Parallelization in Python

Arnav Sood

NYU

April 21, 2016

Tasks

Split programs into **tasks**, which are atomic units of computation. Run these independently.

Tasks

Split programs into **tasks**, which are atomic units of computation.
Run these independently.
Example: `(add, 2, 3)`

Tasks

Split programs into **tasks**, which are atomic units of computation.
Run these independently.
Example: `(add, 2, 3)`
Literals not always required: `(add,x,y)`, `(x = (add,2,3))`

Tasks

Split programs into **tasks**, which are atomic units of computation.
Run these independently.
Example: `(add, 2, 3)`
Literals not always required: `(add,x,y)`, `(x = (add,2,3))`
Have dependency: write $(add, x, y) \rightarrow (x = (add, 2, 3))$

Tasks

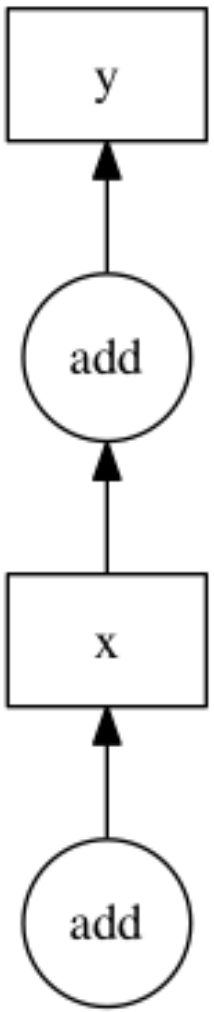
Split programs into **tasks**, which are atomic units of computation.
Run these independently.
Example: `(add, 2, 3)`
Literals not always required: `(add,x,y), (x = (add,2,3))`
Have dependency: write $(add, x, y) \rightarrow (x = (add, 2, 3))$
Induces graph. Specifically **directed, acyclic graph**. (why?)

Directed Acyclic Graphs

Represent dependency relationship between tasks.

Directed Acyclic Graphs

Represent dependency relationship between tasks.
Example: $(add, x, 3); (x = (add, 2, 3))$



Finding f: Tasks → Worker

Worker returns with data. What task do we give it?

Finding f: Tasks → Worker

Worker returns with data. What task do we give it?

- Task upon which most data depends?

Finding f: Tasks → Worker

Worker returns with data. What task do we give it?

- Task upon which most data depends?
- Task upon which most memory depends?

Finding f: Tasks → Worker

Worker returns with data. What task do we give it?

- Task upon which most data depends?
- Task upon which most memory depends?
- Task with data nearest to worker?

Finding f: Tasks $\rightarrow$ Worker

Worker returns with data. What task do we give it?

- Task upon which most data depends?
- Task upon which most memory depends?
- Task with data nearest to worker?
- ...∞ possible heuristics.

Finding f: Tasks → Worker

Worker returns with data. What task do we give it?

- Task upon which most data depends?
- Task upon which most memory depends?
- Task with data nearest to worker?
- ...∞ possible heuristics.

Dask Answer: Some combination of the above.

Embarrassingly Parallel Computation

Not all problems are equally parallelizable.

Embarrassingly Parallel Computation

Not all problems are equally parallelizable.
For example, rank the following:

- Multiplying two (large) matrices.
- Sorting a list.
- Finding the top k elements of a list, given some norm.

Embarrassingly Parallel Computation

Not all problems are equally parallelizable.
For example, rank the following:

- Multiplying two (large) matrices.
- Sorting a list.
- Finding the top k elements of a list, given some norm.

Results:

- Embarrassing.
- Hard.
- Could be made embarrassing.

Lazy evaluation

Idea: Don't do operations that don't matter for the user (for the requested result).

