INTRODUCTION AND MOTIVATION

Multi-disciplinary effort to develop Dataflow Programming Models for Computational Chemistry algorithms:
1. Break down the algorithms into fine-grained tasks with explicitly defined data dependencies to allow overall computation to scale.
2. Provide a robust and scalable DAG execution model for CC methods that are capable for utilizing different dataflow paradigms.
3. Compare “explicit dataflow” with “implicit dataflow” in terms of scalability, resource utilization, programmability.

DATAFLOW PROGRAMMING MODELS

- Focus on tasks and dataflow
  - Express kernels in form of Dataflow graphs
  - Map algorithms from DAG representation to HW platforms
- Develop for a portability layer, not an architecture
- Let the runtime deal with the hardware characteristics
- **Explicit** Dataflow: OpenMP
- **Implicit** Dataflow: StarPU, PaRSEC

GOALS

- Perform high level Coupled Cluster (CC) calculations for very large molecules
- NWChem is a suite of methodologies for Computational Chemistry
  - High Accuracy Methods to compute quantum-mechanical description of chemical systems
  - CC, Kohn–CC (OpenMP, OpenCL), CC-CC (Completely Renormalized CC)
  - Ground & Excited States
- Primary vehicle for this work: Coupled Cluster (CC) Methods
  - Gold standard for accurate quantum-mechanical description
  - CC’s accuracy comes at a significant computational cost
  - Scalability of CC is extremely important for real science

THE ALGORITHM, DATAFLOW, AND RUNTIMES

NWChem: CCSD

State-of-the-art “Coupled Cluster Single Double” (CCSD) Methods:
- CCSD code generated by TCE: multiple “T1|T2” subroutines for equations determining t1/t2 amplitudes matrices (t1 and t2 subroutines)
- Work in deep loop nests; with three types of code:
  -onetask
  - PARALLEL region
  -blocking communication

- Possible configurations
  - PARALLEL regions
  - no PARALLEL regions

- Insert_Task() / Insert_Task() calls

- For connecting data dependencies between subroutines, etc. more work is needed.

Main Goals of converting CC into a dataflow-based form:
1. Elimination of synchronization points by describing data dependencies between matrix blocks
2. Finer grained (pure) tasks to allow for exploitation of more parallelism

Extensions to “implicit” and “explicit” Dataflow:
- **Implicit** Dataflow: OpenMP
  - “Implicit” Dataflow in CC: Insert_Task()
  - For the same level of parallelism as in the OpenMP version, the Insert_task implementation is much simpler and less error-prone
  - No PARALLEL regions
  - No SERIAL constructs
  - No need to explicitly declare which variables are SHARED, PRIVATE, FIRSTPRIVATE, etc.
  - Original OpenMP call is replaced with INSERT_TASK_CORE() call

- **Explicit** Dataflow: PaRSEC
  - PAUSE, PACT, PEOS, PLOC, PPAR

IMPLICIT DATAFLOW

OpenMP

- “Implicit” Dataflow in CC: OpenMP
  - OpenMP feature “tasking” supports task level parallelism
  - ”Implicit” Performance: 32, 64, 128 nodes

EXPLICIT DATAFLOW

PTG

- “Explicit” Dataflow in CC
  - PTV for DGEMM tasks organized in chain:
  - Recognizing the DGEMMs from a serial chain to parallel execution followed by a reduction requires reordering the 4 lines that define the dataflow of matrix C (lines 1-4) by the following single line:
  - PTV snippet for parallel DGEMM tasks:

IMPLICIT DATAFLOW:

- Easy to use: based on the serial task-insertion API
- Performance boost: 1.5x

EXPLICIT DATAFLOW:

- Complete detachment from original control flow (and its limitations)
- Performance boost: 2.6x
- And better scaling
- Bigger engineering job

CONCLUSION

Dataflow models are an alternative approach with significant potential to the CGP model for CC

Integration of “implicit” Dataflow in CC

- **Implicit Task** (enabled) CC implementations for 3 runtimes:
  - OpenMP (standard for parallel shared-memory applications)
  - PaRSEC
  - StarPU

Programmability Advantage:
- Implicit Dataflow Model expresses parallelism by submitting tasks to the sequential flow of the original program.
- Domain scientists can take an extra, multi-million line application, like NWChem, and express additional parallelism by simply submitting tasks to the natural control flow of CC.
- Can re-use long established features from original code, without diverging too much into different representation.

PERFORMANCE SETUP

- **OpenMP**
  - Cascade I7 950:16GB RAM, 2666 MHz system
  - with a total of 16 cores (2.66 GHz)
  - Intel 15.0.0 compiler
  - Scheduling and task level parallelism on distributed architectures, use other runtimes, e.g. PaRSEC, StarPU

- **StarPU**
  - OpenMP: a well known standard for parallel shared-memory applications (only).
  - The main difficulty is to ensure that there are no data dependencies between PARALLEL regions.
  - That makes it difficult to express a complex DAG of tasks (and their data dependencies to one another) using OpenMP.
  - For scheduling and task level parallelism on distributed architectures, use other runtimes, e.g. PaRSEC, StarPU

SUMMARY: OpenMP

- Insert_task() is not limited to shared memory task-level parallelism on distributed architectures
- Insert_task() simpler than OSF_TASK and less error prone
- No ATTRIBUTE declaration needed for variables, no PARALLEL regions, etc.
- For connecting data dependencies between subroutines, etc. more work is required for GA data discovery
- “Simplicity” argument of Insert_task() gets weaker!

SUMMARY: PaRSEC

- Original code performs best for 4 cores/node
- While explicit dataflow manages to improve performance all the way up to 144 cores
- Reasons:
  1. Unit of parallelism is much coarser in original code
  2. Original code uses global atomic variables for load balancing

SUMMARY: PTG