1. Challenges of Parallel and Distributed Programming

- A notoriously laborious and difficult endeavor
  - Wide range of technical difficulties (e.g. deadlock, atomicity, fault-tolerance).
  - Traditional computational problems (e.g. correctness, completeness, termination).
  - While ensuring scalability and performance effectiveness.

- Open research problem:
  - Distributed programming frameworks (e.g. Map reduce [DG08], Graph Lab [LK*10], Pregel [MAB*10], Mizan [KKJA10]).
  - Distributed programming languages (e.g. Erlang [AV90], X10 [SSvP07], NetLog [GW10], Meld [CARG*12]).

- We seek an approach that is declarative, based on logical foundations, expressive and concise.

- Motivated by chemical reaction equations:
  - $6CO_2 + 6H_2O \rightarrow 2H_2CO_3 + 6O_2$

2. Introducing Rule-Based Multiset Rewriting

- Constraint Handling Rules (CHR) [Frô98]
  - Rule-based constraint logic programming language.
  - Based on multiset rewriting over first order predicate terms, called CHR constraints.
  - Concurrent, committed choose and declarative.
  - CHR programs consist of a set of CHR rules of the following form:
    $$r : P \iff G \iff B$$

- Informally means: If we have $P$ and $S$ such that $G$ is satisfiable, replace $S$ with $B$.

- Example: Greatest common divisor (GCD)
  - $\text{base} : \text{gcd}(0,0) \iff \text{true}$
  - $\text{reduce} : \text{gcd}(k,0) \iff 0 < N \land N < M \iff \text{gcd}(M, M)$
  - $\text{gcd}(0, 0) \iff \text{true}$

3. CHR*, Distributed Multiset Rewriting for Ensembles

- Elements are distributed across distinct locations $(k_1, k_2, \ldots)$, each possessing its own multiset of elements.
  - $\text{edge}(k_1, \ldots, \text{base}) \iff \text{edge}(k_2, \ldots, \text{base})$

- Rewrite rules explicitly reference the relative location of constraints:
  - base_rule : $X$ \text{edge}(Y, D) \iff $X$ \text{path}(Y, D).$
  - elim_rule : $X$ \text{path}(Y, D) \iff $X$ \text{path}(Y, D_2) \iff D_2 < D \iff \text{true}.$
  - trans_rule : $X$ \text{edge}(Y, D) \iff $X$ \text{path}(Z, D) \iff $X$ \text{path}(Z, D + D) \iff \text{true}.$

- $I | C$ specifies that matching $C$ is located at $I$.

- Rewrite rules can specify “local rewriting”:
  - $\text{edge}(k_1, x_1, \ldots, x_i) \iff \text{edge}(k_2, x_1, \ldots, x_i)$

- Rewrite rules can specify link-restricted rewriting:
  - $\text{edge}(k_1, x_1, \ldots, x_i) \iff \text{path}(x_1, x_2, \ldots, x_i) \iff \text{edge}(k_2, x_1, \ldots, x_i)$

4. Example: Parallel Mergesort

Parallel mergesort: Assumes tightly coupled ensembles (multicore, shared memory, etc.)

- $\text{X unsorted}(l) \iff \text{X sorted}(l).$
- $\text{X unsorted}(l) \iff \text{len}(X) < 2 \iff \text{exists}(Y, Z) \iff \text{split}(X, Y, Z).$
- $\text{X sorted}(l) \iff \text{exists}(Y, Z) \iff \text{merge}(X, Y, Z).$
- $\text{X unsorted}(X) \iff \text{len}(X) = 2 \iff \text{exists}(Y, Z) \iff \text{split}(X, Y, Z).$
- $\text{X sorted}(X) \iff \text{exists}(Y, Z) \iff \text{merge}(X, Y, Z).$
- $\text{X unsorted}(X) \iff \text{len}(X) = 2 \iff \text{exists}(Y, Z) \iff \text{split}(X, Y, Z).$
- $\text{X sorted}(X) \iff \text{exists}(Y, Z) \iff \text{merge}(X, Y, Z).$

- New locations “dynamically” created to solve sub-problems.

5. Example: Distributed Hyper-Quicksort

Distributed Hyper-Quicksort: Assumes loosely coupled ensembles (network, message passing interface, etc.)

- “Local” sorting algorithm Parallel merge sort rules

- Distributed Hyper quicksort rules
  - $\text{X sorted}(X) \iff \text{len}(X) > 1 \iff \text{exists}(Y, Z) \iff \text{split}(X, Y, Z).$
  - $\text{X sorted}(X) \iff \text{len}(X) = 1 \iff \text{exists}(Y, Z) \iff \text{split}(X, Y, Z).$
  - $\text{X sorted}(X) \iff \text{len}(X) = 0 \iff \text{exists}(Y, Z) \iff \text{split}(X, Y, Z).$

- Data (unsorted numbers) initially distributed across 2^n locations.

- In termination (quiescence), 2^n locations are in total order.

6. Main Challenges

- Effective execution of multiset rewriting in decentralized context:
  - Incremental matching
  - Termination on quiescence
  - Internet (event) driven matching

- Execution of link-restricted rewrite rules is non-trivial:
  - $\text{X sorted}(X) \iff \text{len}(X) > 1 \iff \text{exists}(Y, Z) \iff \text{split}(X, Y, Z).$

- Requires that locations $X$ and $Y$ rewrites respective multisets atomically.

- In general (n locations involved), its essentially n-conscious problem.

- Designing effective mappings from locations to computation resources
  - Initialization: How are locations distributed across actual distributed system?
  - Load-balancing: How are dynamically created “locations” distributed?

- Designing the Language:
  - What are the minimal core language features?
  - What extended language features do we need?
  - What kind of type safety guarantees can we provide?

- Existing woes and challenges of distributed programming:
  - Fault tolerance and recovery.
  - Serializability of distributed computation.

7. Current Contributions and Results

- Developed an operational semantics for 0-link restricted rewriting
  - Based on CHR refined operational semantics [DSdBH04].
  - Decentralized, Incremental, interrupt driven execution.
  - Proven soundness and completeness (exhaustiveness) of rewriting

- Formalized encoding of n-link restricted rewriting into 0-link restricted rewriting
  - Based on 2 Phase commit n-consensus protocol [MLB85].
  - Optimized encoding for 1-link restricted rewriting
  - General encoding for n-link restricted rewriting

- Prototype implementation
  - Implemented in Python, decentralized execution via OpenMPI bindings and thread scheduling via multi-threading libraries.

- CHR based optimization of multiset matching (e.g. optimal join ordering, indexing for non-linear patterns, early guard scheduling)

- Basic resource mapping: Initial locations mapped to OpenMPI nodes, dynamically created locations mapped to threaded computation at source of creation.

8. Future Works

- Finalizing language design and high-performance implementation
  - C, C++ or Haskell/GHC as source language
  - Improved high-level feature encodings
  - Explore implementation via Pregel [MAB*10] or Mizan [KKJA10].

- Improve language design
  - Aggregates, linear comprehensions, Dialogol style retraction
  - Extending core language
  - New features via encoding in core language

- Dealing with unreliable communications and faulty computation resources
  - Fault tolerance backends and fault recovery interfaces
  - Improved n-link restriction encodings (via 3 Phase commit [KD95] or Paxos Algorithm [Lam98])

- Funded by the Qatar National Research Fund as project NPRF 09-667-1-100 (Effective Programming for Large Distributed Ensembles)