Data-Parallel Computing Meets STRIPS

Erez Karpas Tomer Sagi Carmel Domshlak Avigdor Gal
Avi Mendelson Moshe Tennenholtz

Technion-Microsoft Electronic-Commerce Research Center

Outline

1. Motivation
2. DPPS
3. DPPS as Planning

Data Processing — Before “Big Data”

- Database Management Systems (DBMS)
- Declarative query — expressed in SQL
- Query execution plan
  - Easy to generate from declarative query
  - Hard to optimize
- Very limited support for user-defined functions

Data Processing — After "Big Data"

- MapReduce / Hadoop / Dryad
  - Low-level programming
  - Only user-defined functions
  - No declarative queries
- SCOPE / DryadLINQ / Pig / Hive
  - High-level programming
  - Support user-defined functions
  - Limited declarative queries

Data Processing — After “Big Data”

- MapReduce / Hadoop / Dryad
  - Low-level programming
  - Only user-defined functions
  - No declarative queries
- SCOPE / DryadLINQ / Pig / Hive
  - High-level programming
  - Support user-defined functions
  - Limited declarative queries

User Defined Functions in Declarative Queries

- Including user-defined functions hinders query optimization
  - User must specify some base plan
  - Query plan optimizer does not “understand” user-defined functions, and does not know which optimizations are safe
- Existing approaches:
  - No optimization when user-defined function in query
  - User-defined functions must have some pre-specfied signature
  - Static code analysis to “understand” user-defined functions

User Defined Functions in Declarative Queries

- Including user-defined functions hinders query optimization
  - User must specify some base plan
  - Query plan optimizer does not “understand” user-defined functions, and does not know which optimizations are safe
- Existing approaches:
  - No optimization when user-defined function in query
  - User-defined functions must have some pre-specfied signature
  - Static code analysis to “understand” user-defined functions

Running Example: Histogram Computation

- Suppose we have a users table `T` with 10^9 users
- We want two histograms of `T`: by age and by relationship status

Running Example: Histogram Computation

- Suppose we have a users table `T` with 10^9 users
- We want two histograms of `T`: by age and by relationship status

In SQL or similar

```
SELECT COUNT(T.age) FROM T;
SELECT COUNT(T.rls) FROM T;
```

Running Example: Histogram Computation

- Suppose we have a users table `T` with 10^9 users
- We want two histograms of `T`: by age and by relationship status

In SQL or similar

```
SELECT COUNT(T.age) FROM T;
SELECT COUNT(T.rls) FROM T;
```

Query Execution Plan

```
Agg(age, scan(T))
Agg(rls, scan(T))
```

Running Example: Histogram Computation (2)

- Suppose we have a user-defined function, `DAgg`, which aggregates by two fields simultaneously

- The question is how to come up with this execution plan automatically

Running Example: Histogram Computation (2)

- Suppose we have a user-defined function, `DAgg`, which aggregates by two fields simultaneously

Query Execution Plan using DAgg

```
DAgg(age, rls, scan(T))
```

- The question is how to come up with this execution plan automatically

Running Example: Histogram Computation (2)

- Suppose we have a user-defined function, `DAgg`, which aggregates by two fields simultaneously

Query Execution Plan using DAgg

```
DAgg(age, rls, scan(T))
```

- The question is how to come up with this execution plan automatically

Our Contribution

- Introduce Data-Parallel Program Synthesis (DPPS), a formal framework for studying these problems
- Study expressivity and complexity of DPPS
- Show compilation to AI planning

Outline

Data-Parallel Program Synthesis Framework

- Framework is based on tracking *data chunks*
- A data chunk represents some piece of data, e.g.:
  - all records of males between the ages of 18–49
  - the average salary of all males between the ages of 18–49
- We do not need to know the *value* of the data, only its description
- Each data chunk d is associated with the amount σ_d of memory it requires

DPPS Task

- D — a set of possible data chunks, with sizes σ_d
- N — a finite set of computing units, with memory capacities κ_n
- A — a set of possible computation primitives, $a \in A$ described by:
    - $\bar{I} \subseteq D$ is the required input
    - $\bar{O} \subseteq D$ is the produced output
    - $C : N \longrightarrow \mathbb{R}^{0+}$ computation cost on each processor
- $T : N \times D \times N \longrightarrow \mathbb{R}^{0+}$ — the data transmission cost function
- s_0 — the initial state of the computation
- G — the goal of the computation

DPPS Task (2)

- A DPPS state specifies which processor holds which data chunks
- A solution is a sequence of actions (compute / transmit / delete data) which achieves the goal from the initial state
- The possible data chunks D and computations A may be given explicitly or described implicitly
  - If they are described implicitly the sets could be infinite

DPPS Expressivity

Theorem

DPPS is at least as expressive as relational algebra with aggregation

Proof sketch.

