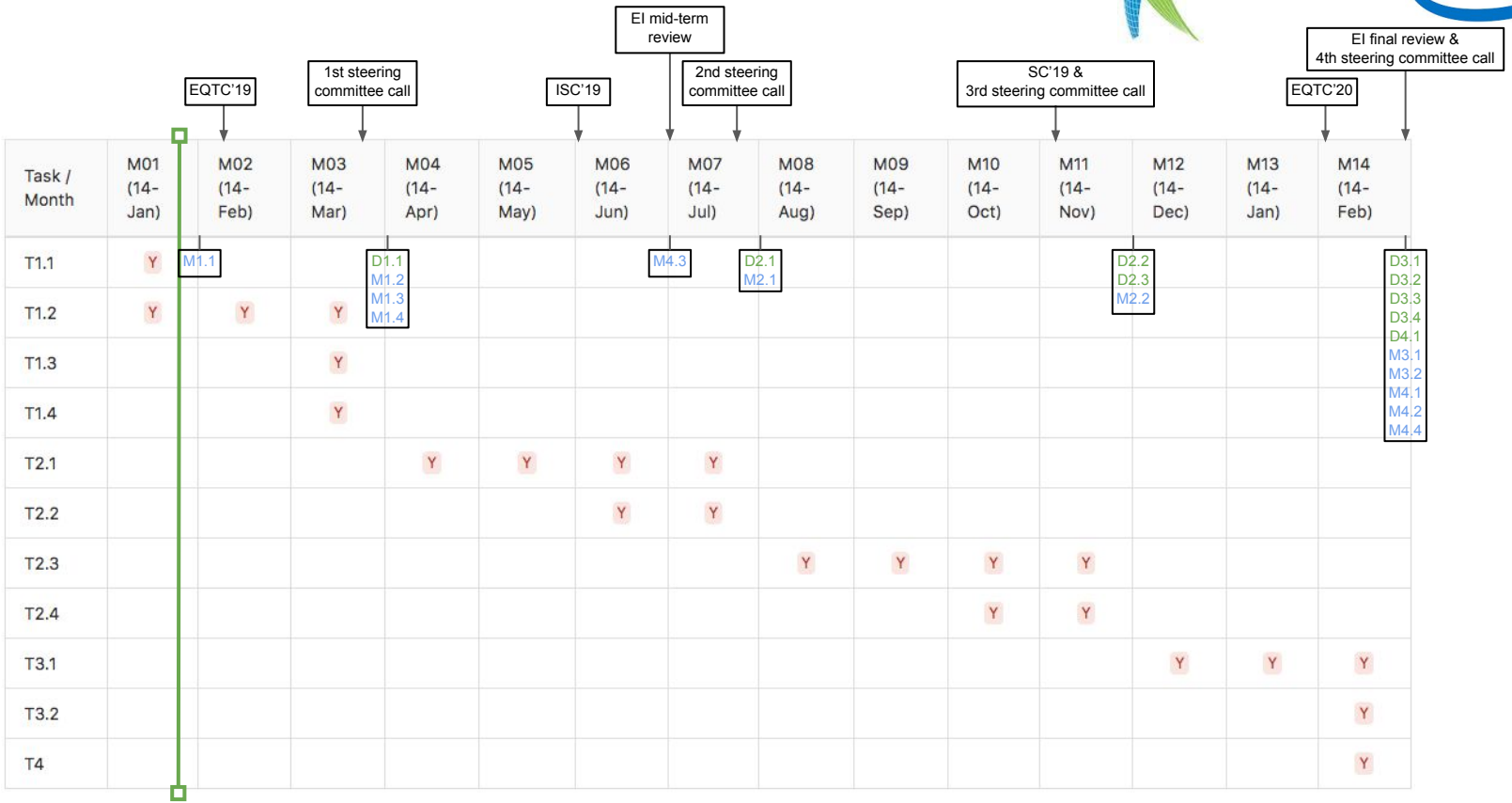
Action points ongoing

- ISC'19
  - FB included ICHEC in the collaboration with LRZ for tutorial session; details of presenters, authors and session agenda discussed; proposal to be submitted by LRZ on Feb 6th 2019
  - ICHEC preparing project poster; to be sent for approval to JK on Feb 4th 2019; to be submitted on Feb 6th 2019
- Technical work in T1.1 and T1.2

Action points pending

- Face-to-face meeting between ICHEC and FB
  - FB at ICHEC (Mar/Apr?); ICHEC at Intel Munich (ISC 2019?)
- Press release
  - ICHEC awaiting quote and approval from Intel
- ISC'19
  - Booth presentation to be scheduled; ICHEC and Intel(?) booths
- ICHEC to contact BTYSTE winner Adam Kelly (PW)