Lazy evaluation

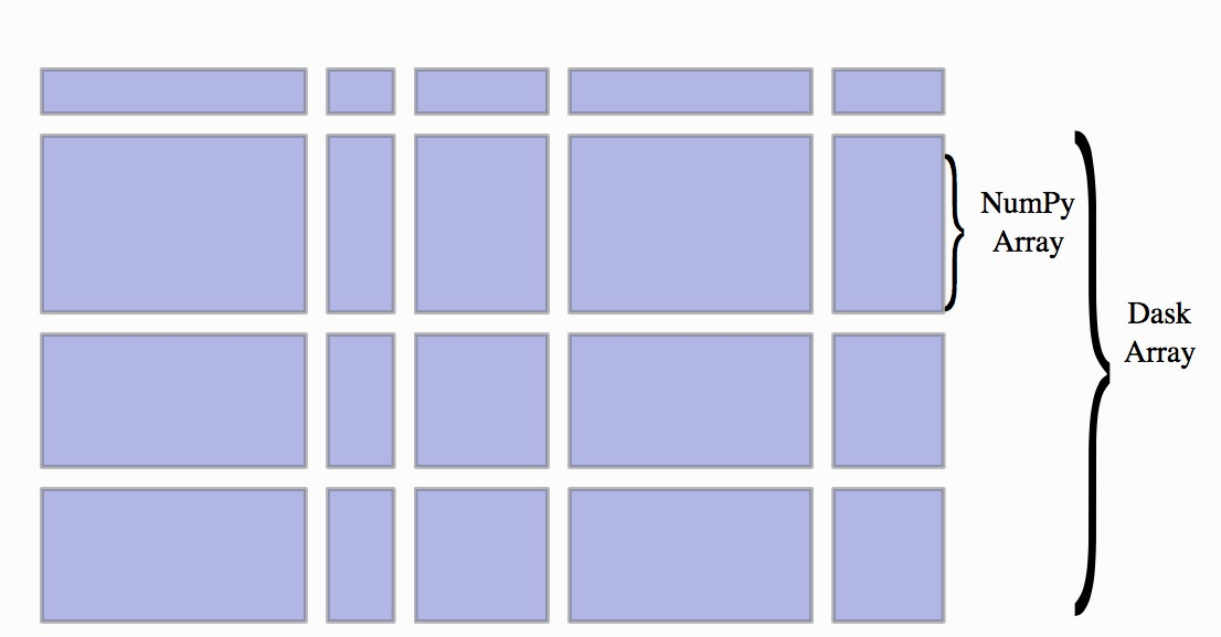
Idea: Don't do operations that don't matter for the user (for the requested result).
Catch: When you specify a custom graph (i.e., a Dask graph), all computations will be run.

Chunked Algorithms

Idea: Split matrix into smaller chunks, all stored in memory. Operate on those.

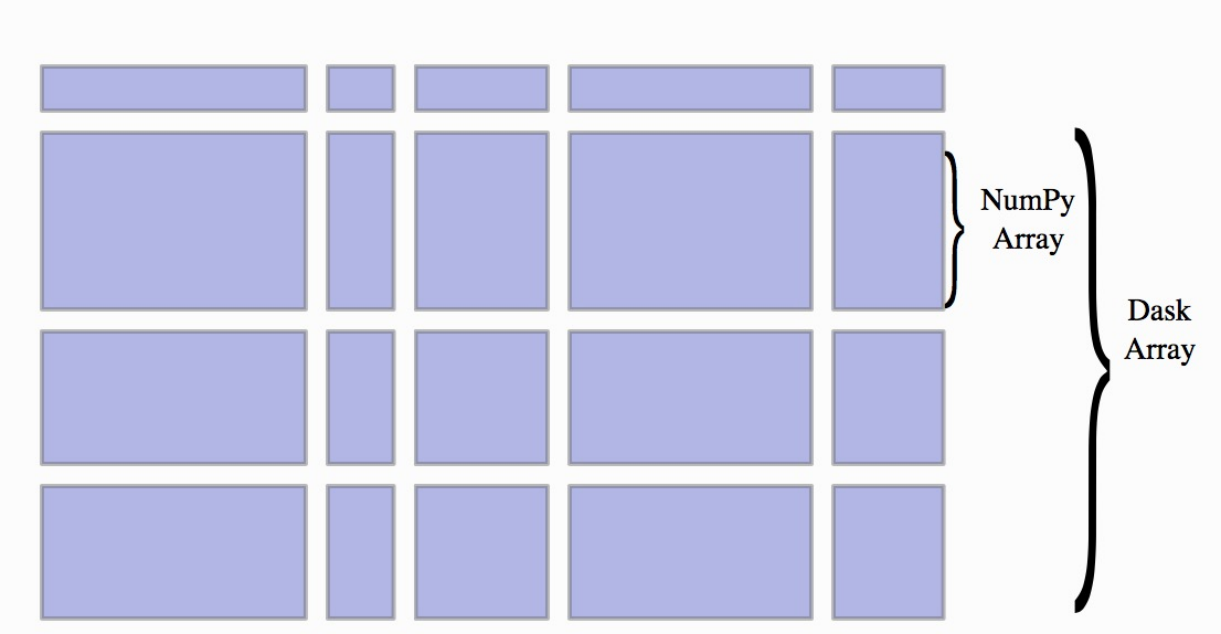
Chunked Algorithms

Idea: Split matrix into smaller chunks, all stored in memory. Operate on those.



Chunked Algorithms

Idea: Split matrix into smaller chunks, all stored in memory. Operate on those.



One Application: Your matrix is too big to load in memory.

Python Language Features

- Threads: Like virtual interpreter instances, operating in same memory space.

Python Language Features

- Threads: Like virtual interpreter instances, operating in same memory space.
- Processes: More heavyweight, seperate memory spaces.

Python Language Features

- Threads: Like virtual interpreter instances, operating in same memory space.
- Processes: More heavyweight, seperate memory spaces.
- Global Interpreter Lock: One thread touches the interpreter (and therefore the stack, all objects in memory, etc.) at a time.

Motivation