On-the-fly Optimization of Parallel Computation of Symbolic Symplectic Invariants

Joseph Ben Geloun, *Camille Coti*, Allen D. Malony

LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, France
University of Oregon, USA

Séminaire LATECE, UQAM
April 7th, 2022

Roadmap

Outline

Group invariants

Group invariants

- define quantum field theory interaction

To **quantize gravity**

- tensor models
- address classical Lie group (unitary and orthogonal) invariants in their construction.

Classical Lie groups :

- orthogonal
- unitary
- real **symplectic group** $Sp(2N)$

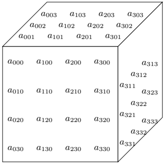
Goal : **enumerate symplectic invariants**

- Applications : e.g., problems in condensed matter, black hole physics

Matrix :

a_{14}	a_{24}	a_{34}	a_{44}
a_{13}	a_{23}	a_{33}	a_{43}
a_{12}	a_{22}	a_{32}	a_{42}
a_{11}	a_{21}	a_{31}	a_{41}

Tensor :



Calculation of symplectic invariants

Relies on **tensor contraction**

- the tensor contains formal, real variables
- result : a polynomial made of these variables
- **is the invariant equal to 0 ?**

Example : rank $d = 3$

- contraction of 4 tensors
- called *complete graph contraction*

Goal : show that for $N > 3$, the invariant is not identically null

Calculation of symplectic invariants (1/2)

Symplectic matrix J

- size $2N \times 2N$

$$J = \begin{pmatrix} 0 & I_N \\ -I_N & 0 \end{pmatrix}, \qquad J^2 = -I_{2N}, \tag{1}$$

with :

- I_N, for all N the identity matrix of $M_N(\mathbb{R})$
- A matrix $K \in Sp(2N)$ obeys $KJK^T = J$, and $K^T JK = J$

Interactions of $Sp(2N)$ tensor models :

- contractions of an even number of tensors T
- contraction metric : the matrix J

Calculation of symplectic invariants (2/2)

Physical applications :

- in particular, contraction of 4 tensors as follows :

$$T^4 = \sum_{\substack{a_1,\dots,a_6 \\ \bar{a}_1,\dots,\bar{a}_6}} [\prod_{i=1}^{6} J_{a_i \bar{a}_i}] T_{a_1,a_2,a_3} T_{a_4,a_5,\bar{a}_3} T_{\bar{a}_4,\bar{a}_2,a_6} T_{\bar{a}_1,\bar{a}_5,\bar{a}_6} \qquad (2)$$

- for all c, a_c and $\bar{a}_c \in [1, 2N]$
- T^4 denotes the invariant

Algorithm 1 : "Naive"

```
Tens = 0
for a1 ← 0, size do
 for a2 ← 0, size do
  for a3 ← 0, size do
   A = T[a1][a2][a3]
   for b1 ← 0, size do
    TAB = J[a1][b1]
    for b2 ← 0, size do
     for b3 ← 0, size do
      TABB = TAB * A * T[b1][b2][b3]
      for c1 ← 0, size do
       for c2 ← 0, size do
        TABC = TABB * J[a2][c2]
        for c3 ← 0, size do
         TABCC = TABC * T[c1][c2][c3] * J[b3][c3]
         for d1 ← 0, size do
          TABCD = TABCC * J[c1][d1]
          for d2 ← 0, size do
           TABCDD = TABCD * J[b2][d2]
           for d3 ← 0, size do
            Tens = Tens + TABCDD * T[d1][d2][d3] * J[a3][d3]
           end for
          end for
         end for
        end for
       end for
      end for
     end for
    end for
   end for
  end for
 end for
end for
```

12 (TWELVE) nested loops

Some algebraic properties

The matrix J has some *symmetries* :

$$J = \begin{pmatrix} 0 & I_N \\ -I_N & 0 \end{pmatrix}, \qquad J^2 = -I_{2N}, \tag{3}$$

Hence the tensor contraction becomes :

$$T^4 = \sum_{I \subset \{1,2,\dots,6\}} (-1)^{6-|I|} \prod_{l \in I} \Big[\sum_{a_l, \bar{a}_l} \delta_{\bar{a}_l, a_l+N} \Big] \times \prod_{l \notin I} \Big[\sum_{a_l, \bar{a}_l} \delta_{a_l, \bar{a}_l+N} \Big] T_{a_1,a_2,a_3} T_{a_4,a_5,\bar{a}_3} T_{\bar{a}_4,\bar{a}_2,a_6} T_{\bar{a}_1,\bar{a}_4,\bar{a}_6} .$$

Algorithm 2 : exploiting the aforementioned properties

```
Tens = TE = T1 = T2 = T3 = T4 = T5 =
T12 = T13 = T14 = T16 = T23 = T24 = T26 =
T123 = T126 = T134 = 0
N = size/2
for a4 ←, N do
 A4 = a4 + N
 for a2 ← 0, N do
  A2 = a2 + N
  for a6 ← 0, N do
   A6 = a6 + N
   W1 = T[a4][a2][a6]
   W2 = T[a4][a2][a6]
   W3 = T[a4][a2][A6]
   W4 = T[a4][a2][a6]
   W5 = T[a4][a2][A6]
   W6 = T[a4][a2][A6]
   W7 = T[a4][a2][A6]
   for a1 ← 0, N do
    A1 = a1 + N
    for a5 ← 0, N do
     A5 = a5 + N
     Z1 = T[a1][a5][a6]
     Z2 = T[a1][a5][a6]
     Z6 = T[a1][a5][A6]
     T5 = W3 ∗ T[a1][A5][a6]
     TE = W4 ∗ T[a1][A5][A6]
     T1 = W3 ∗ Z2
     T13 = T1
     T2 = W5 ∗ Z1
     T23 = T2
     T3 = W3 ∗ Z1
     T4 = W6 ∗ Z1
     T12 = W5 ∗ Z2
     T14 = W6 ∗ Z2
     T134 = T14
     T16 = W1 ∗ Z6
     T24 = W7 ∗ Z1
     T26 = W2 ∗ T[a1][a5][A6]
     T123 = W5 ∗ Z2
     T126 = W2 ∗ Z6
     for a3 ← 0, N do
      A3 = a3 + N
      TE+ = TE ∗ T[a1][a2][a3] ∗ T[a4][a5][a3]
      T5+ = T5 ∗ T[a1][a2][a3] ∗ T[a4][a5][a3]
      X7Y5 = T[a1][a2][a3] ∗ T[a4][A5][a3]
      T1+ = T1 ∗ X7Y5
      T16+ = T16 ∗ X7Y5
      T2+ = T2 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T3+ = T3 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T4+ = T4 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T12+ = T12 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T13+ = T13 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T14+ = T14 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T23+ = T23 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T24+ = T24 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      T26+ = T26 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
T123+ = T123 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
T126+ = T126 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
T134+ = T134 ∗ T[a1][a2][a3] ∗ T[a4][A5][a3]
      end for
     end for
    end for
   end for
  end for
end for
Tens = 4∗(TE+T12+T13+T14+T16+T23+T24+
T26−(T1+T2+T3+T4+T5+T123+T126+T134))
```

“Only” 6 nested loops

Outline

Symbolic computation

Let's compute polynomial expressions by hand :

- $P_1 = x_1$, $P_2 = x_2$
  - $P_1 + P_2 = x_1 + x_2$

Symbolic computation

Let's compute polynomial expressions by hand :

- $P_1 = x_1$, $P_2 = x_2$
  - $P_1 + P_2 = x_1 + x_2$
- $P_1 = x_1 + x_2 + x_3 + x_4$, $P_2 = x_2$
  - $P_1 + P_2 = x_1 + 2 * x_2 + x_3 + x_4$

Symbolic computation

Let's compute polynomial expressions by hand :

