

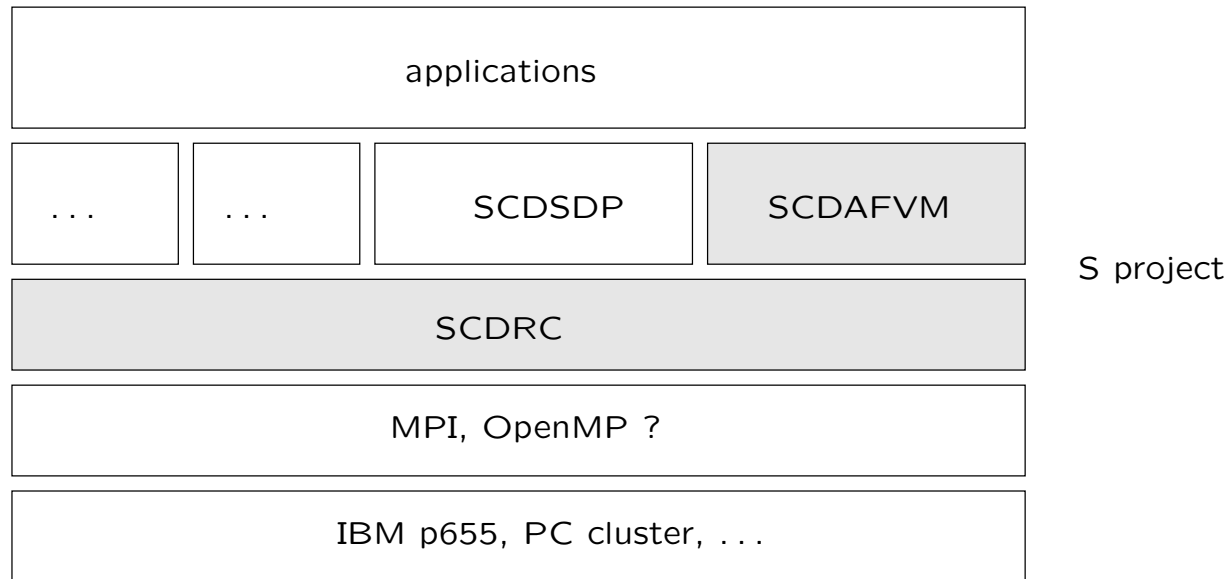
# Structuring distributed relation-based computations with SCDRC

N. Botta, C. Ionescu, C. Linstead, R. Klein

with contributions of:

- J. Gerlach, Fraunhofer Inst. für Rechnerarch. und Softwaretechnik.
- A. Priesnitz, Chalmers University of Technology.

## Software architecture and SCDRC



- SCDRC is a thin layer between message passing libraries and application dependent components.

**Why a software layer ?**

## Why a software layer ?

- Structuring parallel algorithms for adaptive grids, partial differential equations, sequential decision problems is difficult.

## Why a software layer ?

- Structuring parallel algorithms for adaptive grids, partial differential equations, sequential decision problems is difficult.
- Implementing such algorithms *in terms of* message passing libraries is
  - time-consuming,
  - error-prone,
  - requires knowledge of low-level message passing primitives.

**Why distributed relation-based computations ?**

## Why distributed relation-based computations ?

- Relation-based computations are non-trivially parallelizable generic computational patterns (algorithmic skeletons).

## Why distributed relation-based computations ?

- Relation-based computations are non-trivially parallelizable generic computational patterns (algorithmic skeletons).
- Many algorithms can be written in terms of relation-based computations.

## Why distributed relation-based computations ?

- Relation-based computations are non-trivially parallelizable generic computational patterns (algorithmic skeletons).
- Many algorithms can be written in terms of relation-based computations.
- If we are able to implement generic *distributed* relation-based computations, we can express parallel algorithms by combining specializations of DRCs.

