Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Multicore parallelization of block cyclic reduction algorithm

Dimitri Lecas[a], Richard Chevalier[a],Pascal Joly[b]*

[a]IDRIS, Bat 506 BP 167, 91403 ORSAY CEDEX, FRANCE
[b]Laboratoire Jacques-Louis Lions, 4 place Jussieu, 75252 PARIS CEDEX 5, FRANCE

Abstract

The goal of this work is to evaluate how to parallelize the block cyclic reduction using MPI and OpenMP. This algorithm is used to solve elliptic problems much faster than the traditional iterative methods. We explain the parallelism that we exhibit and show the performance of the MPI and the OpenMP version that we have made.

"Project ID: give here the project ID"

1. Block cyclic reduction algorithm

The goal of this work is to evaluate how to parallelize the block cyclic reduction using MPI and OpenMP. This algorithm is used to solve elliptic problems much faster than the traditional iterative methods.

1.1. The cyclic reduction

For example the finite difference of poisson equation is:

$$(P) \Leftrightarrow \begin{cases} Find \quad u \in V \quad such\ as \\ -\Delta u = f \ \ in\ \Omega = \ ]0,1[\times]0,1[\\ u_{|\partial\Omega} = g \end{cases}$$

$$(P) \Leftrightarrow AX = Y \Leftrightarrow \begin{pmatrix} B & T & & & \\ T & B & T & & \\ & T & \ddots & \ddots & \\ & & \ddots & B & T \\ & & & T & B \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{m_y} \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_{m_y} \end{pmatrix}$$

With:

$$\begin{cases} A \in \mathbb{R}^{n\times n} \\ X \in \mathbb{R}^{n} \\ Y \in \mathbb{R}^{n} \end{cases} \quad and \quad \begin{cases} B \in \mathbb{R}^{m_x\times m_x} \\ T \in \mathbb{R}^{m_x\times m_x} \\ U_i \in \mathbb{R}^{m_x} \\ F_i \in \mathbb{R}^{m_x} \end{cases}$$

$$B = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & 4 & -1 \\ & & & -1 & 4 \end{pmatrix} \quad and \quad T = \begin{pmatrix} -1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & -1 & \\ & & & & -1 \end{pmatrix}$$

$$U_j = \begin{pmatrix} u_{1j} \\ u_{2j} \\ u_{3j} \\ \vdots \\ u_{m_x j} \end{pmatrix} \quad and \quad F_j = h^2 \begin{pmatrix} f_{1j} \\ f_{2j} \\ f_{3j} \\ \vdots \\ f_{m_x j} \end{pmatrix}$$

The concept of block cyclic reduction is to iteratively eliminate half of the unknowns until there is an only single block system which can be solved directly.

So we have for j such as: $1 < j < 2^{j_q} - 1$:

$$\begin{aligned} TU_{2*j-2} + BU_{2*j-1} + TU_{2*j} &= F_{2*j-1} \\ TU_{2*j-1} + BU_{2*j} + TU_{2*j+1} &= F_{2*j} \\ TU_{2*j} + BU_{2*j+1} + TU_{2*j+2} &= F_{2*j+1} \end{aligned}$$

If we multiply the first and third lines by T and the second line by $-B$, then sum this three new lines, if $TB = BT$ we eliminate the odd unknowns U_{2*j-1}.

$$T^2U_{2*j-2} + (2T^2 - B^2)U_{2*j} + T^2U_{2*j+2} = TF_{2*j-1} - BF_{2*j} + TF_{2*j+1}$$

We have then the same structure for this new linear system with half of the unknowns:

$$\begin{cases} T^{(1)} = T^2 \\ B^{(1)} = (2T^2 - B^2) \\ F^{(1)}_{2*j} = TF_{2*j-1} - BF_{2*j} + TF_{2*j+1} \end{cases}$$

$$\begin{pmatrix} B^{(1)} & T^{(1)} & & & \\ T^{(1)} & B^{(1)} & T^{(1)} & & \\ & T^{(1)} & \ddots & \ddots & \\ & & \ddots & B^{(1)} & T^{(1)} \\ & & & T^{(1)} & B^{(1)} \end{pmatrix} \begin{pmatrix} U_2 \\ U_4 \\ U_6 \\ \vdots \\ U_{m_y-1} \end{pmatrix} = \begin{pmatrix} F^{(1)}_2 \\ F^{(1)}_4 \\ F^{(1)}_6 \\ \vdots \\ F^{(1)}_{m_y-1} \end{pmatrix}$$

If we continue after $k$iterations we have:

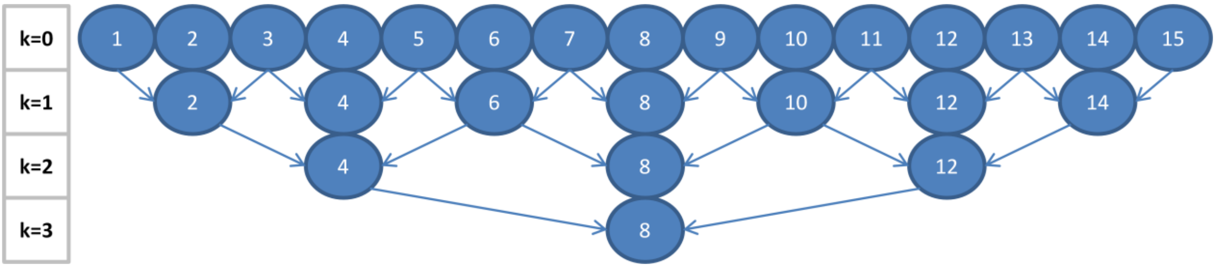
$$\begin{cases} T^{(k)} = \left[T^{(k-1)}\right]^2 \\ B^{(k)} = \left(2\left[T^{(k-1)}\right]^2 - \left[B^{(k-1)}\right]^2\right) \\ F^{(k)}_{2^k*j} = T^{(k-1)}F^{(k-1)}_{2^k*j-2^{k-1}} - B^{(k-1)}F^{(k-1)}_{2^k*j} + T^{(k-1)}F^{(k-1)}_{2^k*j+2^{k-1}} \end{cases}$$