Given a relational algebra expression, we can construct a DPPS task whose operators are the RA operators, and data chunks are possible RA expressions. $\square$

DPPS Complexity ☹

Theorem

Satisficing data-parallel program synthesis is NP-hard, even when the possible data chunks are given explicitly.

Proof sketch.

By reduction from SAT, exploiting memory capacity constraints □

DPPS Complexity ☹☹

Theorem

Optimal data-parallel program synthesis with a single processor is NP-hard, even if the possible data chunks are given explicitly, and there are no memory constraints.

Proof sketch.

By reduction from delete-free planning □

DPPS Complexity ☹☹☹

Theorem

Optimal data-parallel program synthesis with a single data chunk is NP-hard.

Proof sketch.

By reduction from the Steiner tree problem □

DPPS Complexity ☺

Theorem

Satisficing data-parallel program synthesis with no memory constraints can be solved in polynomial time, when the possible data chunks are given explicitly.

Proof sketch.

By reduction from delete-free planning □

Outline

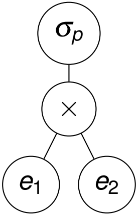
DPPS Compilation

- When the computations and data chunks are given explicitly, compilation to planning is straightforward
  - Predicate `holds(?node, ?data)`
  - Actions
    - For each computation `compute(?node, ?computation)`
    - Transmission `transmit(?node, ?data, ?node2)`
    - Data deletion `del(?node, ?data)`
  - Capacity constraints can be enforced with numerical fluents

DPPS Compilation without Explicit Data

- When the computations and data chunks are given implicitly, compilation is still possible sometimes
- When data chunks have a structure (e.g., expression trees), it is possible to represent such trees using predicates