## Pattern 1: relation-based algorithms

RBA

```
for  $j$  in  $js$  do
  compute  $h([f(i) \mid i \text{ in } R(j)])$ 
end for
```

## Pattern 1: relation-based algorithms

RBA

```
for  $j$  in  $js$  do
  compute  $h([f(i) \mid i \text{ in } R(j)])$ 
end for
```

can be specialized to compute triangle areas, centers, etc.

## Pattern 1: relation-based algorithms

RBA

```
for  $j$  in  $js$  do  
    compute  $h([f(i) \mid i \text{ in } R(j)])$   
end for
```

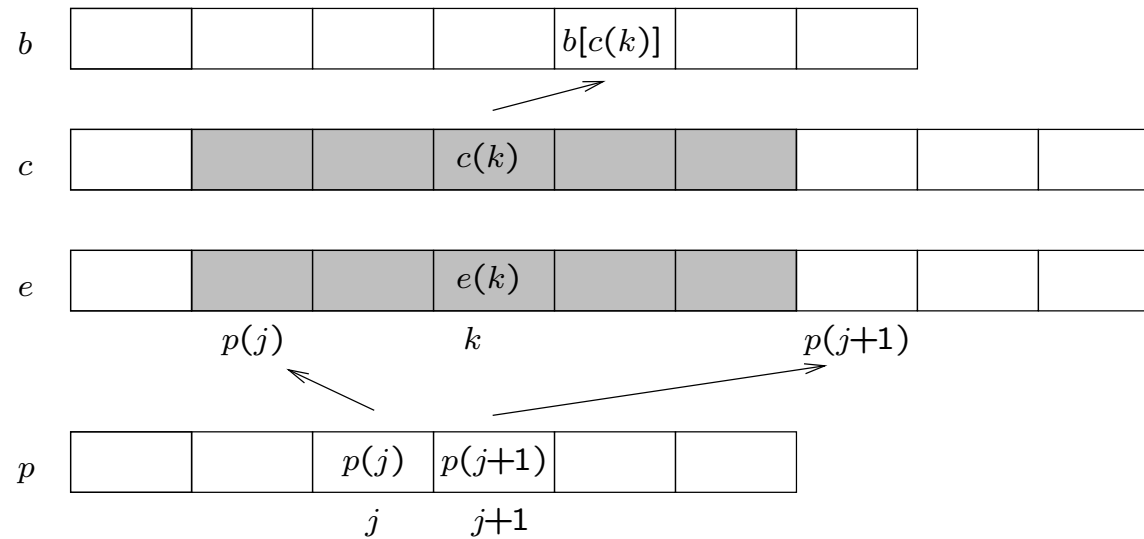
can be specialized to compute triangle areas, centers, etc.

```
 $ta = \text{RBA}(\text{area}, x, vt)$   
for  $j$  in  $[0 \dots \text{size}(vt)]$  do  
    compute  $\text{area}([x(vt(j, 0)), x(vt(j, 1)), x(vt(j, 2))])$   
end for
```

... or sparse matrix-vector multiplications:

mvmult

```
for  $j$  in  $[0 \dots n)$  do
  compute  $\text{sum}([e(k) * b[c(k)] \mid k \text{ in } [p(j) \dots p(j+1))])$ 
end for
```



One has:

$$\begin{aligned} & [e(k) * b[c(k)] \mid k \text{ in } [p(j) \dots p(j + 1)]] \\ & = \\ & [e(k) * b[i] \mid i = c(k), k \text{ in } [p(j) \dots p(j + 1)]] \end{aligned}$$

One has:

$$\begin{aligned} & [e(k) * b[c(k)] \mid k \text{ in } [p(j) \dots p(j + 1)]] \\ & = \\ & [e(k) * b[i] \mid i = c(k), k \text{ in } [p(j) \dots p(j + 1)]] \\ & = \{ \text{let } R(j) = [c(k) \mid k \text{ in } [p(j) \dots p(j + 1)]] \} \\ & [e(k) * b[i] \mid i \text{ in } R(j), k = p(j) + \text{pos}(i, R(j))] \end{aligned}$$

