Working Group 1: status and consultation outcomes

G. Koutsou

Computation-based Science and Technology Research Center (CaSToRC), The Cyprus Institute

CoE f2f, Sept. 23rd 2014, Ferrara

The Cyprus Institute

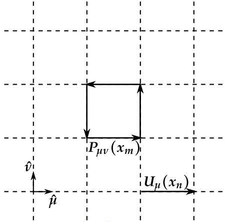
CaSToRC

Outline

- **Working Group (LFT)**
  - GK
  - Stefan Krieg
  - Nazario Tantalo
  - Carsten Urbach
- **Outline**
  - Short intro
  - Lattice 2014 meeting
  - Suggestions for description of work
  - Connections with other WPs
  - Discussion items

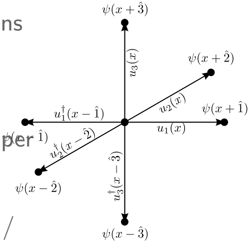
Discretization

4D space - time lattice



- 4-dimensional regular grid
- In parallel implementations, domain decomposed over MPI processes
- Need to exchange boundaries in every application

- **Typical data - type sizes**
  - Links U_μ: 3×3 complex matrix per four directions $\Rightarrow$ 36 complex per grid-point
  - Propagators $M^{-1}b$: 3×4 complex vector $\Rightarrow$ 12 complex per grid-point
  - 1 spinor and gauge links read and 1 spinor write per iteration: 960 bytes I/O per site (DP)
  - $\sim$ 1320 floating point operations per site
  - $\sim$ 1.4 flop/byte ratio (DP) / 2.7 flop/byte (SP) / 5.5 flop/byte (HP)



Characteristics

- **LQCD applications involve inversions of a large sparse matrix**
  - Bulk of calculation is in inverting a large sparse matrix implemented as a stencil operation on a vector
- **The communication paths are known and deterministic**
  - They are nearest neighbour and of constant packet size and determinsitic
  - Communication time is either susceptible to latency or bandwidth depending on surface to volume ratio
- **Simple counting of operations**
  - The application of the matrix on a vector requires additions and multiplications only
  - These are known and fixed on every application. Therefore the number of floating point operations is known and the efficiency can be immediately computed

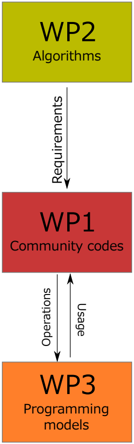
LQCD code development

- **Code development in LQCD community**
  - Relatively simple kernels have allowed for optimizing with moderate effort and predictable performance
  - Many groups implement and maintain their own packages
  - However, the emergence of more complex algorithms combined with diverse architectures create the need for dedicated coding efforts
  - Examples include: multiple grid sizes in the same run, different types of boundary conditions for sub-lattices, different layers of preconditioners, etc.
- **WP1 consultation session at Lattice 2014 conference**
  - Aim
    - to consult the lattice community on the code development services expected by the CoE
  - Attended
    - Most European sites were represented [Bielefeld, Bonn, Cambridge (DAMTP), Cyprus, DESY (ETMC, CLS), Edinburgh (UKQCD, QCDSF), Graz, INFN, Juelich, MAINZ (CLS), NVIDIA, Regensburg (RQCD, CLS), Southampton, Swansea, Wuppertal]

Lattice 2014 meeting

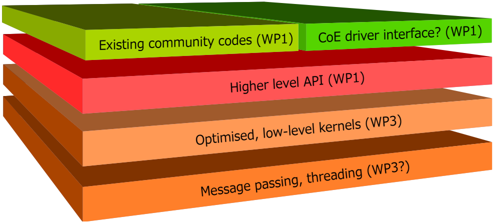
- **Some discussion outcomes**
  - Was acknowledged that there is a need for more unified, efficient European lattice code.
  - Was acknowledged that many efforts are duplicated (optimisations for different machines)
  - Was acknowledged that there is European competence in LQCD code development and that most of the community would adopt a CoE developed community code
- **General strategy**
  - WP1: implements library with common LQCD operations (determines and develops API,documentation, etc)
  - WP3: implements optimized, lower-level kernels (code generators, etc)
  - See document: `https://www.dropbox.com/s/nqh6vuq9gyw4bq3/CoE_lat2014_report.pdf`

WP interactions for code development



- **WP2**
  - Suggests requirements from algorithm development
  - E.g. multiple grids, arbitrary boundary conditions, dense matrix operations (`eigvals()`, etc.)
- **WP3**
  - Develops required low-level kernels for WP1
  - Consults with WP1 on interfaces, layouts, etc.
- **WP1**
  - Determines and implements API
  - Documentation
  - Proof of usability by interfacing in existing codes

Community code development



- **Proof of usability**
  - Use existing community codes as higher level interfaces (e.g. tmLQCD, OpenQCD, BQCD)
  - Build own CoE driver/interface to expose full functionality (interesting examples: Qlua, pyQCD)
- **Architecture-dependent development in lowest two levels**
  - Variable vector width x86
  - GPUs

Suggestions for tasks/deliverables

- **Tasks**
  - Collect requirements from WP2 and suggest kernels to be developed in WP3
  - Define and develop higher-level API
    - Naming conventions, operations, data layouts
  - Determine existing community codes to be interfaced with
  - Specify and develop CoE driver/interface
- **Deliverables**
  - Requirements from algorithms WP2
  - Definition of higher-level API (w/ WP3)
  - Library release
  - Interfaces with community codes
  - CoE driver/interface release

Feedback from SKA

- **WP1 related items by SKA**
  - Real-time analysis of data at the rate of 5-7.5 TBytes/sec
    - "data management" component of WP1, in combination with innovative hardware solutions
  - Pseudo-real time eigenosolve of a 1Mi$\times$1Mi dense matrix every $\sim$5 mins
    - Co-design opportunities as this problem can leverage innovations such as 3D RAM or Cache
    - Can benefit from algorithmic improvements (WP2)