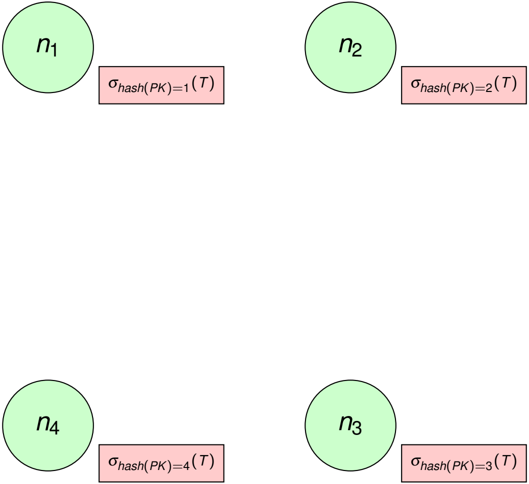
Expression Tree



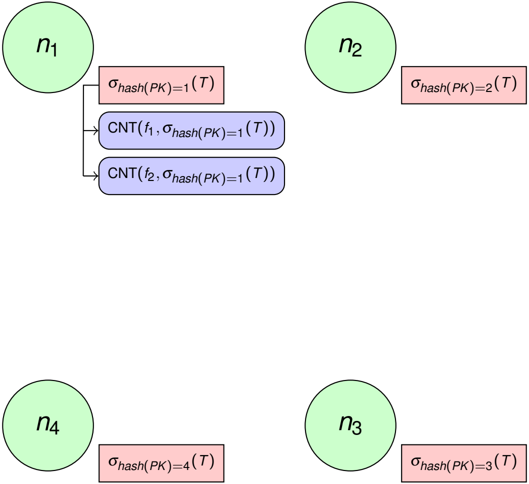
Encoding

`select(`n_1`,` p`,` n_2`)`
`join(`n_2`,` e_1`,` e_2`)`

DPPS Compilation: Proof of Concept

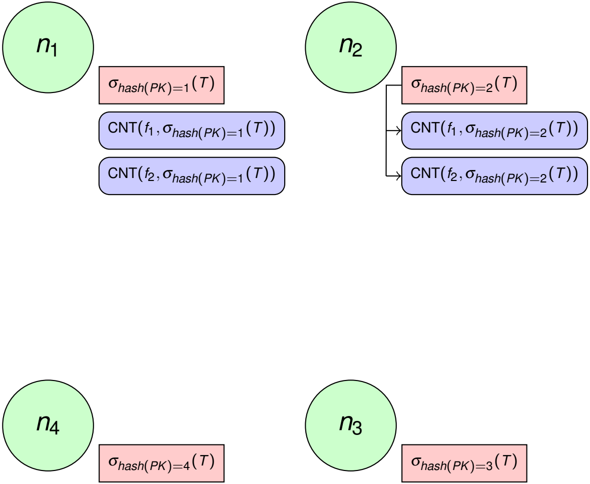


DPPS Compilation: Proof of Concept



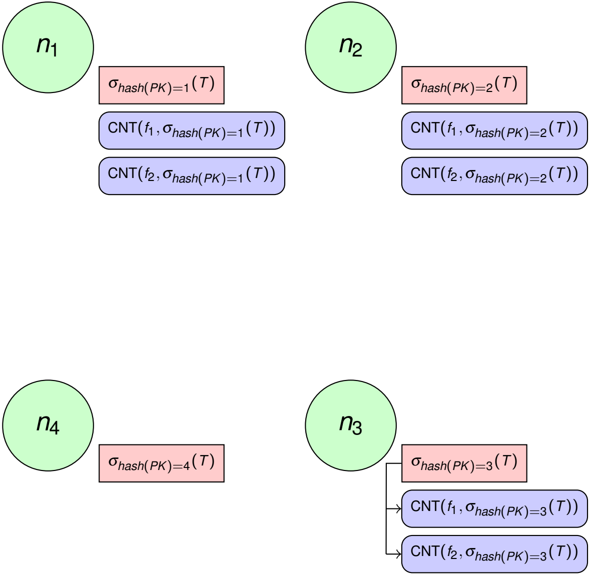
`DAgg(`n_1`,` f_1`,` f_2`,` $\sigma_{hash(PK)=1}(T)$`)`

DPPS Compilation: Proof of Concept



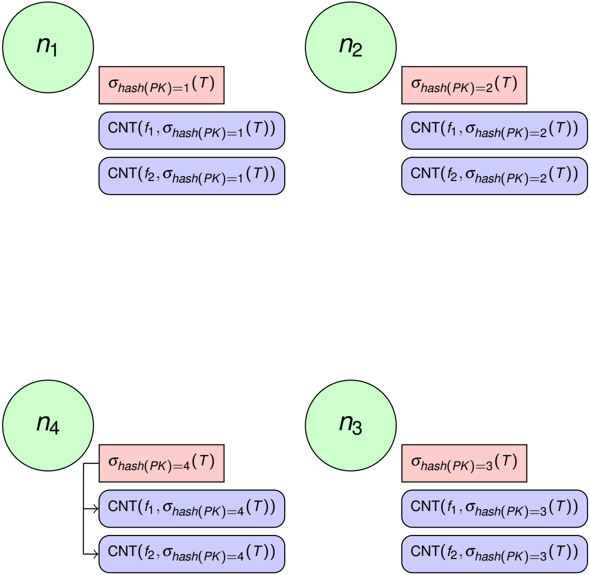
DAgg(n_1, f_1, f_2, $\sigma_{hash(PK)=1}(T)$)
DAgg(n_2, f_1, f_2, $\sigma_{hash(PK)=2}(T)$)