One has:

$$\begin{aligned} & [e(k) * b[c(k)] \mid k \mathbf{in} [p(j) \dots p(j + 1)]] \\ & = \\ & [e(k) * b[i] \mid i = c(k), k \mathbf{in} [p(j) \dots p(j + 1)]] \\ & = \{ \text{let } R(j) = [c(k) \mid k \mathbf{in} [p(j) \dots p(j + 1)]] \} \\ & [e(k) * b[i] \mid i \mathbf{in} R(j), k = p(j) + \text{pos}(i, R(j))] \\ & = \{ \text{let } g(i, j) = p(j) + \text{pos}(i, R(j)) \} \\ & [e(g(i, j)) * b[i] \mid i \mathbf{in} R(j)] \end{aligned}$$

One has:

$$\begin{aligned} & [e(k) * b[c(k)] \mid k \text{ in } [p(j) \dots p(j + 1)]] \\ & = \\ & [e(k) * b[i] \mid i = c(k), k \text{ in } [p(j) \dots p(j + 1)]] \\ & = \{ \text{let } R(j) = [c(k) \mid k \text{ in } [p(j) \dots p(j + 1)]] \} \\ & [e(k) * b[i] \mid i \text{ in } R(j), k = p(j) + \text{pos}(i, R(j))] \\ & = \{ \text{let } g(i, j) = p(j) + \text{pos}(i, R(j)) \} \\ & [e(g(i, j)) * b[i] \mid i \text{ in } R(j)] \\ & = \{ \text{let } f(i, j) = e(g(i, j)) * b[i] \} \\ & [f(i, j) \mid i \text{ in } R(j)] \end{aligned}$$

and therefore  $\text{mvmult} = \text{RBA}(\text{sum}, f, R)$ .

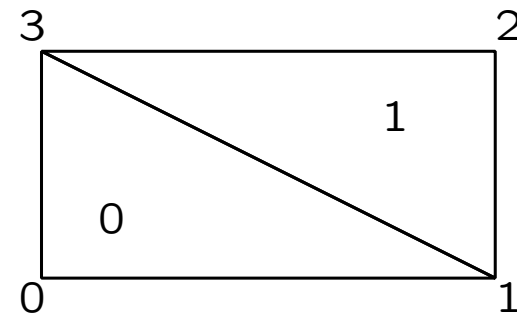
## Parallel distributed RBAs

- RBAs are non-trivially parallelizable if:
  - $f$  is represented by storing disjoint subsets of its graph  $f_0$   
 $\dots f_{np-1}$  on the partitions.

## Parallel distributed RBAs

- RBAs are non-trivially parallelizable if:
  - $f$  is represented by storing disjoint subsets of its graph  $f_0 \dots f_{np-1}$  on the partitions.
  - For any disjoint splitting  $R_0 \dots R_{np-1}$  of  $R$  there are partitions  $p, q$  for which  $\text{ran}(R_p) \cap \text{ran}(R_q) \neq \emptyset$ .

Example:  $R = [[0, 1, 3], [1, 2, 3]]$



## Parallel distributed RBAs

- To parallelize RBAs, one has to answer the questions:

## Parallel distributed RBAs

- To parallelize RBAs, one has to answer the questions:
  - How to distribute  $f$  and  $R$  on the partitions ?

## Parallel distributed RBAs

- To parallelize RBAs, one has to answer the questions:
  - How to distribute  $f$  and  $R$  on the partitions ?
  - How to compute, on a given partition, which values of  $f$  are needed but not locally owned ?

## Parallel distributed RBAs

- To parallelize RBAs, one has to answer the questions:
  - How to distribute  $f$  and  $R$  on the partitions ?
  - How to compute, on a given partition, which values of  $f$  are needed but not locally owned ?
  - How to obtain, on a given partition, such values from the partitions that own them ?

## Conceptual model of distributed functions and relations

- Functions represented as arrays, relations as arrays of arrays.