$$\begin{pmatrix} B^{(k)} & T^{(k)} & & & \\ T^{(k)} & B^{(k)} & T^{(k)} & & \\ & T^{(k)} & \ddots & \ddots & \\ & & \ddots & B^{(k)} & T^{(k)} \\ & & & T^{(k)} & B^{(k)} \end{pmatrix} \begin{pmatrix} U_{2^k} \\ U_{2^k*2} \\ U_{2^k*3} \\ \vdots \\ U_{2^k*\left(2^{jq-k}-1\right)} \end{pmatrix} = \begin{pmatrix} F^{(k)}_2 \\ F^{(k)}_4 \\ F^{(k)}_6 \\ \vdots \\ F^{(k)}_{m_y-1} \end{pmatrix}$$

We continue until $\mathrm{k} = \mathrm{j_q} - 1$ ($m_y = 2^{j_q} - 1$ and $m_x = 2^{j_p} - 1$), then we have only one block equation in the system.

$$B^{(j_q-1)} U_{2^{j_q-1}} = F^{(j_q-1)}_{2^{j_q-1}}$$

For example with $j_q = 4$, $m_y = 2^4 - 1 = 15$, we have 15 unknowns to compute. The elimination scheme is the follow :



1.2. Backward substitution

After solving the only block equation, we compute the « odd » values with the even values that we have computed in the previous step. So after this resolution:

$$B^{(j_q-1)} U_{2^{j_q-1}} = F^{(j_q-1)}_{2^{j_q-1}}$$

We can proceed to the next step with a backward substitution:

$$\begin{cases} B^{(j_q-2)} U_{2^{j_q-1}-2^{j_q-2}} = F^{(j_q-2)}_{2^{j_q-1}-2^{j_q-2}} - T^{(j_q-2)} U_{2^{j_q-1}} \\ B^{(j_q-2)} U_{2^{j_q-1}+2^{j_q-2}} = F^{(j_q-2)}_{2^{j_q-1}+2^{j_q-2}} - T^{(j_q-2)} U_{2^{j_q-1}} \end{cases}$$

After computation of $U_{2^{j_q-1}-2^{j_q-2}}$ et $U_{2^{j_q-1}+2^{j_q-2}}$. We do the step $k = j_q - 3$. We compute for $j = 1, \dots, 2^{j_q-(j_q-2)}$ values $U_{2^{j_q-2}*j-2^{j_q-3}}$:

- For $j = 1$:

$$B^{(j_q-3)} U_{2^{j_q-3}} = F^{(j_q-3)}_{2^{j_q-3}} - T^{(j_q-3)} U_{2^{j_q-2}}$$

- For $j = 2, \dots, 2^{j_q-(j_q-2)} - 1$

$$B^{(j_q-3)} U_{2^{j_q-2}*j-2^{j_q-3}} = F^{(j_q-3)}_{2^{j_q-2}*j-2^{j_q-3}} - T^{(j_q-3)} \left(U_{2^{j_q-2}*j} + U_{2^{j_q-2}*(j-1)} \right)$$

- For $j = 2^{j_q-(j_q-2)}$

$$B^{(j_q-3)} U_{2^{j_q}-2^{j_q-3}} = F^{(j_q-3)}_{2^{j_q}-2^{j_q-3}} - T^{(j_q-3)} U_{2^{j_q}-2^{j_q-2}}$$

And so on. At step k, we have:

- For $j = 1$:

$$B^{(k-1)} U_{2^k-2^{k-1}} = F^{(k-1)}_{2^k-2^{k-1}} - T^{(k-1)} U_{2^k}$$

- For $j = 2, \dots, 2^{j_q-k} - 1$

$$B^{(k-1)} U_{2^k*j-2^{k-1}} = F^{(k-1)}_{2^k*j-2^{k-1}} - T^{(k-1)} \left(U_{2^k*j} + U_{2^k*(j-1)} \right)$$

- For $j = 2^{j_q-k}$

$$B^{(k-1)} U_{2^{j_q}-2^{k-1}} = F^{(k-1)}_{2^{j_q}-2^{k-1}} - T^{(k-1)} U_{2^{j_q}-2^k}$$

Then for the step $k = 1$, we compute the last values :

- For $j = 1$:

$$B^{(0)}U_1 = F_1^{(0)} - T^{(0)}U_2$$

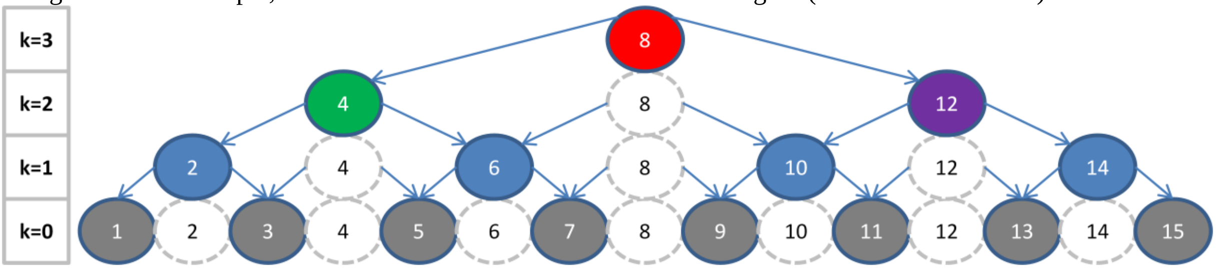
- For $j = 2, \dots, 2^{j_q-1} - 1$

$$B^{(0)}U_{2*j-1} = F_{2*j-1}^{(0)} - T^{(0)}\left(U_{2*j} + U_{2*j-2}\right)$$

- For $j = m_y = 2^{j_q-1}$

$$B^{(0)}U_{2^{j_q-1}} = F_{2^{j_q-1}}^{(0)} - T^{(0)}U_{2^{j_q-1}-1}$$

Again for the example, the backward substitution is shown in this figure (with the same colors) :