DPPS Compilation: Proof of Concept



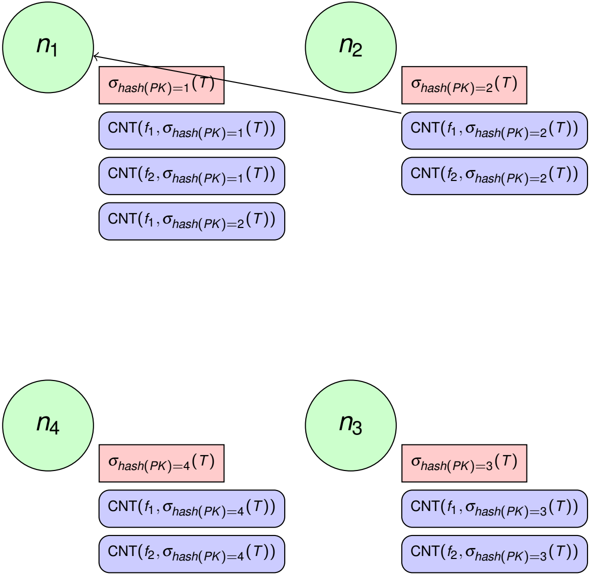
`DAgg`(n_1, f_1, f_2, $\sigma_{hash(PK)=1}(T)$)
`DAgg`(n_2, f_1, f_2, $\sigma_{hash(PK)=2}(T)$)
`DAgg`(n_3, f_1, f_2, $\sigma_{hash(PK)=3}(T)$)

DPPS Compilation: Proof of Concept



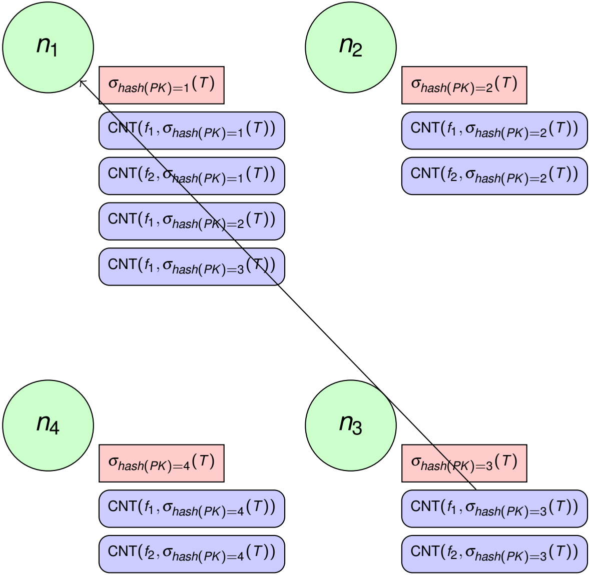
`DAgg(`n_1, f_1, f_2, $\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2, f_1, f_2, $\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3, f_1, f_2, $\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4, f_1, f_2, $\sigma_{hash(PK)=4}(T)$`)`

DPPS Compilation: Proof of Concept



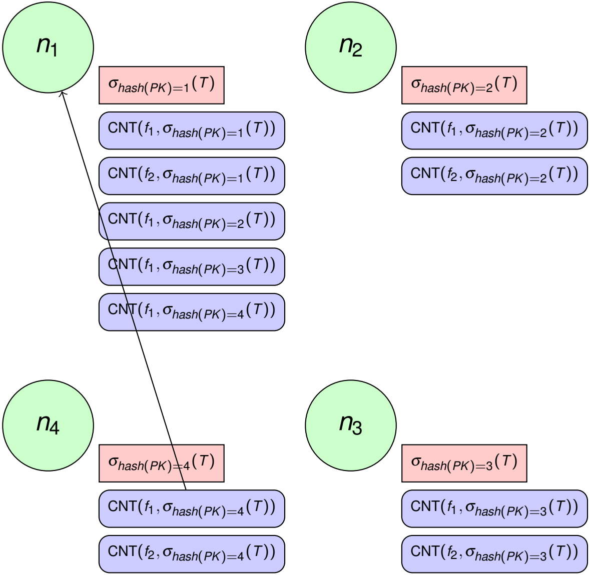
`DAgg(`n_1`,` f_1`,` f_2`,` $\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2`,` f_1`,` f_2`,` $\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3`,` f_1`,` f_2`,` $\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4`,` f_1`,` f_2`,` $\sigma_{hash(PK)=4}(T)$`)`
`transmit(`n_2`,` CNT$(f_1, \sigma_{hash(PK)=2}(T))$`,` n_1`)`