## Conceptual model of distributed functions and relations

- Functions represented as arrays, relations as arrays of arrays.
- Distributed functions:

$$\begin{array}{l} f = [(0, 0), (2, 0), (2, 1), (0, 1)] \\ of = [0, 2, 4] \end{array} \rightarrow \begin{array}{l} f_0 = [(0, 0), (2, 0)] \\ of_0 = [0, 2, 4] \\ f_1 = [(2, 1), (0, 1)] \\ of_1 = [0, 2, 4] \end{array}$$

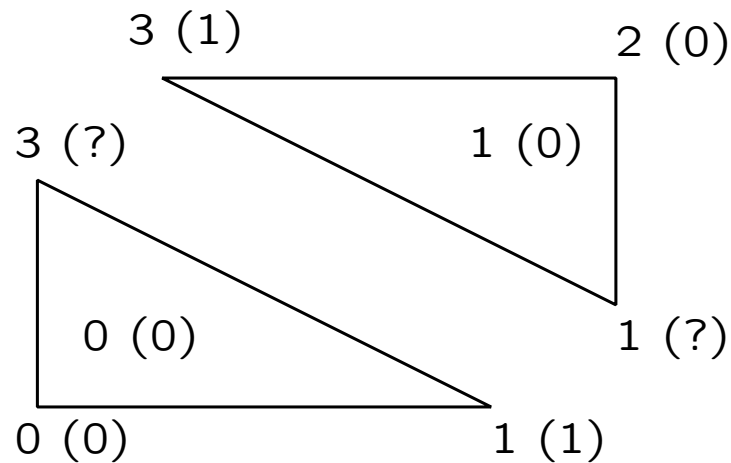
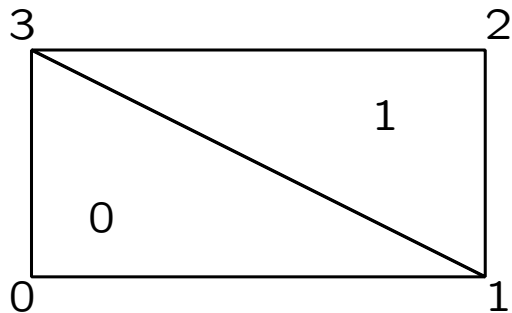
## Conceptual model of distributed functions and relations

- Distributed relations:

$$\begin{array}{l} R = [[0, 1, 3], [1, 2, 3]] \\ oR = [0, 1, 2] \end{array} \rightarrow \begin{array}{l} R_0 = [[0, 1, 3]] \\ oR_0 = [0, 1, 2] \\ R_1 = [[1, 2, 3]] \\ oR_1 = [0, 1, 2] \end{array}$$

## Distributed data and local computations

- P0:  $f_0 = [(0, 0), (2, 0)]$ ,  $R_0 = [[0, 1, 3]]$  and offsets.
- P1:  $f_1 = [(2, 1), (0, 1)]$ ,  $R_1 = [[1, 2, 3]]$  and offsets.



## Local computations and access/exchange tables

Given:

$(R_0 \dots R_{np-1}), (ofs \dots ofs)$  with  $R_p :: A(A(\text{Nat})), ofs :: A(\text{Nat})$

such that:

$is\_offsets(ofs) == true$  and  $max\_elem(R_p(j)) < ofs(np)$

find access and exchange tables

$(at_0 \dots at_{np-1}), (et_0 \dots et_{np-1})$  with  $at_p, et_p :: A(A(\text{Nat}))$

such that:

$\forall (f_0 \dots f_{np-1})$  with  $offsets(\text{map}(\text{size}, (f_0 \dots f_{np-1}))) == ofs,$

the tuple  $(f'_0 \dots f'_{np-1})$  obtained with **complete f** satisfies:

$$f'_p(at_p(j)(k)) == f(R_p(j)(k))$$

## Local computations and access/exchange tables

complete f

$$et'_p = \text{map}(sf'_p, et_p)$$

**where**

$$sf'_p(a) = \text{map}(sf_p, a)$$

$$sf_p(i) = f_p(i - ofs(p))$$

$$f'_p = \text{concat}(f_p, \text{breadth}(\text{exchange}(et'_0 \dots et'_{np-1})(p)))$$