We need now to explain how to compute $B^{(k)}$, $T^{(k)}$ and $F_{2^k*j}^{(k)}$. It can be proved that :

$$T^{(k)} = T^{2^k}$$

$$B^{(k)} = -\prod_{l=1}^{2^k}(\mathrm{B} - 2\cos\,(\theta_{kl})T)$$

$$\text{with } \theta_{kl} = \left(l - \tfrac{1}{2}\right)\pi/2^k \quad \text{For } l = 1,2, \dots, 2^k$$

This can be also written in a very interesting way:

$$\left[\,B^{(k)}\right]^{-1} = -\sum_{l=1}^{2^k}\alpha_{kl}[\mathrm{B} - 2\cos\,(\theta_{kl})T]^{-1}$$

$$\text{With } \alpha_{kl} = \frac{(-1)^l}{2^k}\sin\,(\theta_{kl})$$

We have also :

$$F_{2^k*j}^{(k)} = T^{(k-1)}F_{2^k*j-2^{k-1}}^{(k-1)} - B^{(k-1)}F_{2^k*j}^{(k-1)} + T^{(k-1)}F_{2^k*j+2^{k-1}}^{(k-1)}$$

The trouble is that this formulation cannot be use because of precision instability [1].

1.3. Buneman's algorithm

We choose the Buneman's variant who give numerically stable results [1]. This algorithm introduces two series P and Q :

- For $k = 0$, For $j = 1,2, \dots, 2^{j_q} - 1$:

$$P_j^{(0)} = 0$$

$$Q_j^{(0)} = F_j^{(0)} = F_j$$

- For $0 < k < j_q$, For $j = 1,2, \dots, 2^{j_q-k} - 1$:

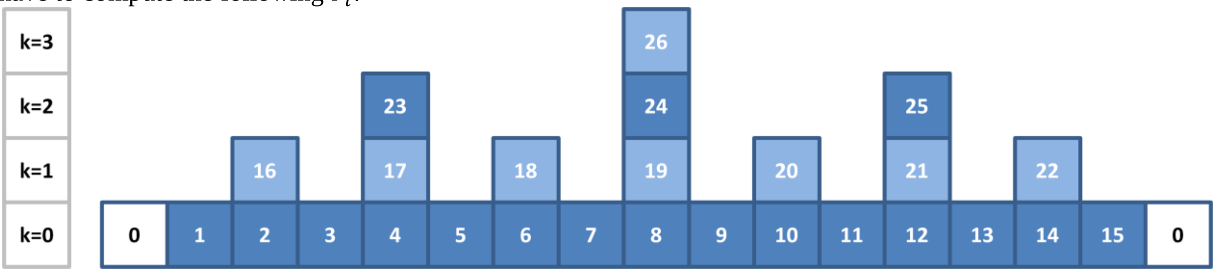
$$P_{2^k*j}^{(k)} = P_{2^k*j}^{(k-1)} - \left[B^{(k-1)}\right]^{-1}\left[T^{(k-1)}\left(P_{2^k*j-2^{k-1}}^{(k-1)} + P_{2^k*j+2^{k-1}}^{(k-1)}\right) - Q_{2^k*j}^{(k-1)}\right]$$

$$Q_{2^k*j}^{(k)} = T^{(k-1)}\left(Q_{2^k*j-2^{k-1}}^{(k-1)} + Q_{2^k*j+2^{k-1}}^{(k-1)} - 2T^{(k-1)}P_{2^k*j}^{(k)}\right)$$

Then we have for $k = 1,2, \dots, j_q - 1$ and $j = 1,2, \dots, 2^{j_q-k} - 1$:

$$F_{2^k*j}^{(k)} = B^{(k)}P_{2^k*j}^{(k)} + Q_{2^k*j}^{(k)}$$

Before the computation of unknowns, we need to compute Buneman's series P and Q. For example for $j_q = 4$ we have to compute the following P_i:



1.4. The algorithm

In summary, the Buneman's variant of block cyclic reduction takes the following form :

1. Computation of Buneman's series

For $j = 1, \dots, 2^{j_q} - 1$

$P_j^{(0)} = 0$ and $Q_j^{(0)} = F_j^{(0)} = F_j$

End

For $k = 1, \dots, j_q - 1$

For $j = 1, \dots, 2^{j_q-k} - 1$

$$\boldsymbol{P}_{2^k*j}^{(k)} = \boldsymbol{P}_{2^k*j}^{(k-1)} - \left[\boldsymbol{B}^{(k-1)}\right]^{-1}\left[\boldsymbol{T}^{(k-1)}\left(\boldsymbol{P}_{2^k*j-2^{k-1}}^{(k-1)} + \boldsymbol{P}_{2^k*j+2^{k-1}}^{(k-1)}\right) - \boldsymbol{Q}_{2^k*j}^{(k-1)}\right]$$

$$\boldsymbol{Q}_{2^k*j}^{(k)} = \boldsymbol{T}^{(k-1)}\left(\boldsymbol{Q}_{2^k*j-2^{k-1}}^{(k-1)} + \boldsymbol{Q}_{2^k*j+2^{k-1}}^{(k-1)}\right) - \boldsymbol{2T}^{(k-1)}\boldsymbol{P}_{2^k*j}^{(k)}$$