DPPS Compilation: Proof of Concept



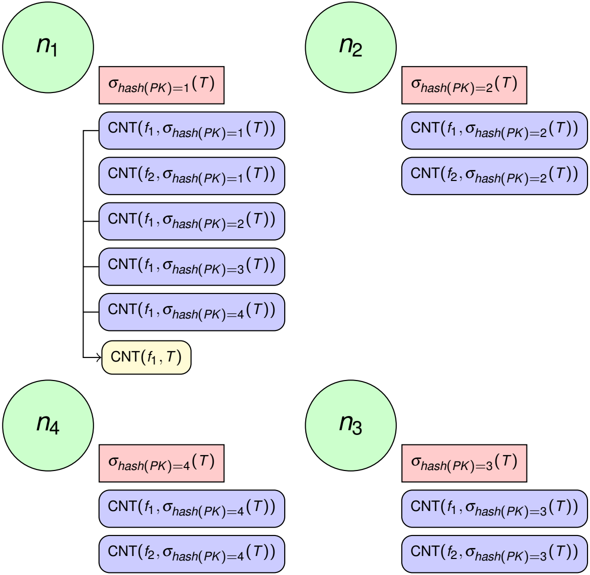
`DAgg(`n_1`,` f_1`,` f_2`,` $\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2`,` f_1`,` f_2`,` $\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3`,` f_1`,` f_2`,` $\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4`,` f_1`,` f_2`,` $\sigma_{hash(PK)=4}(T)$`)`
`transmit(`n_2`,` $\text{CNT}(f_1, \sigma_{hash(PK)=2}(T))$`,` n_1`)`
`transmit(`n_3`,` $\text{CNT}(f_1, \sigma_{hash(PK)=3}(T))$`,` n_1`)`

DPPS Compilation: Proof of Concept



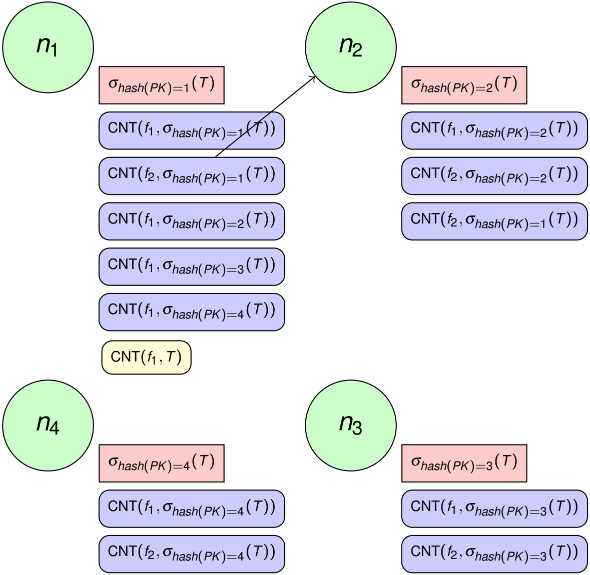
`DAgg(`n_1`,` f_1`,` f_2`,` $\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2`,` f_1`,` f_2`,` $\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3`,` f_1`,` f_2`,` $\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4`,` f_1`,` f_2`,` $\sigma_{hash(PK)=4}(T)$`)`
`transmit(`n_2`,` $\mathrm{CNT}(f_1, \sigma_{hash(PK)=2}(T))$`,` n_1`)`
`transmit(`n_3`,` $\mathrm{CNT}(f_1, \sigma_{hash(PK)=3}(T))$`,` n_1`)`
`transmit(`n_4`,` $\mathrm{CNT}(f_1, \sigma_{hash(PK)=4}(T))$`,` n_1`)`

DPPS Compilation: Proof of Concept



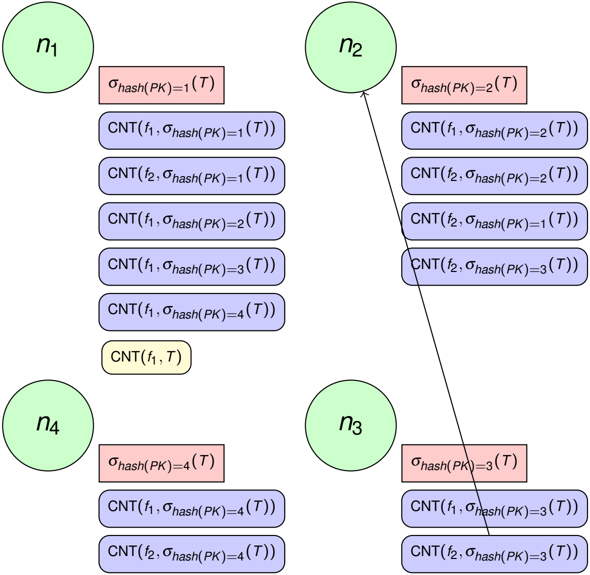
`DAgg(`n_1`, `f_1`, `f_2`,` $\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2`, `f_1`, `f_2`,` $\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3`, `f_1`, `f_2`,` $\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4`, `f_1`, `f_2`,` $\sigma_{hash(PK)=4}(T)$`)`
`transmit(`n_2`,` $\mathrm{CNT}(f_1, \sigma_{hash(PK)=2}(T))$, n_1`)`
`transmit(`n_3`,` $\mathrm{CNT}(f_1, \sigma_{hash(PK)=3}(T))$, n_1`)`
`transmit(`n_4`,` $\mathrm{CNT}(f_1, \sigma_{hash(PK)=4}(T))$, n_1`)`
`merge(`n_1`, `f_1`)`