## SCDRC support for distributed RBAs

- SCDRC supports RBA construction, computation of *access* and *exchange* tables and data exchange with suitable primitives:

## SCDRC support for distributed RBAs

- SCDRC supports RBA construction, computation of *access* and *exchange* tables and data exchange with suitable primitives:

```
typedef Triangle_Center<Coordinate_System> H;  
typedef Array<Vector<Real, 3> > F;  
typedef Reg_Rel<3> R;  
H triangle_center;  
RBA<H, F, R> tca(triangle_center, x, vt);
```

## SCDRC support for distributed RBAs

- SCDRC supports RBA construction, computation of *access* and *exchange* tables and data exchange with suitable primitives:

```
typedef Triangle_Center<Coordinate_System> H;  
typedef Array<Vector<Real, 3> > F;  
typedef Reg_Rel<3> R;  
H triangle_center;  
RBA<H, F, R> tca(triangle_center, x, vt);  
init_access_exch_tables(tca);
```

## SCDRC support for distributed RBAs

- SCDRC supports RBA construction, computation of *access* and *exchange* tables and data exchange with suitable primitives:

```
typedef Triangle_Center<Coordinate_System> H;  
typedef Array<Vector<Real, 3> > F;  
typedef Reg_Rel<3> R;  
H triangle_center;  
RBA<H, F, R> tca(triangle_center, x, vt);  
init_access_exch_tables(tca);  
complete_f(tca);
```

