Spartan: A Distributed Array-Programming Framework with Automatic Tiling

Chien-Chin Huang, Qi Chen, Zhaoguo Wang, Russell Power, Jorge Ortiz, Jinyang Li, Zhen Xiao

NYU  PEKING UNIVERSITY  IBM
Big data problems compute with arrays

Machine Learning

Scientific Computing

Computational Finance

N-dimensional Arrays
Why are array programs loved?

• High-level, array-oriented abstractions.
  • Variable represent arrays.
  • Built-ins that directly compute on arrays.

```python
class neural_network(object):
    def forward_propagation(a1):
        a2 = np.dot(self.w1, a1)
        a2 = sigmoid(a2 + self.b1)
        a3 = np.dot(self.w2, a2)
        a3 = a3 + self.b2
        return a2, a3, softmax(a3)
```
No good way to distribute array programs

- MapReduce is designed for key-value collections.

- Existing distributed arrays require manual performance tuning.

**Presto (distributed R)**

**PETSc**

**SciDB**
Tiling: The performance challenge of distributed arrays

\[ C = A + B \]

\[ C = A1 + A2 + B \]

\[ C = A1 + B1' + B \]

\[ C = A2 + B2 + B \]
Manual tiling is painful

• Applications consist of a large number of expressions.

• Expressions use hundreds of built-in library functions.

• N-dimensional arrays have many ways of tiling.
Spartan’s goal: automatic tiling

No manual tiling!!
Outline

Motivation

Spartan’s Design

Evaluation
Challenge #1: Looking at multiple expressions at a time

\[ C = A + B \]

\[ E = C + D \]

- How to tile \( A, B, C, E \)?
#1. Expression tree captures multiple expressions

- Lazy evaluation captures expressions to build a dependency graph.

```
C = A + B
E = C + D
```

...
Challenge #2: Understanding array access pattern of different functions

- Numpy supports hundreds of array APIs.
#2. High-level operators capture access patterns

add, multiply, divide, log

min, max, sum, mean

dot, diagonal, ravel

filtering

cumsum, cumprod

…*

*Spartan provides 70+ builtins
High-level operators: map

- $C = \text{map}(f_{\text{map}}, A, B)$
- E.g. used to implement addition ($f_{\text{map}} = \text{“+”}$)
High-level operators: join_update

- $C = \text{join_update}(f_{\text{join}}, f_{\text{accum}}, A, \text{axis}_A, B, \text{axis}_B)$
- E.g. used to implement array multiplication ($f_{\text{join}}=\ldots, f_{\text{accum}}=\"+\", \text{axis}_A=\"column\", \text{axis}_B=\"row\")$
Expression graph is made up of high-level operators

\[
C = A + B \\
E = C + D
\]
Challenge #3: How to tile an expression graph?

• *Observation*: High-level operators have known tiling costs.

• *Solution*: Transform the expression graph into a tiling graph to explicitly capture tiling’s communication cost.
#3. Transform expression graph to tiling graph

\[ C = A + B \]
Greedily search for good tiling

- Finding best tiling is NP-Complete (proof in paper)
- We greedily search by choosing tiling for the most connected operator first

\[
\text{cost} = \Theta\text{sizeof (D)}
\]
Recap: how Spartan works

Built-in library

High-level Operators

new array
new array
map
join_update
map

Decide tiling

Spartan Frontend

C = A + B
E = C + D

Spartan Backend

• schedule tasks
• create distributed arrays

Schedule tasks
Outline

• Motivation

• Spartan Design

• Evaluation
High-level operators are expressive

- 70+ Numpy builtins.
- 10 machine learning + 2 computational finance.
Spartan is scalable

- Experimental setup: 256 workers on 128 EC2 large instances.
Spartan is fast

- Experimental setup: 256 workers on 128 EC2 large instances, k-means application, using the best tiling.

<table>
<thead>
<tr>
<th>Relative performance</th>
<th>Presto (Distributed R)</th>
<th>Spark</th>
<th>SciDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spartan</td>
<td>1.7X</td>
<td>10.1X</td>
<td>40X</td>
</tr>
</tbody>
</table>
The performance effect of a bad tiling

- Experimental setup: 256 workers on 128 EC2 large instances.
Related work

• Manual tiling:
  - Global Array Toolkit [Nieplocha ’96]
  - PETSc [Balay ’97]
  - MadLinq[Qian ’12]
  - Distributed R (Presto) [Venkataraman ’13]
  - Elemental [Poulson ’13]

• Compiler-assisted tiling:
  - [Hudak ’90]
  - [Li’ 90]
  - [Li ’91]
  - [Kennedy ’91]
  - [Kremer ’93]
  - [Philippsen ’95]
Conclusions

• Spartan is a distributed array programming framework with automatic tiling.
  • Expression graphs capture multiple expressions.
  • High-level operators expose array access pattern.

• https://github.com/spartan-array/spartan
Limitations of Spartan

• Tiling cost is not always precise
  • Join\_update
• Sparse array
  • Can be solved by using run-time analyzing technique.
• Spartan does not support in-place array modification.
# High-level operators: known tiling cost

<table>
<thead>
<tr>
<th>Operator</th>
<th>0</th>
<th>sizeof(input)</th>
<th>sizeof(input)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>0</td>
<td>sizeof(input)</td>
<td>sizeof(input)</td>
<td>0</td>
</tr>
<tr>
<td>Fold</td>
<td>0</td>
<td>sizeof(input)</td>
<td>sizeof(input)</td>
<td>0</td>
</tr>
<tr>
<td>Join_Update</td>
<td>sizeof(output) + sizeof(input)</td>
<td>sizeof(output) + sizeof(input)</td>
<td>sizeof(output)</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>0</td>
<td>sizeof(input)</td>
<td>sizeof(input)</td>
<td>0</td>
</tr>
<tr>
<td>filter</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Tiling performance

- Spartan Tiling
- Best Tiling


Executed time (seconds)

- 0
- 275
- 550
- 825
- 1100

30
High-level operators: fold

- \( C = \text{fold}(f_{\text{fold}}, A, \text{axis}) \)
- E.g. max value for each row (\( f_{\text{fold}} = \text{max}, \text{axis} = 1 \))

![Diagram](image)

\( f_{\text{fold}} = \text{max} \)
Random tiling comparison

![Graph showing network transmissions vs. experiment index for best and spartan tilings.]

- **Best Tiling**
- **Spartan Tiling (differing results shown)**