DPPS Compilation: Proof of Concept



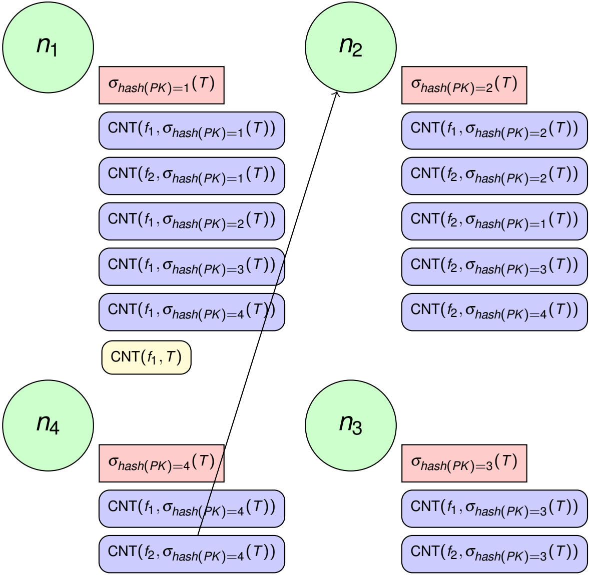
`DAgg(`n_1, f_1, f_2, $\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2, f_1, f_2, $\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3, f_1, f_2, $\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4, f_1, f_2, $\sigma_{hash(PK)=4}(T)$`)`
`transmit(`n_2, CNT(f_1, $\sigma_{hash(PK)=2}(T)$), n_1`)`
`transmit(`n_3, CNT(f_1, $\sigma_{hash(PK)=3}(T)$), n_1`)`
`transmit(`n_4, CNT(f_1, $\sigma_{hash(PK)=4}(T)$), n_1`)`
`merge(`n_1, f_1`)`
`transmit(`n_1, CNT(f_2, $\sigma_{hash(PK)=1}(T)$), n_2`)`

DPPS Compilation: Proof of Concept



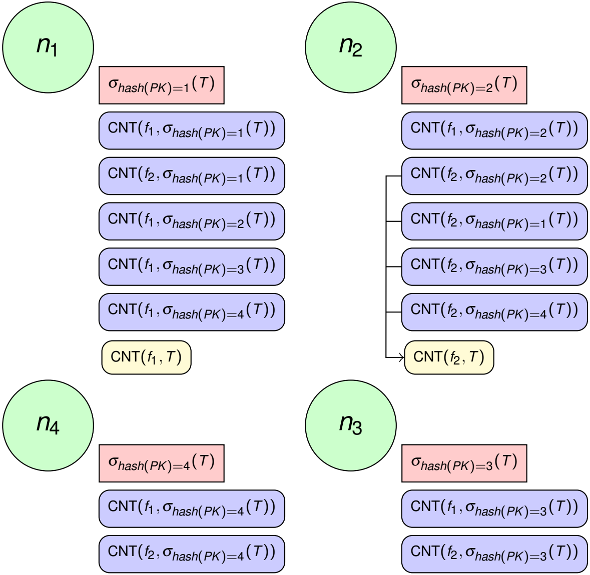
`DAgg(`n_1`, `f_1`, `f_2`,`
$\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2`, `f_1`, `f_2`,`
$\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3`, `f_1`, `f_2`,`
$\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4`, `f_1`, `f_2`,`
$\sigma_{hash(PK)=4}(T)$`)`
`transmit(`n_2`,`
`CNT`$(f_1, \sigma_{hash(PK)=2}(T))$`,`
n_1`)`
`transmit(`n_3`,`
`CNT`$(f_1, \sigma_{hash(PK)=3}(T))$`,`
n_1`)`
`transmit(`n_4`,`
`CNT`$(f_1, \sigma_{hash(PK)=4}(T))$`,`
n_1`)`
`merge(`n_1`, `f_1`)`
`transmit(`n_1`,`
`CNT`$(f_2, \sigma_{hash(PK)=1}(T))$`,`
n_2`)`
`transmit(`n_3`,`
`CNT`$(f_2, \sigma_{hash(PK)=3}(T))$`,`
n_2`)`