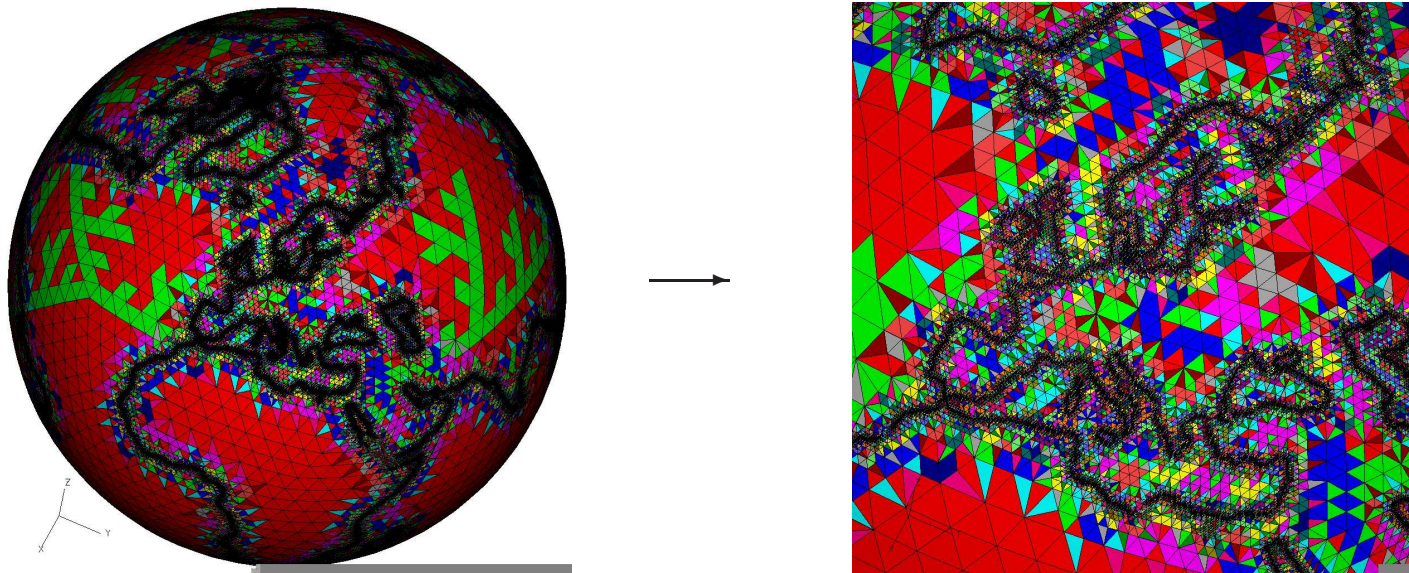
## SCDRC support for distributed RBAs

- Distributed RBAs can be evaluated in parallel as common functions in a sequential computational environment:

```
for(Nat j = 0; j < n_triangles; j++) {  
    const Real area = taa(j);  
    const Vector<Real, 3> center = tca(j);  
    a += area;  
    c += area * center;  
}
```

## Distributed RBCs for partitioning and grid computations

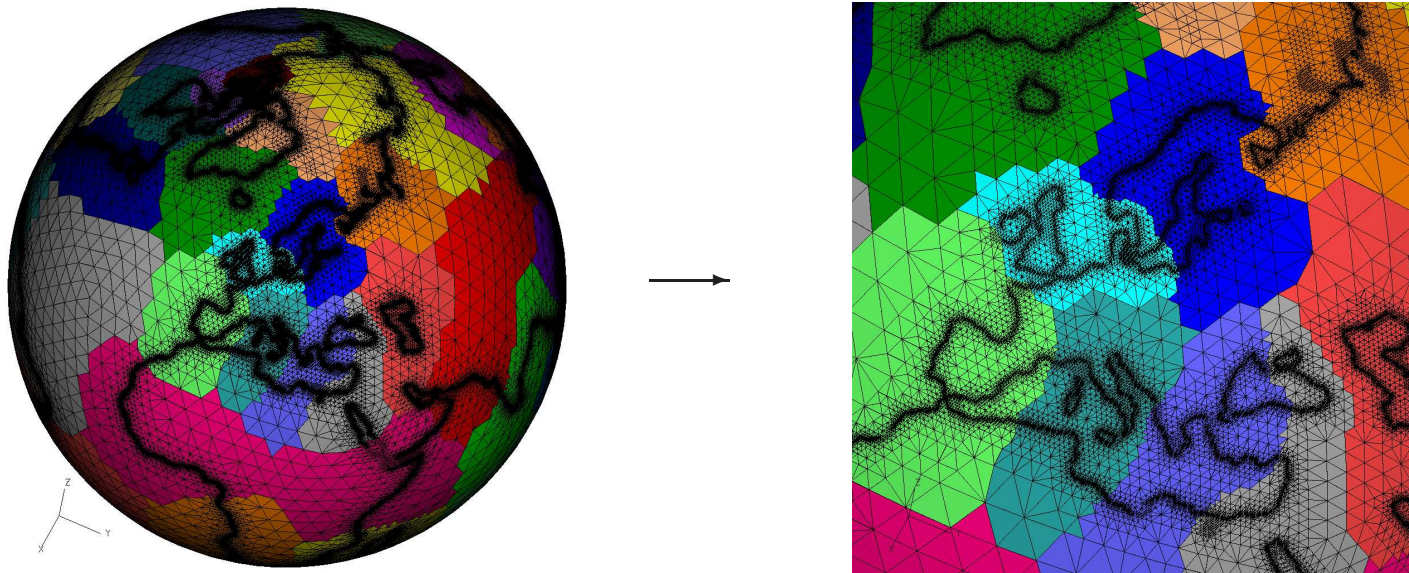
- Trivial partitioning  $\Rightarrow$  poor clustering, high comm. volume !



188966 triangles, 43 partitions: total comm. volume = 338495 units

## Distributed RBCs for partitioning and grid computations

- Partitioning using Metis and SCDRC renumbering algorithms:



188966 triangles, 43 partitions: total comm. volume = 2543 units

## Distributed RBCs for partitioning and grid computations

- To use graph partitioning algorithms, one needs operations to:
  - converse  $R$ .

## Distributed RBCs for partitioning and grid computations

- To use graph partitioning algorithms, one needs operations to:
  - converse  $R$ .
  - compose  $R^\circ$  with  $R$  (neighbor triangle computation).

