Solving linear system

Francesco Sanfilippo

Istituto Nazionale di Fisica Nucleare - Sezione Roma Tre

27 - 31 March 2017

Contents of Day 1

Contents of Day 2

Checking the solution

1. Limits, stability and efficiency of various algorithms
2. Convergence criterions

Accelerating the convergence

1. Mixed precision algorithms
2. Choosing a starting guess
3. Preconditioning the problem

Solving similar problems at the same time

1. Shifted problems $A + \sigma \mathrm{Id}$
2. Deflating the problem

Contents of Day 3

Review of Parallelisation

1. Distributed memory
2. Shared memory
3. Vectors

Gather/scatter approaches

$\rightarrow$ 1+2 different examples of gathering of non-local data

Contents of Day 4

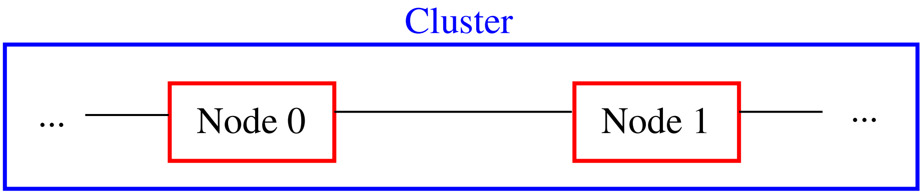
More specifically on parallelisation

1. Communication/computation overlap
2. Multithreading
3. Vectorization

An example of a physical application

$\rightarrow$ Lattice QCD

Lev.1 - Distributed memory paralellism (MPI)



Features

Pro: Using more nodes allow to increase the total computational power

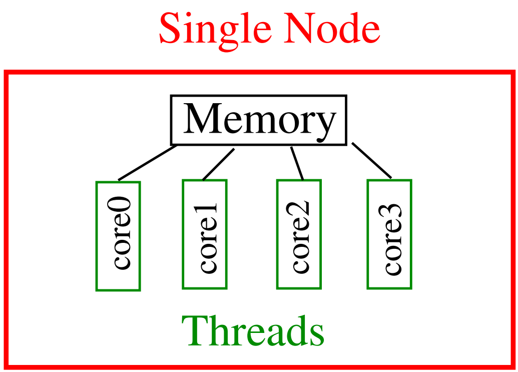
Con: Different **Nodes** do not share memory

Con: Access to non-local data comes with a **cost** (bandwidth, latency...)

Tricks

- Arrange computation in such a way to minimize non-local access
- Overlap computation and communication
- Treat different cores inside a cpu with a different approach...

Lev.2 - Shared memory paralellism (OpenMP)



Features

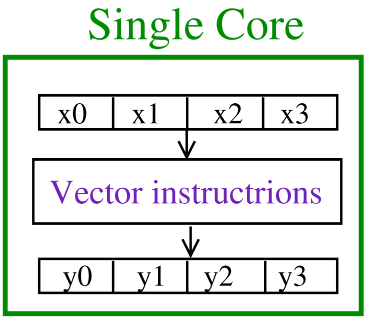
Pro: Different cores on the same CPU can share the same portion of memory, thus avoiding non-local costly access

Con: They can more easily mess-up one with the others trying to access simultaneously to the same data (race condition)

Trick

Split the problem in a way that minimise the need for synchronization barriers/mutex

Lev.3 - Vector paralellism (SIMD instructions)