End

End

2. Solve the single block equation

For $k = j_q - 1$

$$\boldsymbol{U}_{2^{j_q}-2^{j_q-1}}^{(j_q-1)} = \left[\boldsymbol{B}^{(j_q-1)}\right]^{-1}\boldsymbol{Q}_{2^{j_q}-2^{j_q-1}}^{(j_q-1)} + \boldsymbol{P}_{2^{j_q}-2^{j_q-1}}^{(j_q-1)}$$

3. Backward substitution

For $k = j_q - 1, \dots, 1$

For $j = 2, \dots, 2^{j_q - k} - 1$

$$U^{(k-1)}_{2^k * j - 2^{k-1}} = \left[B^{(k-1)}\right]^{-1} \left(Q^{(k-1)}_{2^k * j - 2^{k-1}} - T^{(k-1)} \left(U^{(k)}_{2^k * j} + U^{(k)}_{2^k * j - 2^k} \right) \right) - P^{(k-1)}_{2^k * j - 2^{k-1}}$$

End

For $j = 1$

$$U^{(k-1)}_{2^k - 2^{k-1}} = \left[B^{(k-1)}\right]^{-1} \left(Q^{(k-1)}_{2^k - 2^{k-1}} - T^{(k-1)} U^{(k)}_{2^k} \right) - P^{(k-1)}_{2^k - 2^{k-1}}$$

For $j = 2^{j_q - k}$

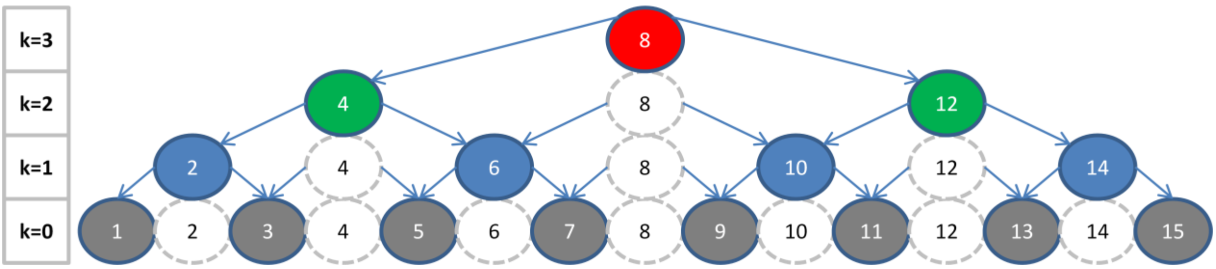
$$U^{(k-1)}_{2^{j_q} - 2^{k-1}} = \left[B^{(k-1)}\right]^{-1} \left(Q^{(k-1)}_{2^{j_q} - 2^{k-1}} - T^{(k-1)} U^{(k)}_{2^{j_q} - 2^k} \right) - P^{(k-1)}_{2^{j_q} - 2^{k-1}}$$

End

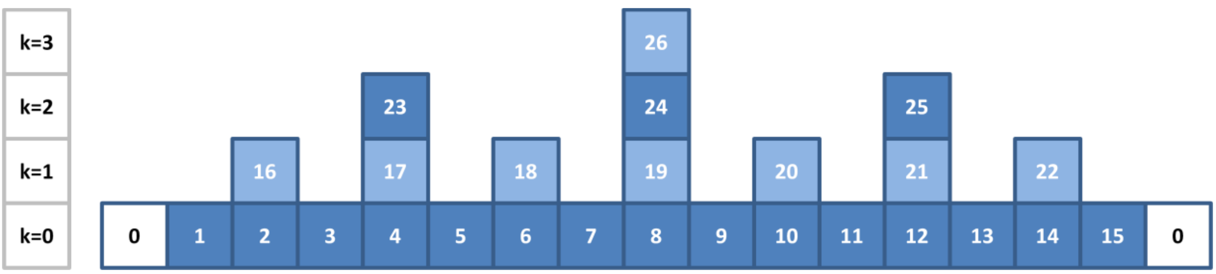
2. Parallelization of the algorithm

2.1. Two levels of parallelization

In the backward part, we can compute in parallel the unknown solution in the same k-level. For each k-level there are $2^{(j_q-1)-k}$ unknowns that can be computed independently. For example with $j_q = 4$:



At level $k = 0$, unknowns $u_1, u_3, u_5, u_7, u_9, u_{11}, u_{13}$ and u_{15} can be computed in parallel, idem for unknowns u_2, u_6, u_{10} and u_{14}. In buneman's part, we can also compute P and Q in the same k-level in parallel. For $j_q = 4$, $P_{16}, P_{17}, P_{18}, P_{19}, P_{20}, P_{21}$ *and* P_{22} can be computed independently:



We can exhibit parallelism in « $B^{(r-1)}X = Y$ ». This computation is used a lot in the algorithm.

$$B^{(r-1)}X = Y \quad \Leftrightarrow \quad X = \left[B^{(r-1)}\right]^{-1} Y$$

$$\left[\boldsymbol{B}^{(r)}\right]^{-1} = -\sum_{l=1}^{2^r} \alpha_{rl}[\mathbf{B} - 2\cos(\theta_{rl})T]^{-1}$$

$$X = -\sum_{l=1}^{2^r} \alpha_{rl}[\mathbf{B} - 2\cos(\theta_{rl})T]^{-1}\, Y$$

We can also write:

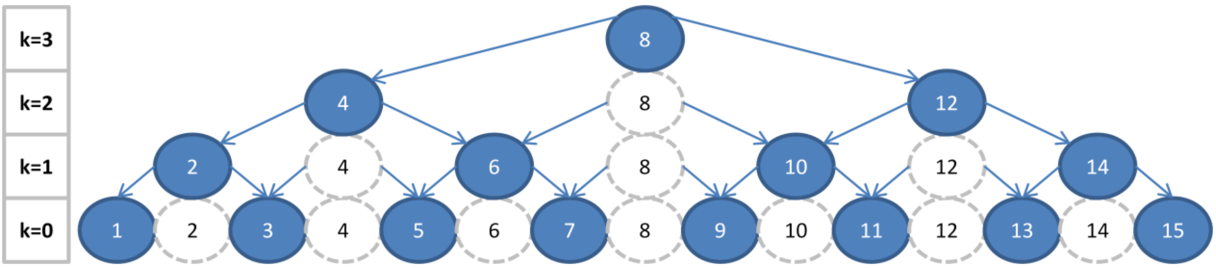
$$X = -\sum_{l=1}^{2^r} \alpha_{rl} X_l$$

$$[\mathbf{B} - 2\cos(\theta_{rl})T]X_l = Y$$

The « $\alpha_{rl}X_l$ » can be computed independently, so we can distribute the computation of X_l.

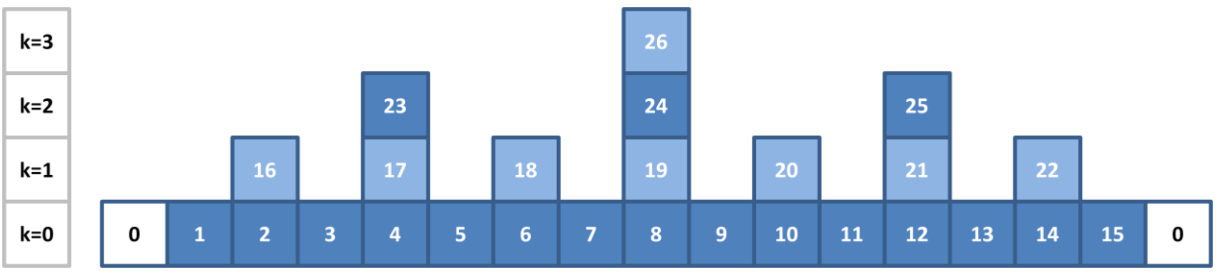
2.2. Dependencies

There are dependencies between unknowns u_i of level k and level $k-1$. This is shown in this graph for $j_q = 4$:

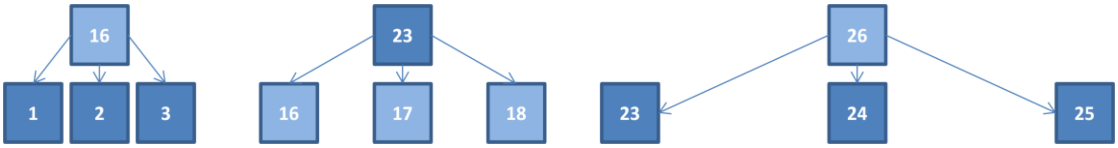


To compute the unknown u_5 we need to known u_4 and u_6.

For buneman's part, there are also dependencies between P_i of level k and level k-1. This is shown in this graph:

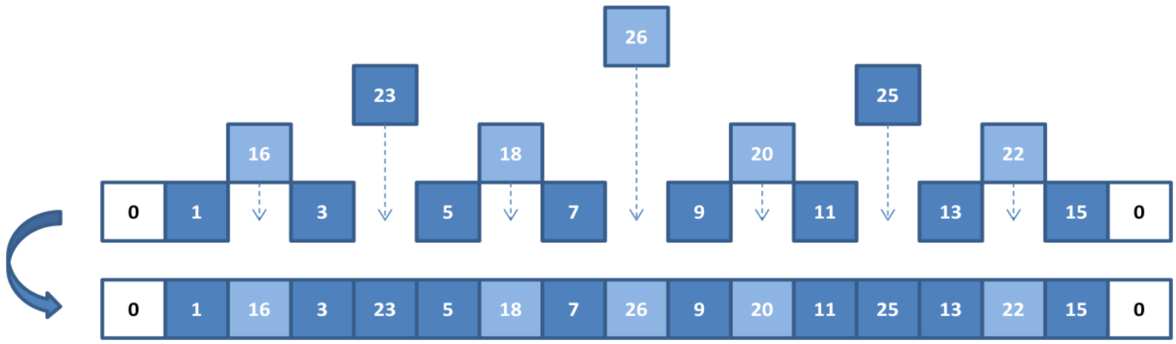


To compute P_{16} we need to known $P_1, P_2 and P_3$, for P_{23}we need $P_{16}, P_{17} and P_{18}$. To summarize, at each level k, to compute a P_i we need to known the 3 P_i on level k-1 that are just below.

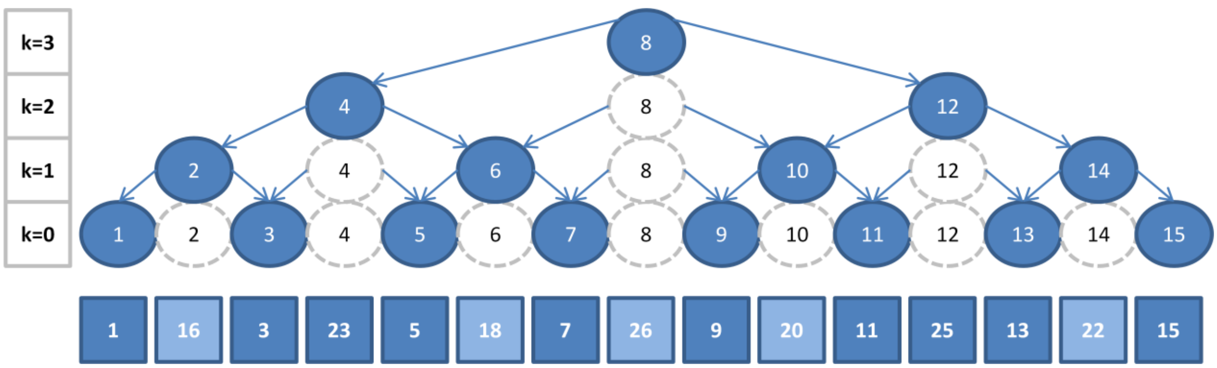


Not all the P_i computed are used in the backward substitution, in fact we need to known only one P_i for the unknown u_i. The list of $P_i$needed in the example is:

This list can be obtained by keeping only the top element of each tower:



To summarize the association between unknown and buneman’s term is the following :

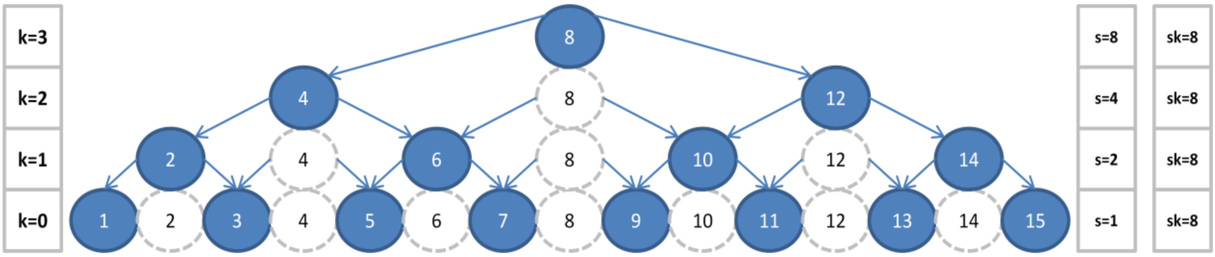


2.3. Number of resolution

The number of resolution (computation of $\boldsymbol{X_l}$) is different between the two part

Backward substitution

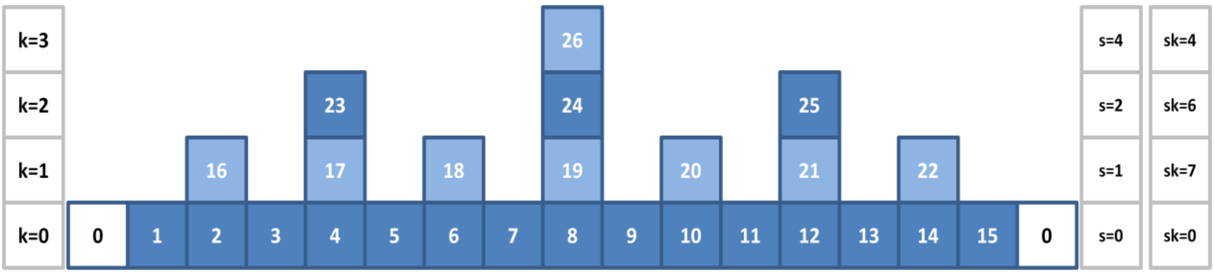
At level k, there is $2^{(j_q-1)-k}$ unknown and for each unknown there is 2^k $\boldsymbol{X_l}$ to compute. So for each level k there is $2^{(j_q-1)-k} \times 2^k = 2^{(j_q-1)}$ $\boldsymbol{X_l}$ to compute.



« k » level in bacward substitution
« sk » total number of resolution for a level k
« s » number of resolution for an unknown

Buneman's part

At level k, there is $2^{j_q-k} - 1$ P_i and for each P_i there is 2^{k-1} $\boldsymbol{X_l}$ to compute. So for each level k there is $\left(2^{j_q-k} - 1\right) \times 2^{k-1} = 2^{(j_q-1)} - 2^{k-1}$ $\boldsymbol{X_l}$ to compute.

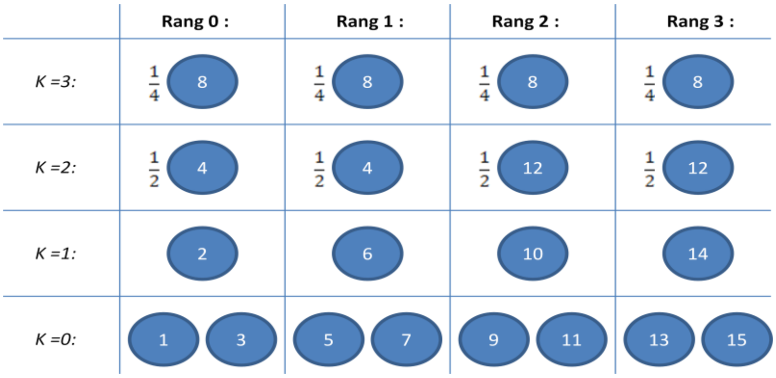


We can see that the number of resolution for buneman is dependent on k but is less than the number of resolution for the backward substitution.

.

2.4. Distribution of computations

This is an example how we do the distribution of $\boldsymbol{X_l}$ using 4 processors:



- For k=3, each processor compute ¼ of $\boldsymbol{X_l}$ for unknown u_8. There is then a reduction to assemble results.
- For k=2, there are two groups, the first group (rank 0 and 1) compute ½ of $\boldsymbol{X_l}$ for u_4, the second one compute ½ of $\boldsymbol{X_l}$ for u_{12}. There is then a reduction in each group.