- $P_1 = x_1$, $P_2 = x_2$
  - $P_1 + P_2 = x_1 + x_2$
- $P_1 = x_1 + x_2 + x_3 + x_4$, $P_2 = x_2$
  - $P_1 + P_2 = x_1 + 2 * x_2 + x_3 + x_4$
- $P_1 = x_1 + x_2 + x_3 + x_4$, $P_2 = x_2 + x_4 + x_6 + x_8$
  - $P_1 + P_2 = x_1 + 2 * x_2 + x_3 + 2 * x_4 + x_6 + x_8$

Symbolic computation

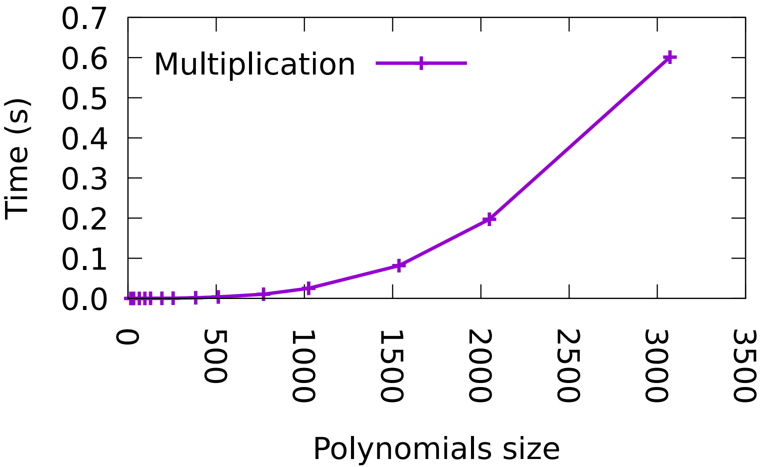
Let's compute polynomial expressions by hand :

- $P_1 = x_1,\ P_2 = x_2$
  - $P_1 + P_2 = x_1 + x_2$
- $P_1 = x_1 + x_2 + x_3 + x_4,\ P_2 = x_2$
  - $P_1 + P_2 = x_1 + 2 * x_2 + x_3 + x_4$
- $P_1 = x_1 + x_2 + x_3 + x_4,\ P_2 = x_2 + x_4 + x_6 + x_8$
  - $P_1 + P_2 = x_1 + 2 * x_2 + x_3 + 2 * x_4 + x_6 + x_8$
- $P_1 = x_1 + x_2 + x_3 + x_4,\ P_2 = -x_1 - x_2 - x_3 - x_4$
  - $P_1 + P_2 = x_1 - x_1 + x_2 - x_2 + x_3 - x_3 + x_4 - x_4$
  - $P_1 + P_2 = 0 + 0 + 0 + 0$

$\rightarrow$ **The computation time depends on the number of elements in the polynomial**

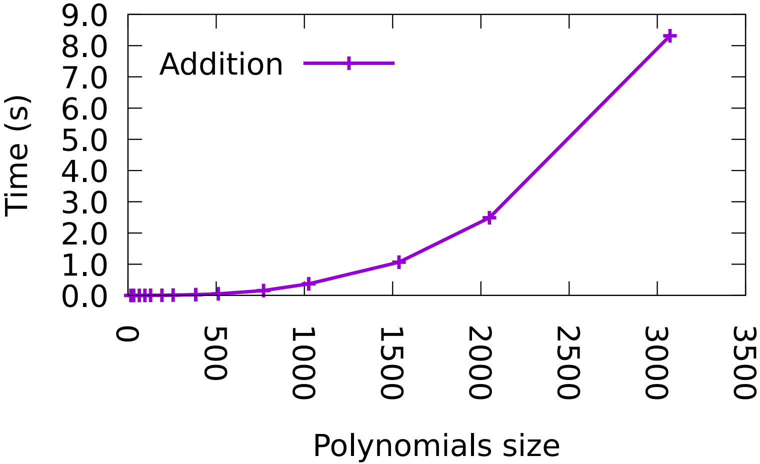
Computation time : multiplication

Operation type : $P_1 = 2 * P_2$



Computation time : addition

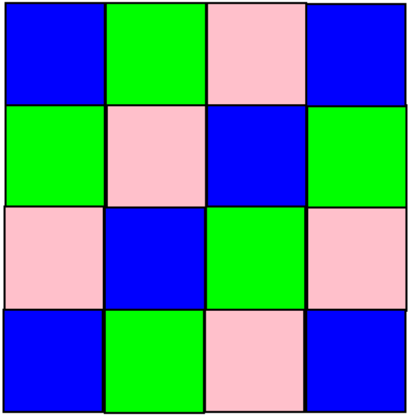
Operation type : $P_1 = P_2 + P_3$



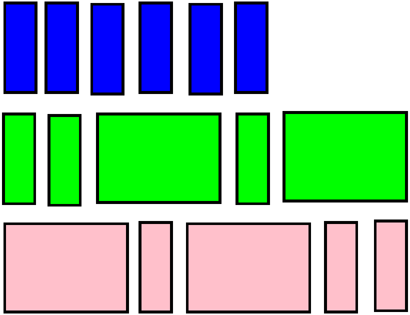
Outline

Parallelizing the computation : domain decomposition

Naive approach : domain decomposition



Problem : not all the subparts of the matrix will take the same time



Parallelizing the computation : load balancing

Other problem : remember the computation time of our polynomials

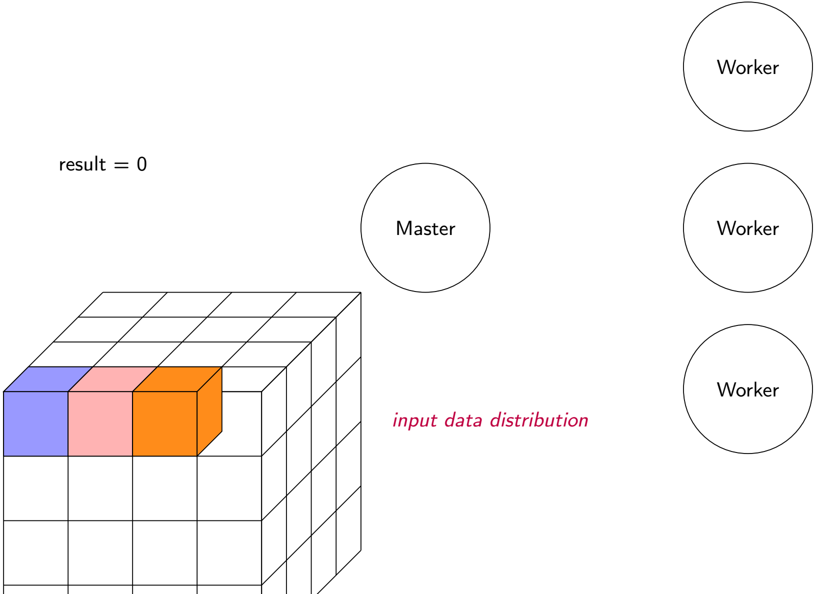
- Process 1 : $x_1 + x_2 - x_1 - x_2$
- Process 2 : $x_1 + x_2 + x_1 + x3$

$\rightarrow$ Process 1 will compute a lot faster than process 2

We need automatic, dynamic load balancing

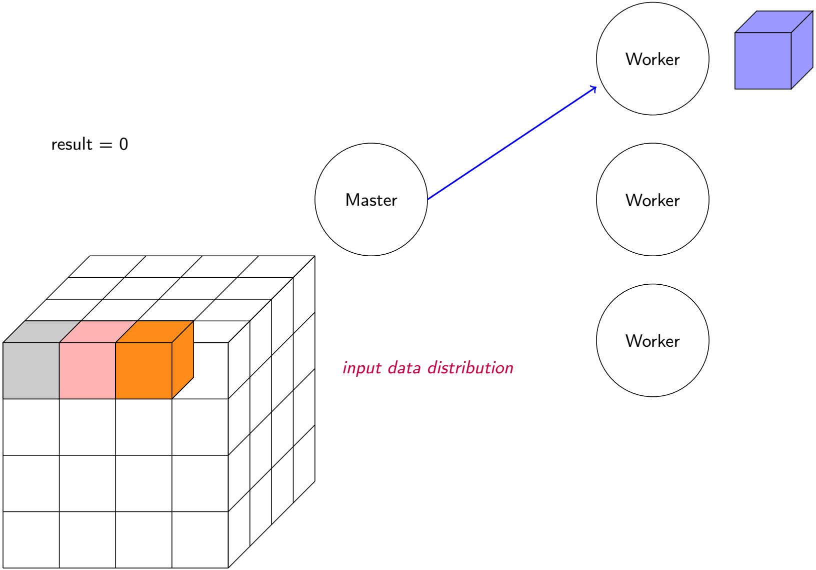
$\rightarrow$ Use a master-worker scheme.