DPPS Compilation: Proof of Concept



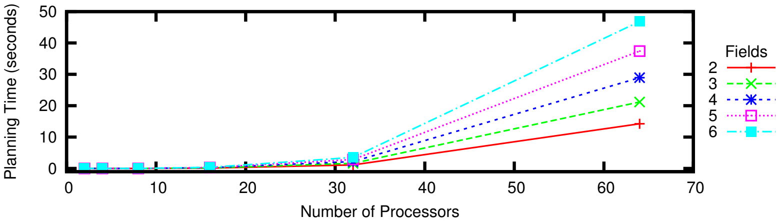
`DAgg(`n_1, f_1, f_2, $\sigma_{hash(PK)=1}(T)$`)`
`DAgg(`n_2, f_1, f_2, $\sigma_{hash(PK)=2}(T)$`)`
`DAgg(`n_3, f_1, f_2, $\sigma_{hash(PK)=3}(T)$`)`
`DAgg(`n_4, f_1, f_2, $\sigma_{hash(PK)=4}(T)$`)`
`transmit(`n_2, `CNT`$(f_1, \sigma_{hash(PK)=2}(T))$, n_1`)`
`transmit(`n_3, `CNT`$(f_1, \sigma_{hash(PK)=3}(T))$, n_1`)`
`transmit(`n_4, `CNT`$(f_1, \sigma_{hash(PK)=4}(T))$, n_1`)`
`merge(`n_1, f_1`)`
`transmit(`n_1, `CNT`$(f_2, \sigma_{hash(PK)=1}(T))$, n_2`)`
`transmit(`n_3, `CNT`$(f_2, \sigma_{hash(PK)=3}(T))$, n_2`)`
`transmit(`n_4, `CNT`$(f_2, \sigma_{hash(PK)=4}(T))$, n_2`)`

DPPS Compilation: Proof of Concept



$\mathtt{DAgg}(n_1, f_1, f_2, \sigma_{hash(PK)=1}(T))$
$\mathtt{DAgg}(n_2, f_1, f_2, \sigma_{hash(PK)=2}(T))$
$\mathtt{DAgg}(n_3, f_1, f_2, \sigma_{hash(PK)=3}(T))$
$\mathtt{DAgg}(n_4, f_1, f_2, \sigma_{hash(PK)=4}(T))$
$\mathtt{transmit}(n_2, \mathrm{CNT}(f_1, \sigma_{hash(PK)=2}(T)), n_1)$
$\mathtt{transmit}(n_3, \mathrm{CNT}(f_1, \sigma_{hash(PK)=3}(T)), n_1)$
$\mathtt{transmit}(n_4, \mathrm{CNT}(f_1, \sigma_{hash(PK)=4}(T)), n_1)$
$\mathtt{merge}(n_1, f_1)$
$\mathtt{transmit}(n_1, \mathrm{CNT}(f_2, \sigma_{hash(PK)=1}(T)), n_2)$
$\mathtt{transmit}(n_3, \mathrm{CNT}(f_2, \sigma_{hash(PK)=3}(T)), n_2)$
$\mathtt{transmit}(n_4, \mathrm{CNT}(f_2, \sigma_{hash(PK)=4}(T)), n_2)$
$\mathtt{merge}(n_2, f_2)$

DPPS Compilation: Proof of Concept



- Histogram of *F* fields of a table divided across *N* processors
- Solved by GBFS using relaxed plan heuristic in Fast Downward
- Solutions were optimal (although this is not guaranteed)

Summary

- DPPS is a flexible framework for describing data-parallel computations
- Solving DPPS is possible through compilation to AI planning
- We expect DPPS to lead to interesting questions in AI planning

Thank You