Project timeline



T1.1 - Install and test qHiPSTER on Kay

- qHiPSTER installation on Kay
  - Intel-QS ‘consistent-naming’ branch installed using Intel Parallel Studio XE 2018 update 4
  - Requires BigMPI to be used to allow for larger message sizes which occur when number of states becomes large (BigMPI must be built separately)
  - Build SDK-release of Intel-QS using BigMPI by following the updated installation steps on Git
  - Uses -O3 optimisation
  - Key issues:
    - Paths must be set manually by user to install BigMPI (make.inc)
    - SDK-release does not build due to several issues (LICENSE.txt, NoisyQureg.hpp dependency)
    - Ideally should not require user to adjust any files in order to install
    - Option to install BigMPI with qHiPSTER (and also the SDK-release) would be useful
    - Unit tests

T1.1 - Install and test qHiPSTER on Kay

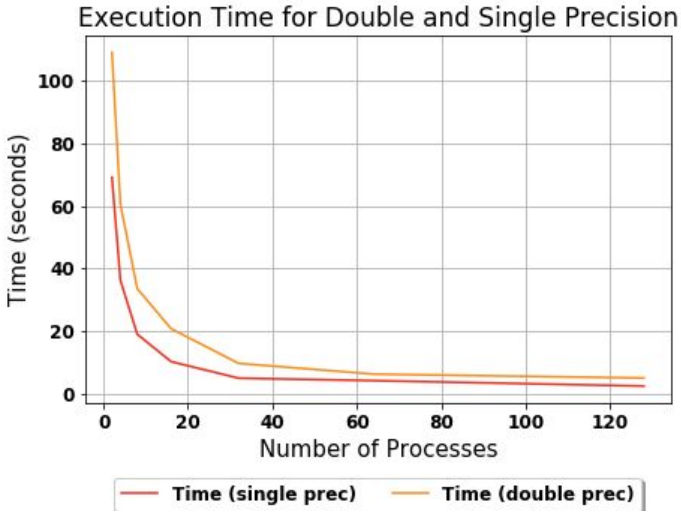
- Using qHiPSTER on Kay:
  - Load Intel 2018-u4 module, and custom modules for qHiPSTER and BigMPI
  - Run CMake using CMake version 3 with custom CMakeLists.txt file
    - Required precompiler flags:

```
-DMKL_ILP64 \
-DCOLLECT_TIME \
-DSTANDALONE \
-DOPENQU_HAVE_MPI \
-DBIGMPI \
-DINTELQS_HAS_MPI \
-DNOREPA_HAS_MPI"   # Not required for anything outside of Intel-QS/tests
```

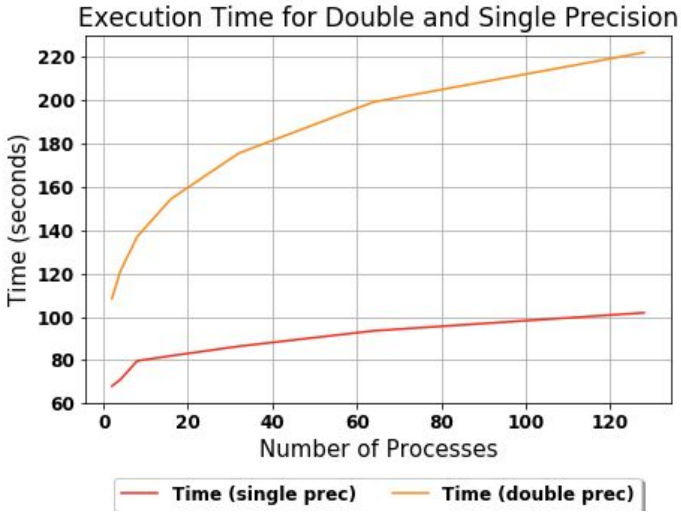
  - Build executable and run in standard way for a HPC environment

T1.1 - Install and test qHiPSTER on Kay

- Testing qHiPSTER on Kay
  - Tested qft_test.cpp from supplied tests for up to 35 qubits
  - Tests failed for problems with greater than 2^28 local states (16GB) per node
  - Strong scaling (34 qubits) and weak scaling (2^27 local states)