Master-worker ?

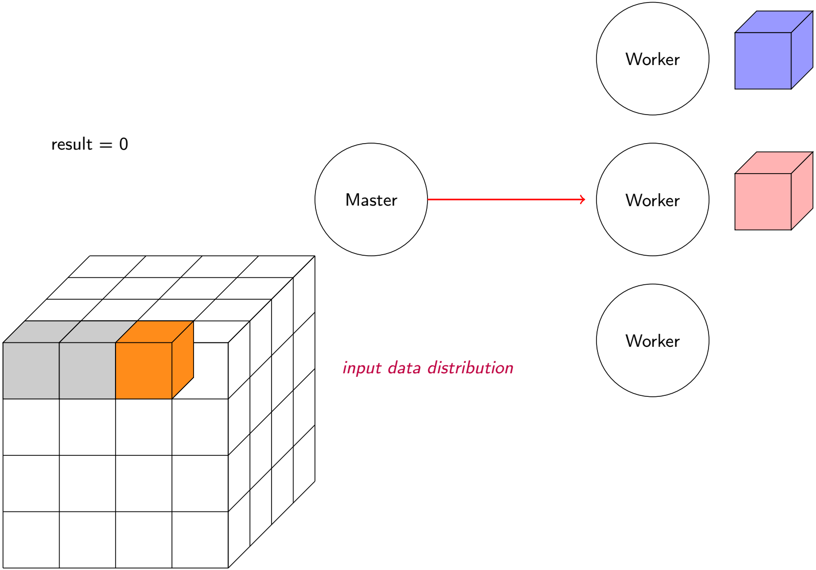


Master-worker ?

result = 0

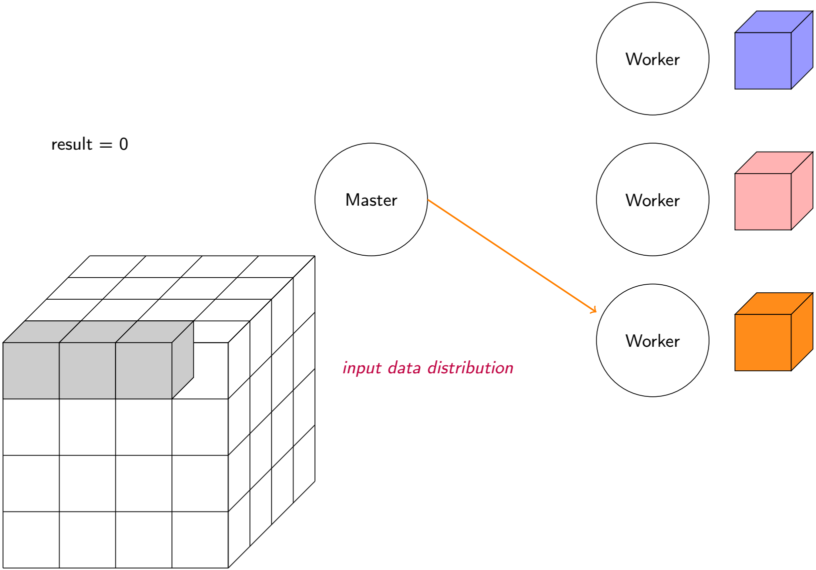


Master-worker ?



Master-worker ?

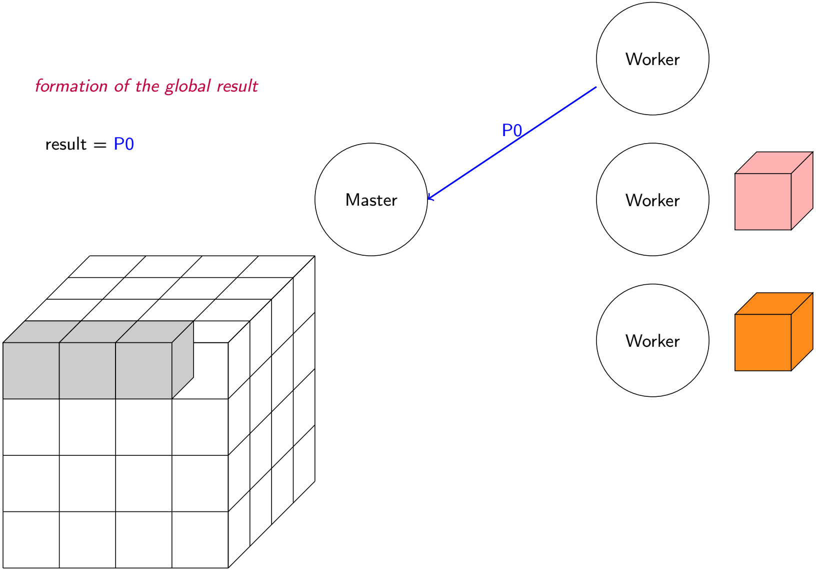
result = 0



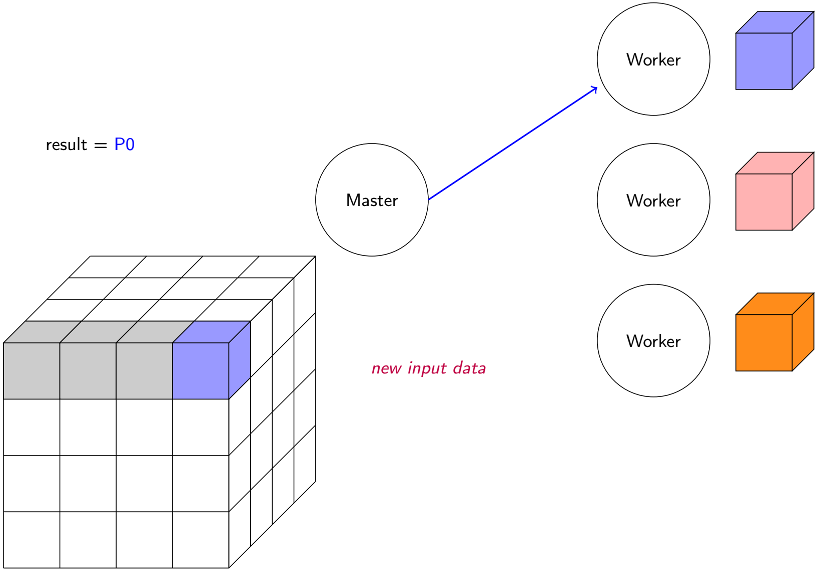
Master-worker ?

formation of the global result

result = P0



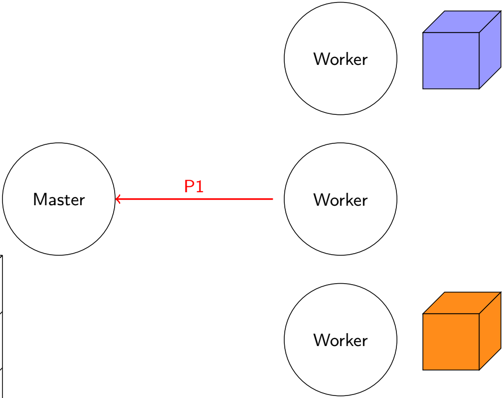
Master-worker ?