3. OpenMP

3.1. Data sharing

The vector of unknown u_i and the buneman's series P_i *and* Q_i are shared.

```
!$OMP PARALLELE DEFAULT (NONE) &
!$OMP SHARED (u, pbu, qbu, … ) &
!$OMP PRIVATE (i, j, ja, jb, tmp1, tmp2, … ) &
```

3.2. Worksharing

Since OpenMP doesn't support the concept of group (like communicator in MPI), the distribution cannot be done using the worksharing constructs (OMP DO). So the distribution is done like in MPI using the rank and the number of total processor, and computing the bound of the loop j (loop on unknown or buneman's term) and of the loop l (loop on the resolution)

```
…
rank = OMP_GET_THREAD_NUM()
…
if (nb_thread_by_node == 0) then
!Cas 1 : one thread compute several node (ja/=jb)
nb_node_by_thread =((nb_node/nb_processor+1)/2)*2
jb=(rank+1)* nb_node_by_thread
ja=rank* nb_node_by_thread +1
else
!Cas 2 : Group of threads compute a node (ja=jb)
jb = (rank/nb_thread_by_node)+1
ja = jb
endif
…
```

```
…
! Section 1:
! "kth" rank in the group of threads
kth = modulo(rank,nbth)

! Section 2:
! Number of resolution by thread
kmod = nb_system/nbth
! computation of l-range
l_min = kth*kmod +1
l_max = (kth+1)*kmod

…
! Section 3:
! Resolution
do l = l_min, l_max
…
call CHOLESKY(…)
…
enddo

…
```

- « $nbth$ » is the number of threads, nbth = nb_thread_by_node
- « $rank$ » is the rank
- « $nb\_systeme$ » is the number of resolution for a node.
- « $CHOLESKY(\ldots)$ » is the fonction that compute one $\boldsymbol{X_l}$.

For the reduction part, again we cannot use the CRITICAL directive, since we want to do reduction inside a group of threads, so we use the LOCK routines. Inside a group, threads share the lock, to avoid race condition in updating the unknown.

```
…
! Allocation of vector of lock
ALLOCATE ( tab_lock_group(nb_proc) )
…
!$OMP PARALLEL
…
! Init of the lock
!$OMP DO
do I = 1, nb_proc
    call OMP_init_lock( tab_lock_group(i) )
enddo
!$OMP END DO
! ----------------------------------------------------------------------------------------
…
! ---------- Reduction ------------------------------------------
call OMP_set_lock( tab_lock_group(group_number) )
          u(:,jk2)=u(:,jk2) – tmp1(:)
```

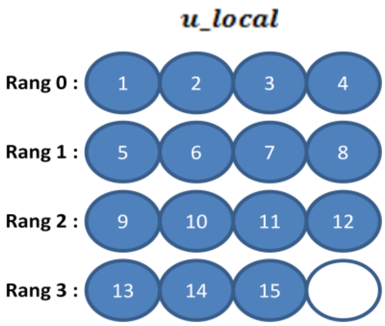
```
call OMP_unset_lock( tab_lock_group(group_number) )
! ---------------------------------------------------------------------------------
…
! ----------- Destroy ------------------------------------
do i = 1, nb_proc
call OMP_destroy_lock( tab_lock_group(i) )
enddo
! ----------------------------------------------------------------------------------
…
!$OMP END PARALLEL
```

4. MPI version

Unlike OpenMP, we have in MPI the concept of group with the MPI communicator. The distribution of computation is the same as the OpenMP version. The main problem for the MPI version is to balance the memory print between processor.

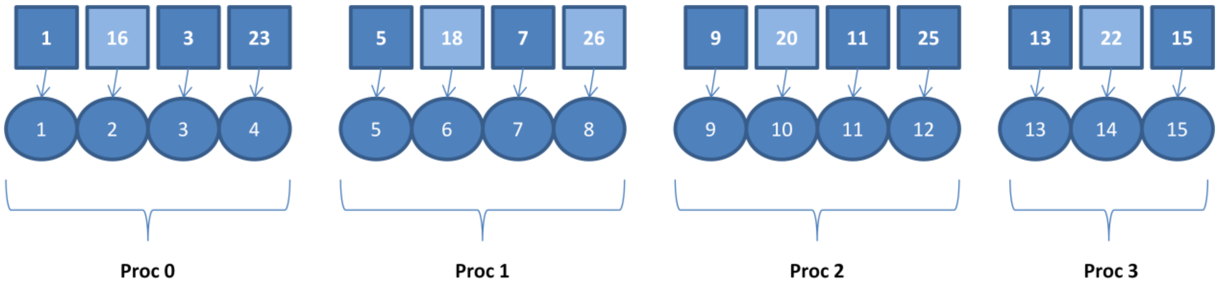
4.1. Memory distribution

The distribution of unknown is simple, the vector is decomposed as many blocks as there are processor. For example with 4 processors we have:

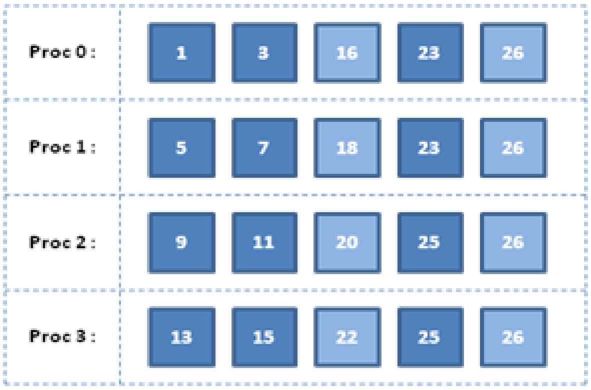


- Rank-0 processor manage unknown 1 to 4
- Rank-1 processor manage unknown 5 to 8
- Rank-2 processor manage unknown 9 to 12
- Rank-3 processor manage unknown 13 to 15

For buneman's series, it is more complicated, we need the P_i *and* Q_i associated we the unknown that the processor manages, for example:



But that is not enough, since a processor contributes to determining other unknowns than the one that he manage,, for example processor rank-0 help to compute the unknown u_8, so he used the P_{26} *and* Q_{26}. So we choose to keep all the P_i *and* Q_i that we need for the backward substitution. For example, again with 4 processors:



4.2. Worksharing

We use the MPI communicator to manage the different group of processor, and the reduction is done via the collective call $MPI\_ALLREDUCE$.