Strong Scaling



Weak Scaling

T1.1 - Install and test qHiPSTER on Kay

- Issues
  - Previously mentioned installation issues
  - Lack of documentation of qHiPSTER for use as SDK
  - Consider pushing ‘consistent-naming’ branch to ‘master’
- Questions
  - Is there a reason why MPI is disabled in test directory on ‘consistent-naming’ branch (-DNOREPA_HAS_MPI)?
  - Is MPIX_Sendrecv_x and MPIX_Allreduce_x the only BigMPI routines required by qHiPSTER (as they are the only BigMPI routines implemented by it)
  - It appears that the qft_test.cpp program did not use BigMPI routines due to it aborting because of communication errors for larger problem sizes
    - E.g. 36 qubits - single precision: 64 nodes each with 2^29 local states (32GB) out of a maximum of 192GB per node and 36 qubits)

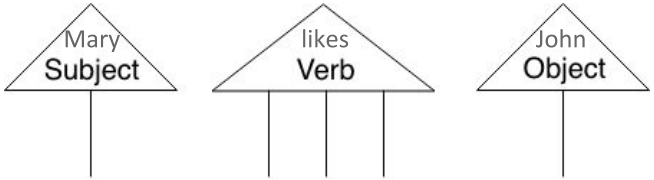
T1.2 - Mapping strategy for DisCo model

Observations:

1. DisCo model is formally specified using category theory
   a. Type specification and reduction, meaning computation operations
2. Quantum algorithms available from literature:
   a. Quantum nearest neighbour ⇒ from a set of vectors, identifies one that is “closest” to a given vector
   b. Sentence derivation ⇒ for a sentence, computes a “head” word which is representative of the sentence meaning
3. Requirements:
   a. Represent data structures and NLP operations using Dirac notation
   b. Define Q algorithms from literature using Dirac notation
   c. Map elements in Dirac notation from 3(a,b) to Q notation (gates/registers/circuits)
   d. Map elements in Q notation from 3(c) to qHiPSTER paradigm

T1.2 - Mapping strategy for DisCo model

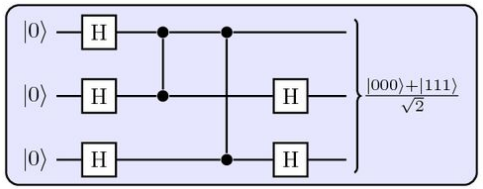
- Example of NLP ⟶ Dirac notation mapping strategy for statement: "***Mary likes John***"
  - Interactions between word types allow us to define the sentence meaning
  - Word interactions can be described using category theory, then mapped onto Dirac notation



Category theoretic graphical notation

$$|\text{Mary}\rangle \quad |\text{likes}\rangle := |\text{People}\rangle \otimes |\text{Sentences}\rangle \otimes |\text{People}\rangle \quad |\text{John}\rangle$$

Quantum Dirac notation



http://texample.net/media/tikz/examples/PDF/quantum-circuit.pdf

Quantum circuit notation (qHiPSTER)

- Challenge:
  - State creation and interactions: need understanding of required mappings to quantum gates/registers

T1.2 - Mapping strategy for DisCo model

Outline of algorithm:

1. Given a corpus C = {s_1, …, s_N} and target sentence t.
2. Pre-compute features of C and t.
   - Tag words with types, number of distinct types, number of unique words of each type, … ?
3. Define words in C and t as Q states.
   - Requires pre-computed features of C and t as input.
4. Store Q states of C and t in a Q memory s.t. object accesses are bounded.
   - * This is required to guarantee quadratic improvement in time.
5. Compute “head” for each sentence in C and t using the (deferred Q) sentence derivation algorithm.
   - Each head is a word whose output wire contains the sentence meaning.
   - Define Q operations for sentence derivation algorithm.
   - * How do we implement the bipartite graph and its partitioning?
   - * Define Q operations for type reduction of phrases/sentences (inherent in sentence derivation or implementation required?)
6. Apply the Q nearest neighbour algorithm on heads of t and each sentence in C.
   - Computes sentence(s) in C that are closest to t.
   - Hook the output wires of two heads ⇒ computes similarity of sentences using inner product.

Next steps

- Mapping strategies
  - Elements in algorithm
    1. Representation of data structures and NLP operations (word interactions, etc.) using Dirac notation
    2. Map elements in Dirac notation to Q notation (states)
    3. Map elements in Q notation to qHiPSTER paradigm
  - Algorithms
    1. Define Q nearest neighbour and sentence derivation algorithms using Dirac notation
    2. Map algorithms in Dirac notation to Q notation (gates/circuits)
    3. Map algorithms in Dirac notation to qHiPSTER paradigm
- Upcoming milestone M1.1 (M01) on 13-Feb-2019
- Action points pending
  - Press release (JK)
  - ISC'19 poster (VK)
  - FB at ICHEC
- Next meeting
  - Time: February 25th 2019; 10:00 (Irish time)
  - Venue: ICHEC, Dublin, https://meet.google.com/zvp-kmhp-pak
- Any other business?