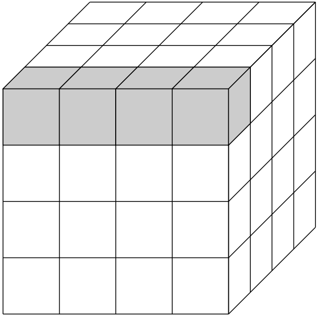


Master-worker ?

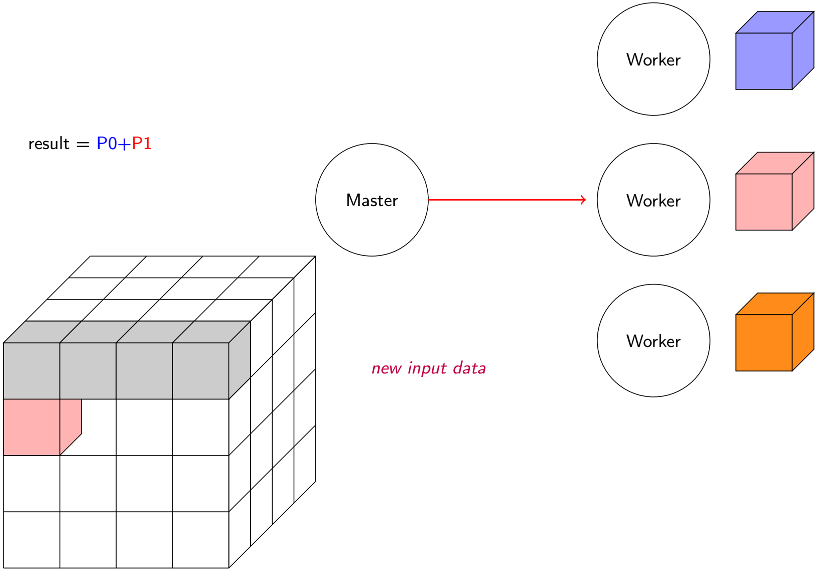
formation of the global result

result = P0+P1

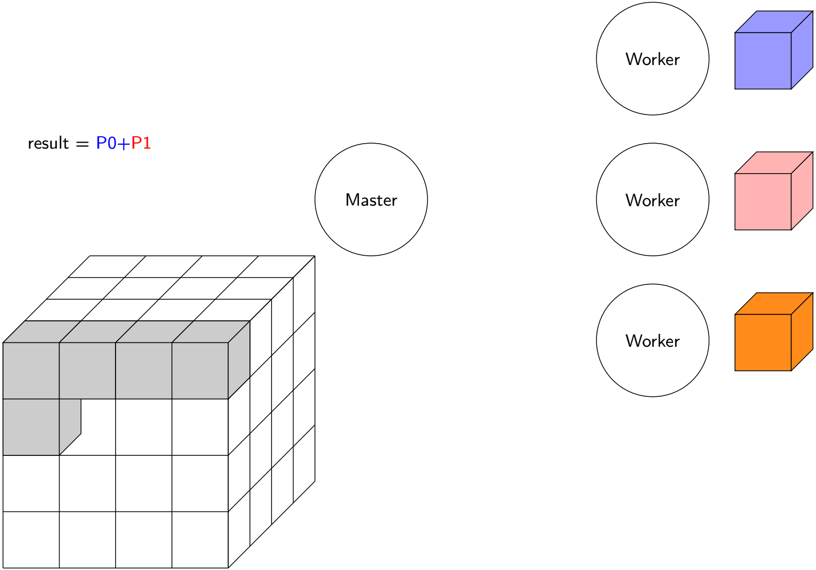




Master-worker ?



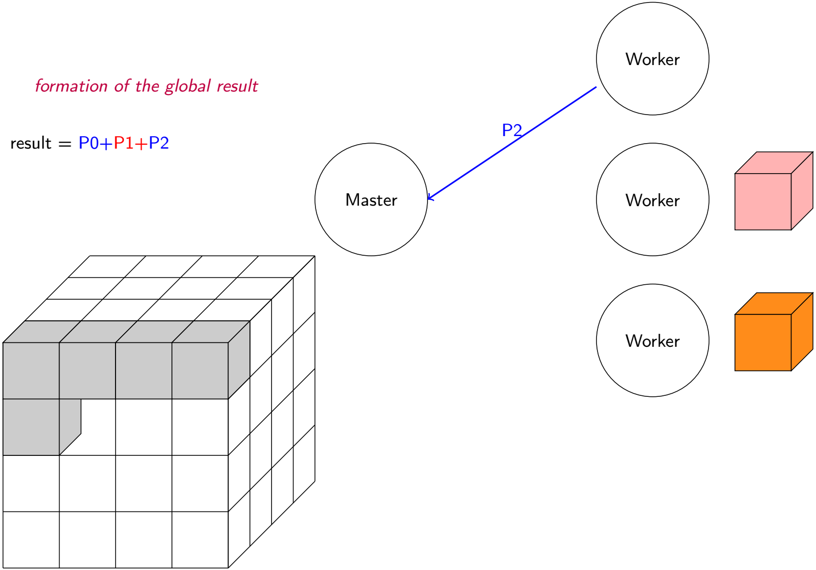
Master-worker ?



Master-worker ?

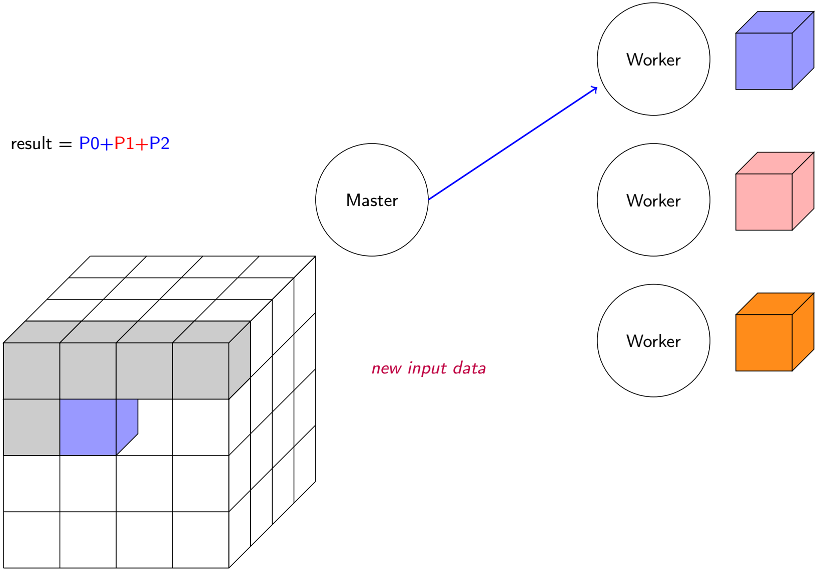
formation of the global result

result = P0+P1+P2

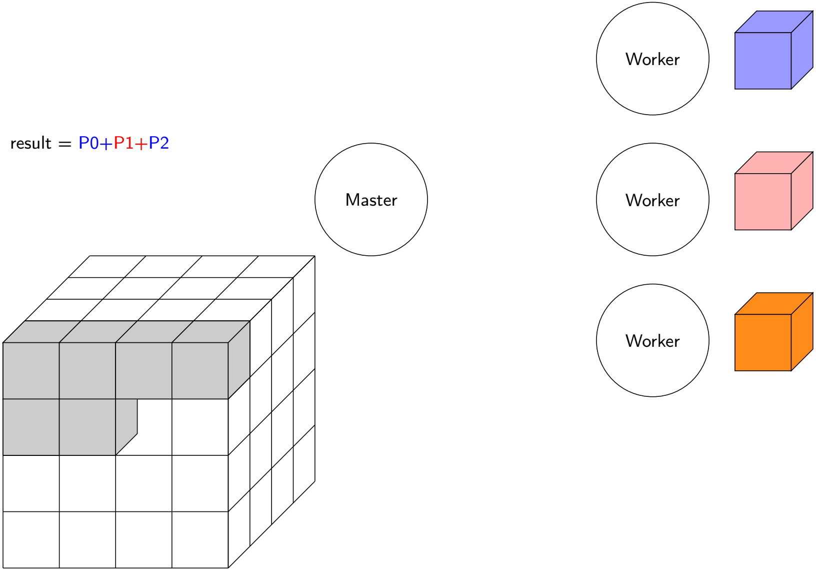


Master-worker ?

result = P0+P1+P2



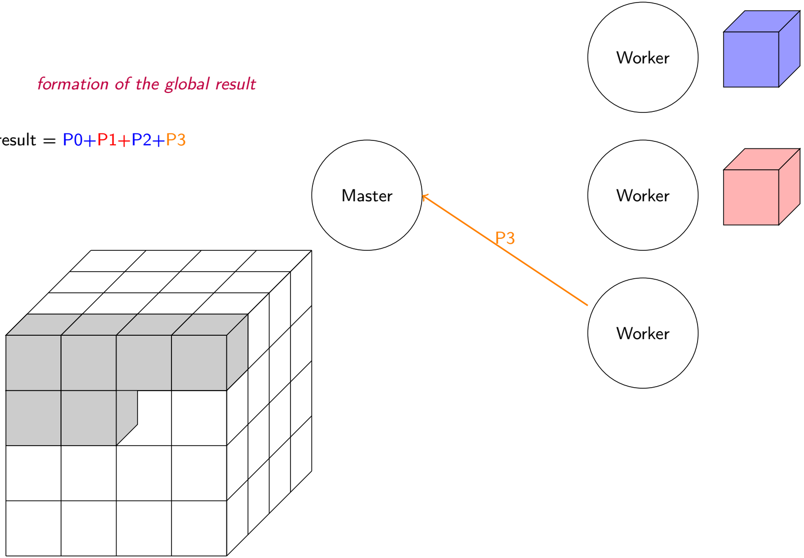
Master-worker ?



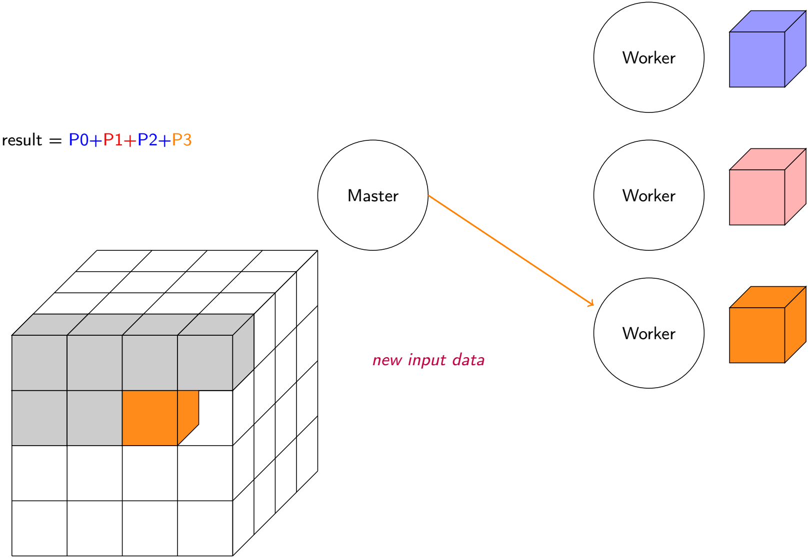
Master-worker ?

formation of the global result

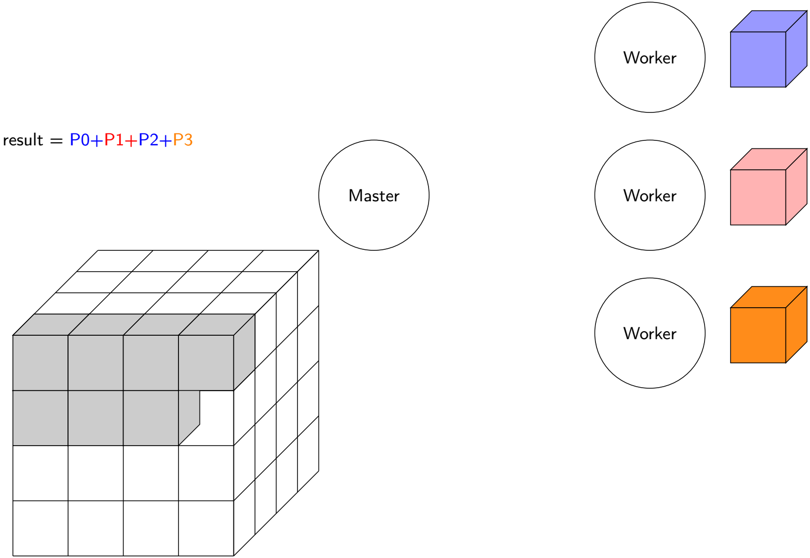
result = P0+P1+P2+P3



Master-worker ?



Master-worker ?



Master-worker ?

Traditional master-worker :

- The master maintains two queues : data and results
- The master sends chunks of data to the workers
- The workers compute partial sums
- The master gets results from the workers, **combines them to form the global result**

In our case :

- The workers send partial sums
- The master adds them to form the **global sum** (the invariant)

Problem : this global sum gets bigger and bigger

$\rightarrow$ Bottleneck on the master, busy adding polynomials

Delegate the sum

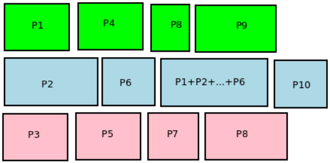
Bottleneck on the master

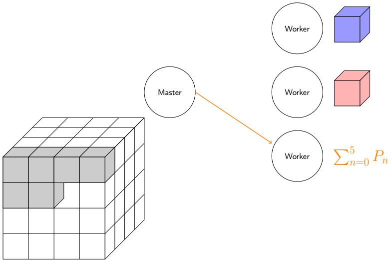
- Ask a worker to compute this sum
- The workers can have two different types of tasks :
  - Compute a partial sum (inner loops)
  - Add partial sums to form the global sum

However :

- Adds interactions between the master and a worker
- A worker computing the global sum is not computing any partial sum

→ Switch to this scheme when the global sums are too expensive





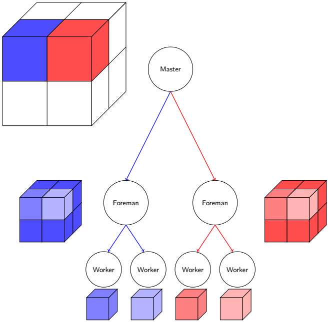
Hierarchical master-worker

Another reason why the master can become a bottleneck :

- Workers' requests are too frequent
- Granularity is too small, too many workers

→ Use a hierarchical pattern

  - The workers request work from a foremen
  - The foremen request work from the master
  - The master sends a bigger chunk of work to each foreman
  - The foremen split this chunk into smaller chunks
  - The foremen compute intermediate sums, or delegate to a worker



If the master becomes a bottleneck, how to tell why ?

- **Use measurements**
- Which time proportion does the master spend outside communications ?

Stateful worker

The main challenge is the computation of the global sum

- Do not centralize it on the master
- **Keep the partial sums on the workers**, add them while waiting for more data
- Only at the end, add them to form the global polynomial (tree)

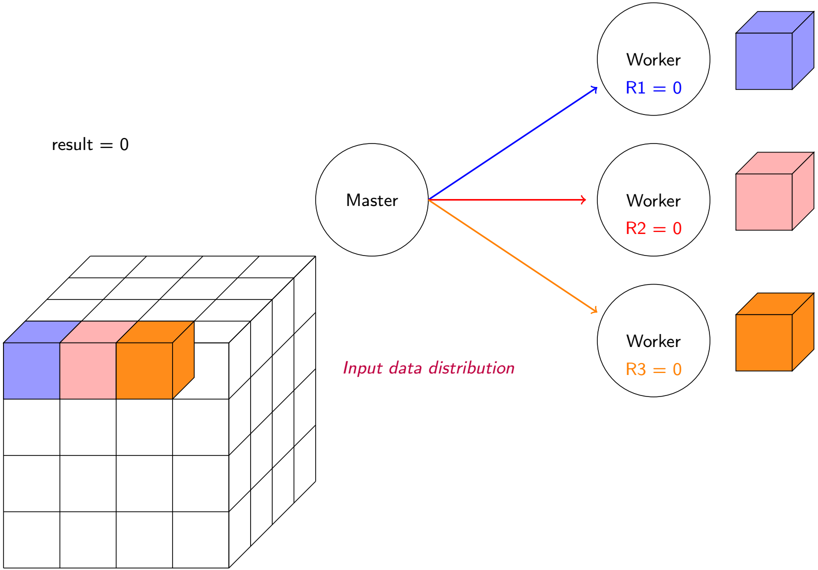
Algorithm 1 Master

```
/* prepare parameter sets */
for a4 ← 0, N do
 for a2 ← 0, N do
  for a6 ← 0, N do
   params.push_back({ a4,a2,a6)})
  end for
 end for
end for
 /* distribute parameter sets */
while !parameters.empty() do
 src = recv( request, ANY_SOURCE )
 p = params.pop()
 send( src, p, TAG_WORK )
/* wait for all the workers */
while running() do
 src = recv( request, ANY_SOURCE )
 send( src, 0, TAG_END )
/* global sum */
Tens = reduction_sum()
```

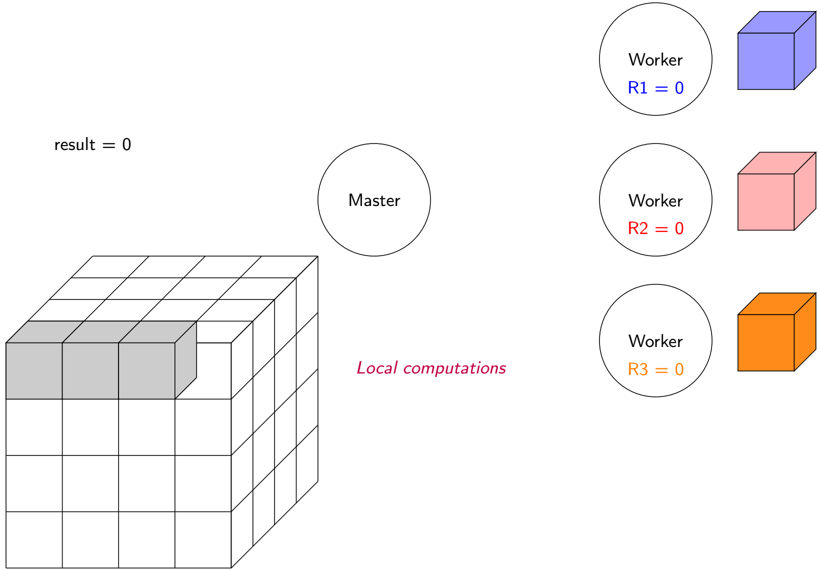
Algorithm 2 Worker

```
Tens = 0
T = 0
while true do
 /* ask for some work */
 send( root, 0, TAG_REQ )
 /* as I wait for a parameter set, add my polynomials
*/
 req = Irecv( ROOT, ANY_TAG )
 if T != 0 then
  Tens += T
 p, tag = wait( req )
 if tag == TAG_END then
  break
 /* compute a polynomial for the parameters I have
received */
 T = compute( p )
/* global sum */
reduction_sum(Tens)
```