5. Performance

Vargas is an IBM Power 6 composed of 112 SMP nodes p575 IH with 32 cores Power 6 per node.

5.1. OpenMP version

Here is the time taken for the OpenMP version, the scalability is good for 16 threads. The scalability is limited by the scalability of Buneman's part, this section is more difficult to load balance.

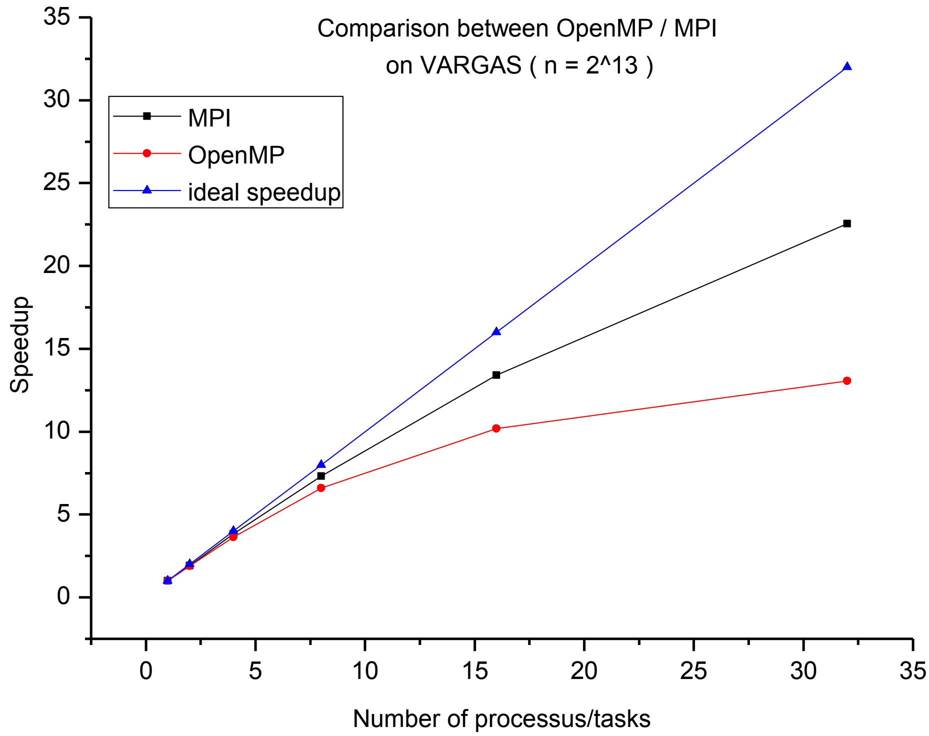
	Time for OpenMP version on VARGAS					
$j_q = j_p$	12			13		
Compilation flags	–qsmp=omp –O2			–qsmp=omp –O2		
Sequential time	15 sec			59 sec		
Nb threads	Buneman Time (s)	Resolution Time (s)	Total time (s)	Buneman Time (s)	Resolution Time(s)	Total time (s)
1	7.090	6.889	13.979	28.89	29.22	58.11
2	3.990	3.500	7.490	15.95	14.72	30.67
4	2 .130	1.830	3.960	8.481	7.539	16.02
8	1.241	1.009	2.250	4.800	4.010	8.810
16	0.860	0.709	1.570	3.360	2.339	5.699
32	0.799	0.600	1.399	2.849	1.600	4.449

5.2. MPI version

Here is the time taken with the MPI version:

MPI version on VARGAS									
$j_q = j_p$	12			13			17		
Compilation flags	–qsmp=omp –O2			–qsmp=omp –O2			–qsmp=omp –O2		
Sequential time	15 sec			63 sec			Not enough memory		
Nb Processor	Buneman's Time (s)	Resolution time (s)	Total time (s)	Buneman's Time (s)	Resolution time (s)	Total time (s)	Buneman's time (s)	Resolution time (s)	Total time (s)
1	7.250	6.780	14.03	30.47	29.49	59.959	...	...	...
2	4.010	3.390	7.400	16.67	14 .77	31.42	...	...	...
4	2.019	1.710	3.710	8.229	7.429	15.66	...	...	...
8	1.129	0.879	1.990	4.440	3.779	8.199	...	...	...
16	0.680	0.460	1.129	2.529	1.950	4.469	...	...	...
32	0.460	0.250	0.709	1.610	1.049	2.660	...	...	...
64	0.349	0.159	0.509	1.149	0.600	1.740	...	...	...
128	0.310	0.100	0.409	0.930	0.389	1.320	...	...	...
256	0.289	0.078	0.370	0.870	0.289	1.149	459	135	594
512	0.244	0.056	0.300	0.720	0.360	1.080	440	117	556
1024	0.289	0.070	0.360	0.690	0.189	0.879	432	110	540

5.3. Speedup comparison



6. Conclusion and outlook

In conclusion, it’s possible to parallelize the buneman’s variant of the block cyclic reduction, but managing the load balance is not easy in buneman’s series computation.

There is some evolution possible, the task feature of OpenMP should be useful for better managing the imbalance. Also there is a Fourier variation that exhibit more parallelism [2].

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557. The work is achieved using the PRACE Research Infrastructure resources [give the machine names, and the corresponding sites and countries].

References

1. R.W. Hockney : A fast direct solution of Poisson’s equation using Fourier Analysis. Journal of Asso. Comput. Mach,v 8,1965.
2. B.L. Buzbee , G.H Golub & C.W. Nielson : On direct methods for solving Poisson’s equation. SIAM J. Numerical Analysis,v 7,1970.