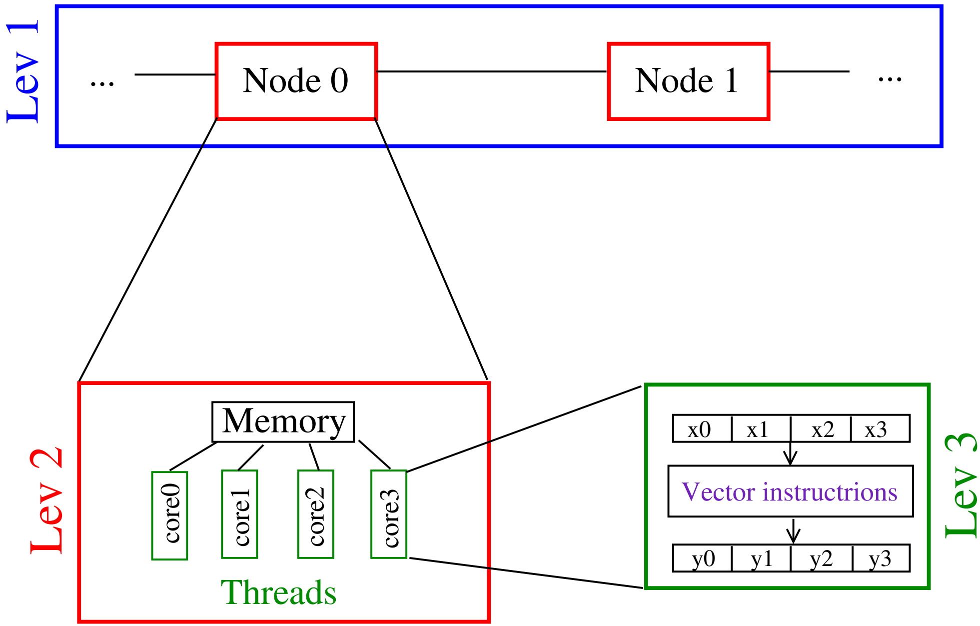
Features

- Pro: Modern cpus include instructions performing many operations at the same time
- Con: Often this restricts to **S**ame **I**nstruction being applied to **M**any **D**ata (SIMD)

Trick

Layout data in such a way to take advantage of vector instructions

Three levels of parallelisation



Target: Achieve a speed-up as close as possible to the **optimal** one

1 dimensional example with 2 nodes

Distribute vectors through different nodes

Node 0

p_0	p_1	p_2	p_3	p_4	$\dots$
$= p^0$					

Node 1

$p_{0'}$	$p_{1'}$	$p_{2'}$	$p_{3'}$	$p_{4'}$	$\dots$
$= p^1$					

Vectors are split in the (direct) sum of the sub-vectors distributed to the two nodes $p = p^0 \oplus p^1$

Distribute the matrix

$$\begin{array}{|c|}\hline t^0 \\ \hline t^1 \\ \hline \end{array} = \begin{array}{|c|c|}\hline A^{00} & A^{01} \\ \hline A^{10} & A^{11} \\ \hline \end{array} \times \begin{array}{|c|}\hline p^0 \\ \hline p^1 \\ \hline \end{array}$$

Problem: Computing b^0 (on node 0) involves also x^1 (which is in node 1)
Where to store **different parts** of A?

1 dimensional example with 2 nodes

Distribute vectors through different nodes

Node 0					
p_0	p_1	p_2	p_3	p_4	$\dots$
$= p^0$					

Node 1					
$p_{0'}$	$p_{1'}$	$p_{2'}$	$p_{3'}$	$p_{4'}$	$\dots$
$= p^1$					

Vectors are split in the (direct) sum of the sub-vectors distributed to the two nodes $p = p^0 \oplus p^1$

Distribute the matrix

$$\begin{array}{|c|}\hline t^0 \\ \hline t^1 \\ \hline\end{array} = \begin{array}{|c|c|}\hline A^{00} & A^{01} \\ \hline A^{10} & A^{11} \\ \hline\end{array} \times \begin{array}{|c|}\hline p^0 \\ \hline p^1 \\ \hline\end{array}$$

Problem: Computing b^0 (on node 0) involves also x^1 (which is in node 1)
Where to store **different parts** of A?

$$\begin{array}{|c|}\hline t^0 \\ \hline t^1 \\ \hline\end{array} = \begin{array}{|c|c|}\hline A^{00} & A^{01} \\ \hline A^{10} & A^{11} \\ \hline\end{array} \times \begin{array}{|c|}\hline p^0 \\ \hline p^1 \\ \hline\end{array}$$

Half of the A must stored in each node: either divide the rows or the columns

Computing $t = Ap$

Local part of the multiplication

- On node 0 we can easily compute $A^{00}p^0$
- On node 1 we can easily compute $A^{11}p^1$

Computing $t = Ap$

Local part of the multiplication

- On node 0 we can easily compute $A^{00}p^{0}$
- On node 1 we can easily compute $A^{11}p^{1}$

What about the non-local part?

Computing $t = Ap$

Local part of the multiplication

- On node 0 we can easily compute $A^{00}p^0$
- On node 1 we can easily compute $A^{11}p^1$

What about the non-local part?

Local computation of non-local part (gather approach)

- Make each node i acquire from other node(s) j the elements needed to compute $A^{ij}p^j$
- Combine the various parts in place:

$$t^i = \underbrace{A^{ii}t^i}_{local} + A^{ij} \underbrace{p^j}_{gathered}$$

Computing $t = Ap$

Local part of the multiplication

- On node 0 we can easily compute $A^{00}p^0$
- On node 1 we can easily compute $A^{11}p^1$

What about the non-local part?

Local computation of non-local part (gather approach)

- Make each node i acquire from other node(s) j the elements needed to compute $A^{ij}p^j$
- Combine the various parts in place:

$$t^i = \underbrace{A^{ii}t^i}_{local} + A^{ij}\underbrace{p^j}_{gathered}$$

Non-Local computation of non-local part (scatter approach)

- Make each node j compute $A^{ij}p^j$ for all other node(s) i
- Combine the various parts after receiving them:

$$t^i = \underbrace{A^{ii}p^i}_{local} + \underbrace{A^{ij}p^j}_{scattered}$$

Not much of difference if the matrix A is <u>dense</u>

Computing $t = Ap$

Local part of the multiplication

- On node 0 we can easily compute $A^{00}p^0$
- On node 1 we can easily compute $A^{11}p^1$

What about the non-local part?

Local computation of non-local part (gather approach)

- Make each node i acquire from other node(s) j the elements needed to compute $A^{ij}p^j$
- Combine the various parts in place:

$$t^i = \underbrace{A^{ii}t^i}_{local} + A^{ij}\underbrace{p^j}_{gathered}$$

Non-Local computation of non-local part (scatter approach)

- Make each node j compute $A^{ij}p^j$ for all other node(s) i
- Combine the various parts after receiving them:

$$t^i = \underbrace{A^{ii}p^i}_{local} + \underbrace{A^{ij}p^j}_{scattered}$$

Not much of difference if the matrix A is <u>dense</u>

What if A is <u>sparse</u>?

Sparse matrix - Laplace equation problem

$$
\begin{pmatrix} t_0^0 \\ t_1^0 \\ t_2^0 \\ \hline t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix}
=
\left(\begin{array}{ccc|ccc}
d & s & 0 & 0 & 0 & s \\
s & d & s & 0 & 0 & 0 \\
\underline{0} & \underline{s} & \underline{d} & \underline{s} & \underline{0} & \underline{0} \\
0 & 0 & s & d & s & 0 \\
0 & 0 & 0 & s & d & s \\
s & 0 & 0 & 0 & s & d
\end{array}\right)
\begin{pmatrix} p_0^0 \\ p_1^0 \\ p_2^0 \\ \hline p_0^1 \\ p_1^1 \\ p_2^1 \end{pmatrix}
$$

Sparse matrix - Laplace equation problem

$$\begin{pmatrix} t_0^0 \\ t_1^0 \\ t_2^0 \\ \hline t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix} = \begin{pmatrix} d & s & 0 & 0 & 0 & s \\ s & d & s & 0 & 0 & 0 \\ 0 & s & d & s & 0 & 0 \\ \hline 0 & 0 & s & d & s & 0 \\ 0 & 0 & 0 & s & d & s \\ s & 0 & 0 & 0 & s & d \end{pmatrix} \begin{pmatrix} p_0^0 \\ p_1^0 \\ p_2^0 \\ \hline p_0^1 \\ p_1^1 \\ p_2^1 \end{pmatrix}$$

$A^{00} p^0$

This part of the multiplication involves only local data

Sparse matrix - Laplace equation problem

$$
\begin{pmatrix} t_0^0 \\ t_1^0 \\ \underline{t_2^0} \\ t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix}
=
\begin{pmatrix}
d & s & 0| & 0 & 0 & \boldsymbol{s} \\
s & d & s| & 0 & 0 & 0 \\
\underline{0} & \underline{s} & \underline{d}| & \underline{\boldsymbol{s}} & \underline{0} & \underline{0} \\
0 & 0 & s| & d & s & 0 \\
0 & 0 & 0| & s & d & s \\
s & 0 & 0| & 0 & s & d
\end{pmatrix}
\begin{pmatrix} p_0^0 \\ p_1^0 \\ \underline{p_2^0} \\ \boldsymbol{p_0^1} \\ p_1^1 \\ \boldsymbol{p_2^1} \end{pmatrix}
$$

$A^{00}p^0$

This part of the multiplication involves only local data

$A^{01}p^1$

- This part of the multiplication involves non-local data
- Not all the entries of p^1 contributes to $A^{01}p^1$!

Sparse matrix - Laplace equation problem

$$
\begin{pmatrix} t_0^0 \\ t_1^0 \\ \underline{t_2^0} \\ t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix} =
\begin{pmatrix}
d & s & 0| & 0 & 0 & \mathbf{s} \\
s & d & s| & 0 & 0 & 0 \\
\underline{0} & \underline{s} & \underline{d}| & \underline{\mathbf{s}} & \underline{0} & \underline{0} \\
0 & 0 & s| & d & s & 0 \\
0 & 0 & 0| & s & d & s \\
s & 0 & 0| & 0 & s & d
\end{pmatrix}
\begin{pmatrix} p_0^0 \\ p_1^0 \\ \underline{p_2^0} \\ \mathbf{p_0^1} \\ p_1^1 \\ \mathbf{p_2^1} \end{pmatrix}
$$

$A^{00} p^0$

This part of the multiplication involves only local data

$A^{01} p^1$

- This part of the multiplication involves non-local data
- Not all the entries of p^1 contributes to $A^{01} p^1$!

Only sp_0^1 and sp_2^1 must be computed

Gather inputs: collect p_0^1 and p_2^1 on node 0 and multiply them by s

or

Scatter outputs: compute sp_0^1 and sp_2^1 on node 1 and send it to node 0

The two approaches are equally good in this case

Sparse matrix - A slightly different problem

$$\begin{pmatrix} t_0^0 \\ t_1^0 \\ \underline{t_2^0} \\ t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix} = \left(\begin{array}{ccc|ccc} d & s & 0 & 0 & \mathbf{s} & \mathbf{s} \\ s & d & s & 0 & 0 & 0 \\ \underline{0} & \underline{s} & \underline{d} & \underline{\mathbf{s}} & \underline{\mathbf{s}} & \underline{0} \\ 0 & 0 & s & d & s & 0 \\ s & 0 & s & s & d & s \\ s & 0 & 0 & 0 & s & d \end{array}\right) \begin{pmatrix} p_0^0 \\ p_1^0 \\ \underline{p_2^0} \\ \mathbf{p_0^1} \\ \mathbf{p_1^1} \\ \mathbf{p_2^1} \end{pmatrix}$$

$A^{00} p^0$

This part of the multiplication involves only local data

$A^{01} p^1$

- This part of the multiplication involves non-local data
- Now all the entries of p^1 contributes to $A^{01} p^1$!

Sparse matrix - A slightly different problem

$$\begin{pmatrix} t_0^0 \\ t_1^0 \\ \underline{t_2^0} \\ t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix} = \begin{pmatrix} d & s & 0| & 0 & s & s \\ s & d & s| & 0 & 0 & 0 \\ \underline{0} & \underline{s} & \underline{d}| & \underline{s} & \underline{s} & \underline{0} \\ 0 & 0 & s| & d & s & 0 \\ s & 0 & s| & s & d & s \\ s & 0 & 0| & 0 & s & d \end{pmatrix} \begin{pmatrix} p_0^0 \\ p_1^0 \\ \underline{p_2^0} \\ \mathbf{p_0^1} \\ \mathbf{p_1^1} \\ \mathbf{p_2^1} \end{pmatrix}$$

$A^{00}p^0$

This part of the multiplication involves only local data

$A^{01}p^1$

- This part of the multiplication involves non-local data
- Now all the entries of p^1 contributes to $A^{01}p^1$!

sp_0^1, sp_1^1 and sp_2^1 must be computed

Gather inputs: collect p_0^1, p_1^1 and p_2^1 on node 0, combine them: $sp_0^1 + sp_1^1$ and $sp_1^1 + sp_2^1$

or

Scatter outputs: compute $sp_0^1 + sp_1^1$ and $sp_1^1 + sp_2^1$ on node 1 and send it to node 0

Sparse matrix - A slightly different problem

$$\begin{pmatrix} t_0^0 \\ t_1^0 \\ \underline{t_2^0} \\ t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix} = \begin{pmatrix} d & s & 0| & 0 & s & s \\ s & d & s| & 0 & 0 & 0 \\ \underline{0} & \underline{s} & \underline{d}| & \underline{s} & \underline{s} & \underline{0} \\ 0 & 0 & s| & d & s & 0 \\ s & 0 & s| & s & d & s \\ s & 0 & 0| & 0 & s & d \end{pmatrix} \begin{pmatrix} p_0^0 \\ p_1^0 \\ \underline{p_2^0} \\ \boldsymbol{p_0^1} \\ \boldsymbol{p_1^1} \\ \boldsymbol{p_2^1} \end{pmatrix}$$

$A^{00}p^0$

This part of the multiplication involves only local data

$A^{01}p^1$

- This part of the multiplication involves non-local data
- Now all the entries of p^1 contributes to $A^{01}p^1$!

sp_0^1, sp_1^1 and sp_2^1 must be computed

Gather inputs: collect p_0^1, p_1^1 and p_2^1 on node 0, combine them: $sp_0^1 + sp_1^1$ and $sp_1^1 + sp_2^1$

or

Scatter outputs: compute $sp_0^1 + sp_1^1$ and $sp_1^1 + sp_2^1$ on node 1 and send it to node 0

The second approach involves less communications (the inverse could happen)

Gather Approach: 0 version

$$
\begin{pmatrix} t_0^0 \\ t_1^0 \\ \underline{t_2^0} \\ t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix} = \left(\begin{array}{ccc|ccc} d & s & 0 & 0 & 0 & s \\ s & d & s & 0 & 0 & 0 \\ \underline{0} & \underline{s} & \underline{d} & \underline{s} & \underline{0} & \underline{0} \\ 0 & 0 & s & d & s & 0 \\ 0 & 0 & 0 & s & d & s \\ s & 0 & 0 & 0 & s & d \end{array}\right) \begin{pmatrix} p_0^0 \\ p_1^0 \\ \underline{p_2^0} \\ p_0^1 \\ p_1^1 \\ p_2^1 \end{pmatrix}
$$

Let's try to keep the approach "as simple as possible"

Minimal modification to the scalar code

- Whenever a component is needed, ask for it to the appropriate node
- When computing t components requiring non-local p, write to the node holding it and ask it to send this piece of info

Gather Approach: 0 version

$$
\begin{pmatrix} t_0^0 \\ t_1^0 \\ t_2^0 \\ \hline t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix}
=
\left(\begin{array}{ccc|ccc}
d & s & 0 & 0 & 0 & s \\
s & d & s & 0 & 0 & 0 \\
0 & s & d & s & 0 & 0 \\ \hline
0 & 0 & s & d & s & 0 \\
0 & 0 & 0 & s & d & s \\
s & 0 & 0 & 0 & s & d
\end{array}\right)
\begin{pmatrix} p_0^0 \\ p_1^0 \\ p_2^0 \\ \hline p_0^1 \\ p_1^1 \\ p_2^1 \end{pmatrix}
$$

Let's try to keep the approach "as simple as possible"

Minimal modification to the scalar code

- Whenever a component is needed, ask for it to the appropriate node
- When computing t components requiring non-local p, write to the node holding it and ask it to send this piece of info

Communication problems

- All nodes have to periodically listen for request
- Problem: **deadlocks**

Gather Version 0 - Avoiding deadlocks

$$
\begin{pmatrix} t_0^0 \\ t_1^0 \\ \underline{t_2^0} \\ t_0^1 \\ t_1^1 \\ t_2^1 \end{pmatrix} = \left(\begin{array}{ccc|ccc} d & s & 0 & 0 & 0 & s \\ s & d & s & 0 & 0 & 0 \\ \underline{0} & \underline{s} & \underline{d} & \underline{s} & \underline{0} & \underline{0} \\ 0 & 0 & s & d & s & 0 \\ 0 & 0 & 0 & s & d & s \\ s & 0 & 0 & 0 & s & d \end{array}\right) \begin{pmatrix} \color{red}{p_0^0} \\ p_1^0 \\ \underline{p_2^0} \\ \color{red}{p_0^1} \\ p_1^1 \\ p_2^1 \end{pmatrix}
$$

Let's try to keep the approach "as simple as possible"

Deadlock

If two nodes are simultaneously asking to each other some non-local data they get **stuck!**

Solution

(Valid for THIS operator that communicate only with first neighbors)

1. Fix parity of nodes (even/odd)
2. Split the application of the matrix in two steps:
   1. in the first, EVEN nodes listen while ODD ones ask
   2. in the second they switch of role

Gather Version 0 - Problems

Gather Version 0 - Problems

Efficiency

- Lot of overhead
- Many branches
- 1/2 nodes are always idle

Scalability

- Difficult to thread
- Difficult to generalize to arbitrary number of nodes

Gather Approach 1 - Shift the vector to apply

Diagonal part

$$
\left(\begin{array}{c} t_0^0 \\ t_1^0 \\ t_2^0 \\ \hline t_0^1 \\ t_1^1 \\ t_2^1 \end{array}\right)
=
\left(\begin{array}{ccc|ccc}
\boldsymbol{d} & s & 0 & 0 & 0 & s \\
s & \boldsymbol{d} & s & 0 & 0 & 0 \\
0 & s & \boldsymbol{d} & s & 0 & 0 \\ \hline
0 & 0 & s & \boldsymbol{d} & s & 0 \\
0 & 0 & 0 & s & \boldsymbol{d} & s \\
s & 0 & 0 & 0 & s & \boldsymbol{d}
\end{array}\right)
\left(\begin{array}{c} p_0^0 \\ p_1^0 \\ p_2^0 \\ \hline p_0^1 \\ p_1^1 \\ p_2^1 \end{array}\right)
$$

Start applying **diagonal part** (local by definition)

SHIFT the vector to apply: $vp' = p_{i+1}$

$$
\left(\begin{array}{c} t_0^0 \\ t_1^0 \\ t_2^0 \\ \hline t_0^1 \\ t_1^1 \\ t_2^1 \end{array}\right)
=
\left(\begin{array}{ccc|ccc}
\boldsymbol{s} & 0 & 0 & 0 & s & d \\
d & \boldsymbol{s} & 0 & 0 & 0 & s \\
s & d & \boldsymbol{s} & 0 & 0 & 0 \\ \hline
0 & s & d & \boldsymbol{s} & 0 & 0 \\
0 & 0 & s & d & \boldsymbol{s} & 0 \\
0 & 0 & 0 & s & d & \boldsymbol{s}
\end{array}\right)
\left(\begin{array}{c} p_1^0 \\ p_2^0 \\ p_0^1 \\ \hline p_1^1 \\ p_2^1 \\ p_0^0 \end{array}\right)
$$

Entries above the matrix are now on the diagonal (therefore all operations are local)

Gather Approach 1 - Shift the vector to apply

Shifting backward (to shift forward do the opposite)

- Create a buffer vector p'
- Copy local data: $p'_i = p_{i+1}$ if $i + 1$ is local
- Communicate backward the first local element and receive the last one from forward node

Gather Approach 1 - Shift the vector to apply

Shifting backward (to shift forward do the opposite)

- Create a buffer vector p'
- Copy local data: $p'_i = p_{i+1}$ if $i+1$ is local
- Communicate backward the first local element and receive the last one from forward node

Avoiding buffer and local copy

- Introduce a shifted local coordinate: $s(i) = (i + \delta) \bmod N_{loc}$
- Every time we shift backward: $\delta = \delta + 1$
- Every time we shift forward: $\delta = \delta - 1$
- We leave local data where it is but we replace
  - $p^n_{s(N_{loc}-1)}$ with $p^{n+1}_{s(0)}$ when we shift backward
  - $p^n_{s(0)}$ with $p^{n-1}_{s(N_{loc}-1)}$ when we shift forward

Gather Approach 1 - Shift the vector to apply

Pro

- Easy to program, no deadlock
- No branch
- No idle node
- Easy to generalize to different operator (do more shifts)
- Communications are all done in a single bunch for each shift (good for regular patterns)
- Consecutive access to memory (only if really shifting the vector)

Con

Requires:

- To take a copy of the source vector (more memory) or
- To undo the shift to come back to original vector (more communications)

Gather Approach 2 - Caching non-local data

$$\begin{pmatrix} p_0^0 & p_1^0 & p_2^0 \end{pmatrix}$$

Initialization

1. Identify all the needed component before applying the matrix
2. Send to each node in advance the list of component that he will have to send
3. Set a buffer for non-local data, keep track of things to receive from/send to other nodes

At each application

1. Collect all data needed from other nodes
2. Simultaneously send to other nodes what they need in turn
3. Apply the matrix

Where to store non-local data?

- We can store non-local data together with local one
- Define the cached buffer at the end/origin of the local vector
- We then can easily point to cache when non-local data is required (and no branch occur!)

$$(0|\; p_0^0 \quad p_1^0 \quad p_2^0 \;|0) \quad \xrightarrow{comm} \quad ({\color{red}p_2^1}|\; p_0^0 \quad p_1^0 \quad p_2^0 \;|{\color{red}p_0^1})$$

Gather Approach 2 - Caching non-local data

Pro

- 1 single block of communication
- no copy involved

Con

- Harder to program (when in multiple dimensions and using cartesian grid)
- Needs more book-keping (for more complicated operators)
- Cached data can take significant amount of memory
- Read from memory in a fragmented way

Modifications to the solver

Very few modifications are needed

At the beginning

- Vectors must be distributed
- Matrix must be distributed or its parameter broadcasted

At each iteration

- Global scalar products must be computed
- Application of the matrix to the vector must me performed according to one of the proposed approach