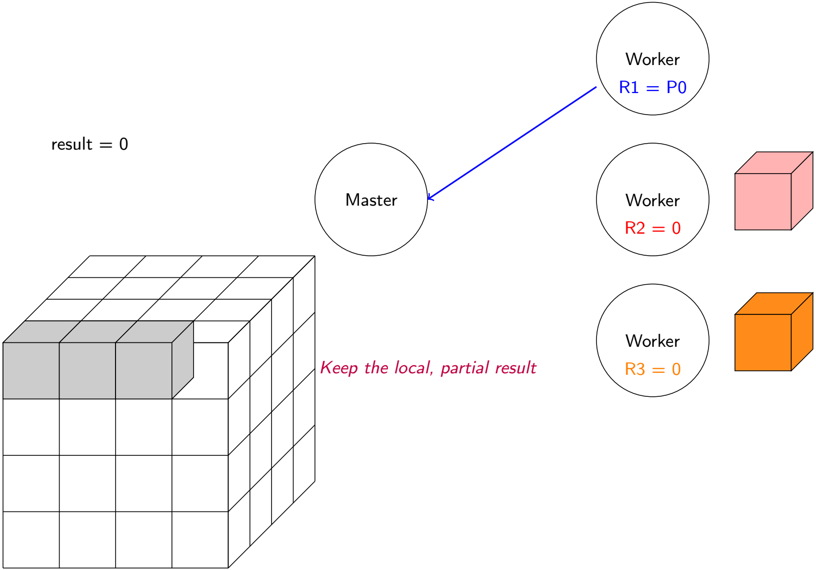
Stateful worker



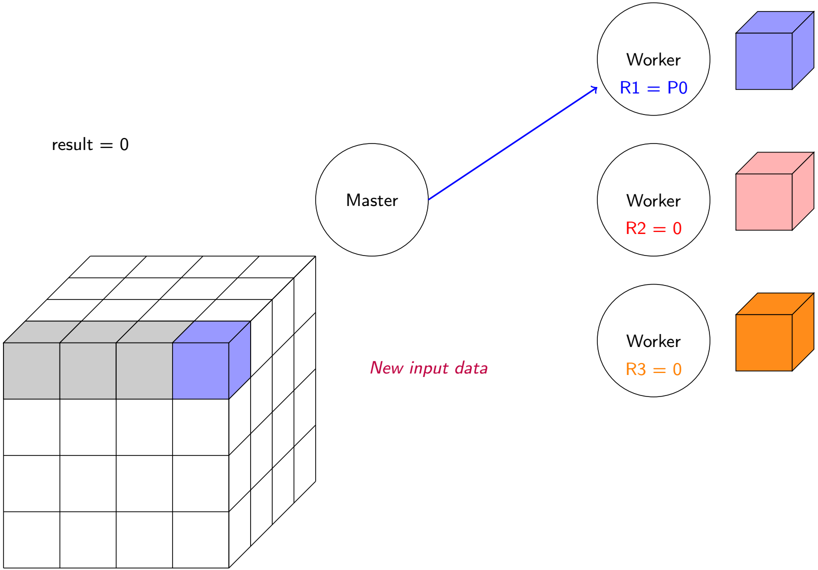
Stateful worker



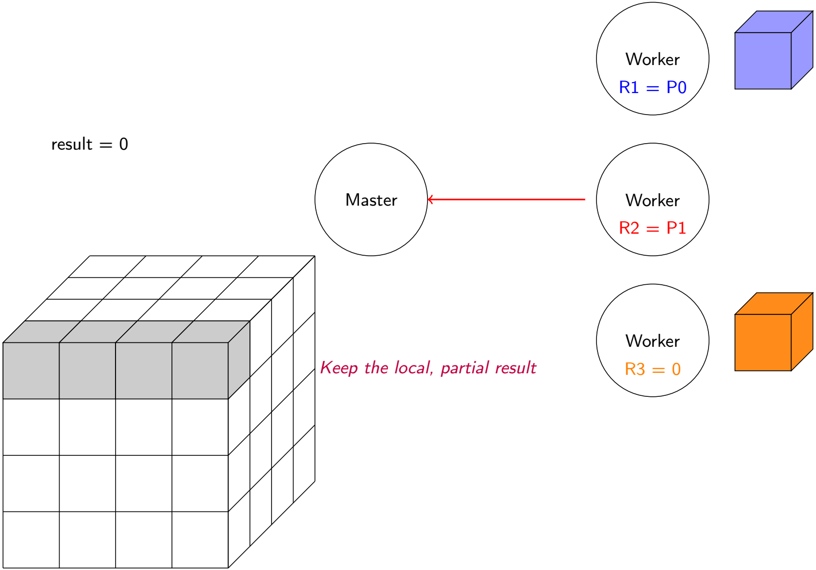
Stateful worker



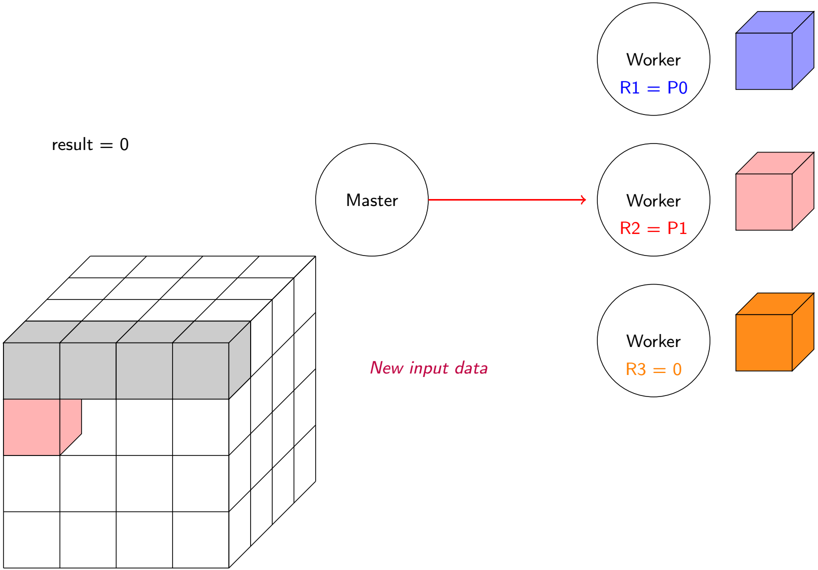
Stateful worker



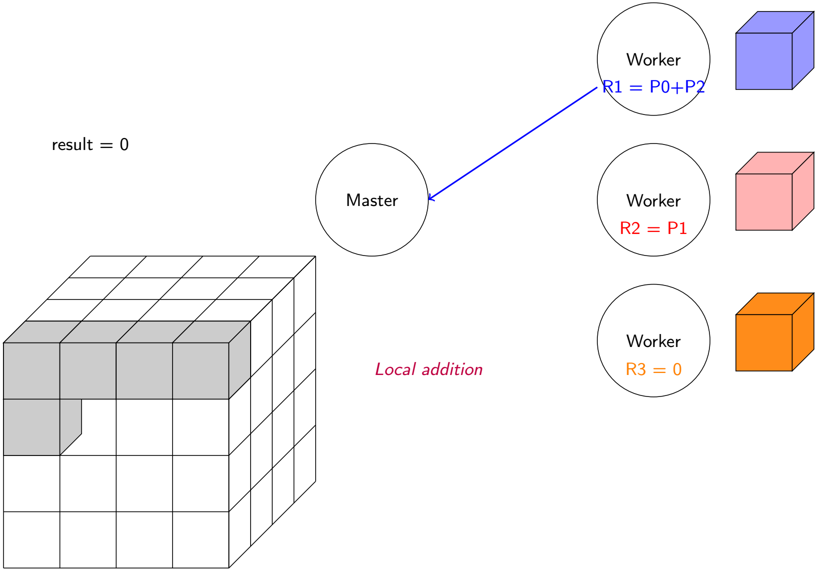
Stateful worker



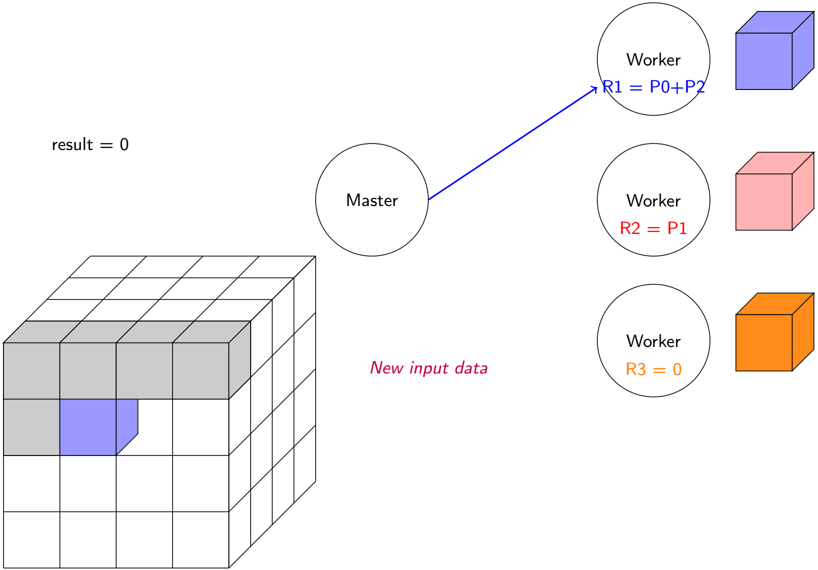
Stateful worker



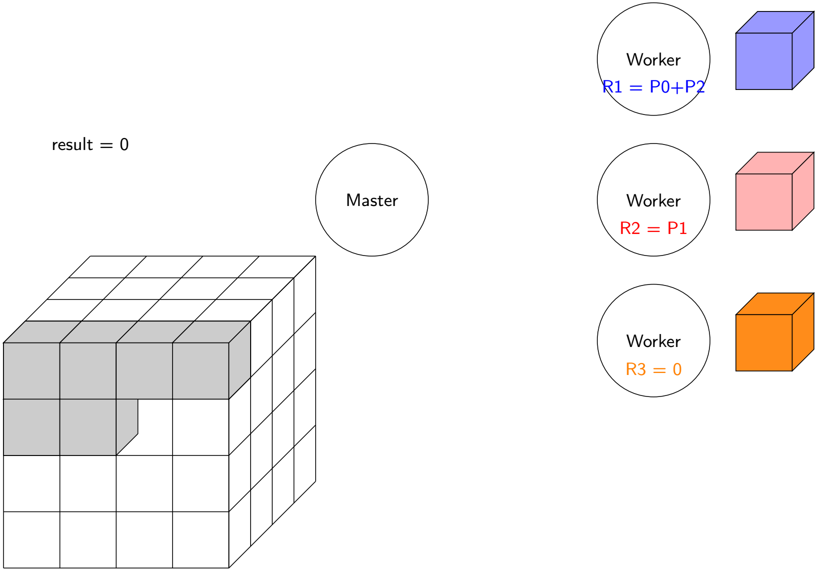
Stateful worker



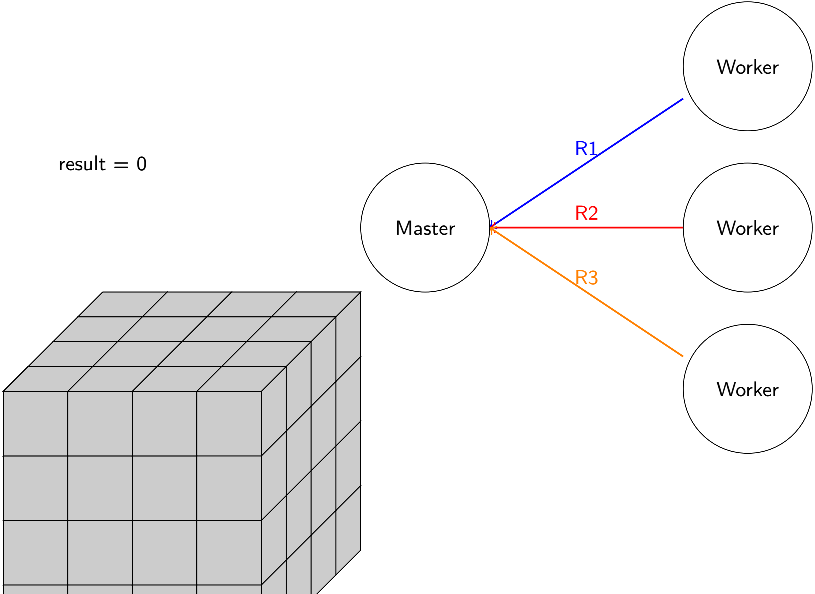
Stateful worker



Stateful worker



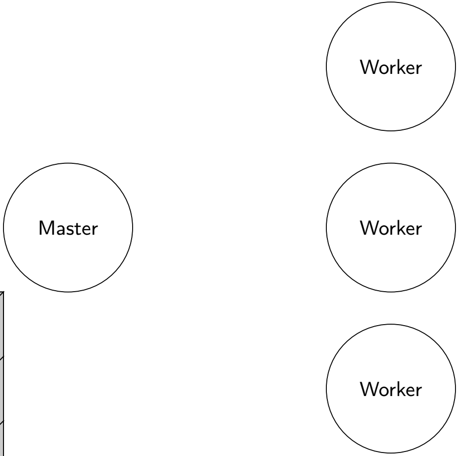
Stateful worker

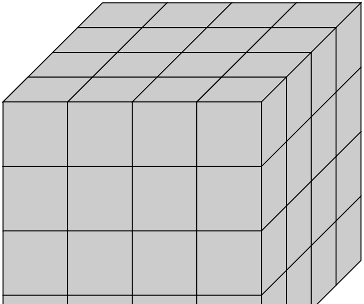


Stateful worker

Formation of the global result

result = R1+R2+R3





Outline

Experimental setup

Software environment

- OpenMPI 4.1, Linux kernel 4.9.0, Debian 9.8, g++ 8.3.0

Hardware

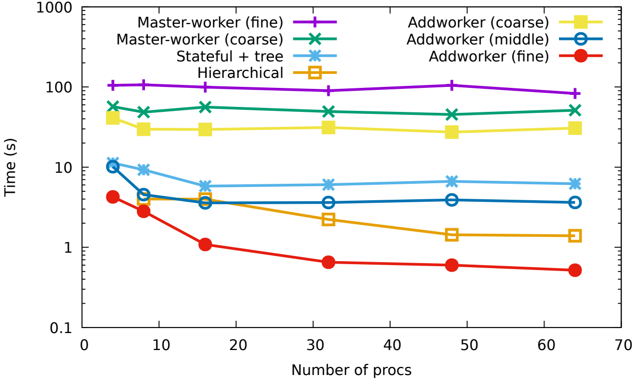
- Grid'5000 cluster : Parapide (Rennes)
- 20 nodes, 2x Intel Xeon X5570 CPUs (4 cores/CPU), 24 GB of memory
- 20 Gb InfiniBand + GigaEthernet

Symbolic computing libraries :

- GiNaC 1.7.6 (not Gignac !)
- Obake : successor of Piranha, better on multivariate polynomials

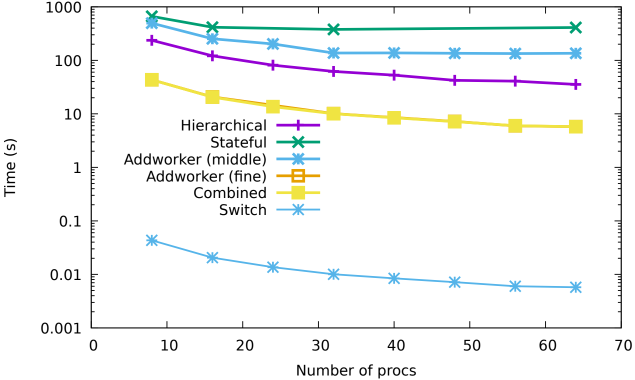
Small tensor (N=4, size=8)

Using Obake.



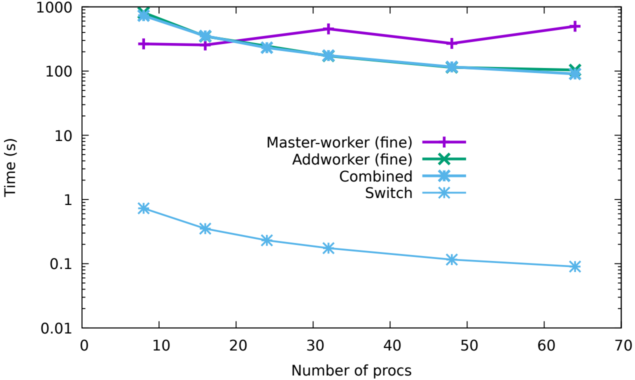
Medium tensor (N=6, size=12)

Using Obake. Blue bottom line : when the switch between *master-worker* and *addition on a worker* happens.



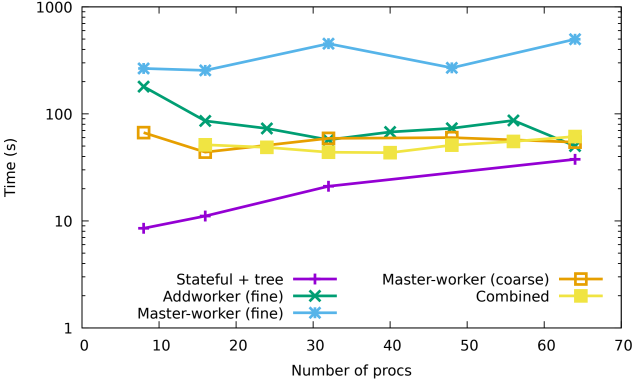
Large tensor (N=8, size=16) 1/2

Using Obake. At small scale : we switch too late.



Large tensor (N=8, size=16) 2/2

Using GiNaC : the polynomial operations do not take the same time.



Hierarchical ?

We have **never seen the algorithm switch to the hierarchical scheme**

- Policy : when the master is overloaded by requests $\rightarrow$ switch to the hierarchical scheme
- Maybe because when a lot of requests are received, a lot of additions are needed (intermediate and partial)

Outline

Conclusion 1/2

In this problem, the computational work **varies** during the computation

- We were not sure it would (annulling terms $\rightarrow$ reduced computation time)
- Increases in particular in the **critical path** (global sum)
- Non-linear

Goal : **get as much as we can away from the critical path**

Granularity of the computation :

- Increase the number of workers $\rightarrow$ refine the granularity to keep them busy
  - Too small grain $\rightarrow$ computation time too short wrt communications

Scalability :

- Increase the **size of the problem**
- Workers have more work
- **More (expensive) polynomial additions** (in the critical path)

Conclusion 2/2

Polynomial additions to for the global sum

- Become expensive quickly
  - **Switch to a pattern that computes them on a worker**
  - Good choice most of the times, switch quickly
- Stateful workers : much faster... **except to form the global polynomial**
  - most of the times its cost is higher than the gain during the computation.

Hierarchical scheme

- Never encountered a case where the switch policy applies
- The workload on each worker **increases faster than the congestion on the master** (as the size increases to scale)
- Larger problem $\rightarrow$ larger polynomials to add

Dynamic workload, evolving (roughly) monotonically : advantage of **run-time performance measurements** to make decisions.