## Distributed RBCs for partitioning and grid computations

- To use graph partitioning algorithms, one needs operations to:
  - converse  $R$ .
  - compose  $R^\circ$  with  $R$  (neighbor triangle computation).
  - intersect, join, etc.

## SCDRC support for distributed RBCs

- SCDRC supports: *sequential* relational operations ...

```
CRS_Rel tv;  
converse(tv, vt);  
CRS_Rel tvv;  
compose(tvv, tv, vt);  
CRS_Rel r;  
subtract_id(r, tvv);  
  
partition_kway(vtpart, n_cuts, r, n_p());
```

## SCDRC support for distributed RBCs

- ... *local* relational operations in a distributed computational environment:

```
CRS_Rel tv;  
local::converse(tv, vt);  
CRS_Rel tvt;  
local::compose(tvt, tv, vt);  
CRS_Rel r;  
local::subtract_id(r, tvt);  
  
local::partition_kway(vtpart, n_cuts, r, n_p());
```

## SCDRC support for distributed RBCs

- ... and *parallel* relational operations in a distributed computational environment

```
CRS_Rel tv;  
converse(tv, vt, ofs);  
CRS_Rel tvt;  
compose(tvt, tv, vt);  
CRS_Rel r;  
subtract_id(r, tvt);
```

with a suitable set of primitives and with a seamless notation.

## Preliminary results

- A simplistic cost model ...

$$C(np) = C_{comp.}(np) + C_{comm.}(np)$$

$$k_1/np + c_1 * np * pbs$$

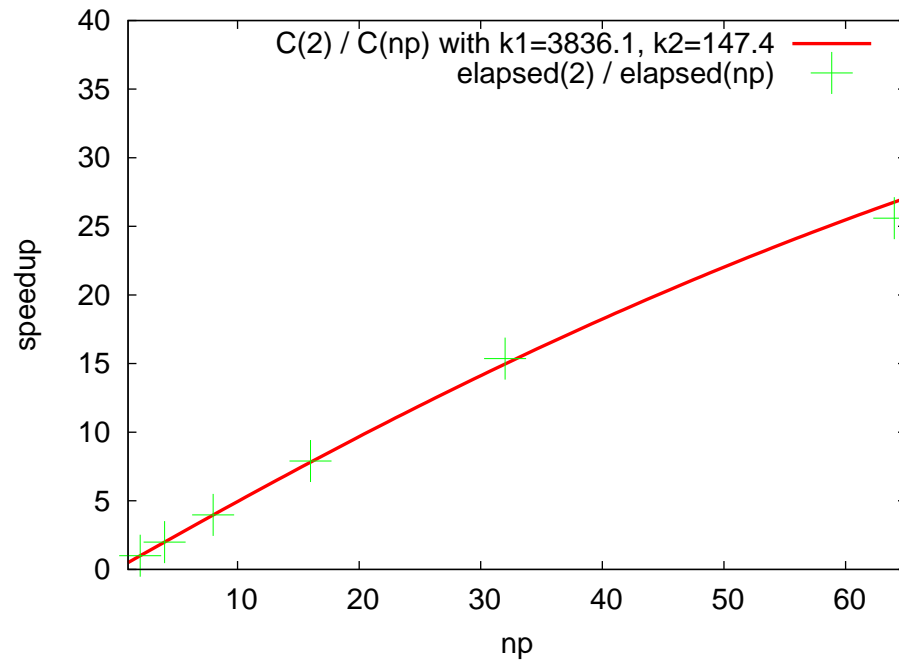
$$k_1/np + c_1 * np * c_2 * (pvs)^{1/2}$$

$$k_1/np + c_1 * np * c_2 * (c_3/np)^{1/2}$$

$$k_1/np + k_2 * np^{1/2}$$

## Preliminary results

- ... explains quite well low comm. frequency speedup results:



## Preliminary results

- For high frequencies, a more realistic model is required:

