SCCS 694
Case Studies on Parallelization of NASA's Weather Code: Results on Interpolation Techniques

Director : Geoffrey Fox
Project Leader : Miloje Makivic
Research Assistant : Manu Natarajan

Northeast Parallel Architectures Center
Syracuse University, 111, College Place
Syracuse, NY, 13244-4100

Contents

1 Abstract

The purpose of this project is to investigate suitable domain decompositions that lead to efficient parallelizations of the interpolation techniques used in the Semi-Lagrangian advection scheme. In this report we present the results of our investigation using the lasagna decomposition of the input domain of the bilinear and bicubic interpolations used in NASA's 2-D Shallow Water Model.

Our approach to the usage of the lasagna decomposition reduces the amount of communication needed for either the bilinear or bicubic interpolations. The criteria for goodness of the algorithm is based on the ratio of the useful work done to the amount of communication required.

2 Introduction

Weather forecasts are essential for the planning of many activities. They need to be accurate and available in time. The basis of weather forecasting are Numerical Weather Prediction (NWP) models which assimilate data and predict future conditions of atmosphere. Given the requirement that forecasts be available in real-time, the numerical models must be efficient. There is also considerable interest in building quality datasets or model predictions of weather conditions for research purposes and as case studies. These involve processing of voluminous data using the numerical models in an efficient manner.

One chronic problem with NWP models is that the longest time step that can be chosen is decided by stability rather than accuracy. This is alleviated to a considerable degree in the Semi-Lagrangian Transport Model using a semi-implicit time differencing scheme [4, 5]. This was demonstrated for a limited area shallow water model (one layer, homogeneous fluid). An excellent review of Semi-Lagrangian integration schemes for atmospheric models is treated by [6].

Later work addressed the deficiencies associated with polar singularity in the latitude-longitude co-ordinate system [3]. NASA implemented a modified model based on this paper

Another notable issue is that for accuracy to be improved, among other things higher resolutions are required. This calls for efficient implementations on powerful machines. Effort has been underway to implement these models on parallel computers [1].

In order to establish the most suitable paradigm, certain modules of NASA's serial implementation were identified as test-cases. Initial studies at Syracuse University and NASA have indicated that the interpolation techniques deployed occupied a total running time of 25% of running time in the serial code. It was also observed that the usage of the construct "forall" did not have a good speed-up in the data parallel prototype on the CM-5. This can be attributed to the irregular, non-local pattern of data accesses. In addition the decomposition was carried out in the North-South axis as shown in Fig.

[1]Weather Modelling project undertaken at NASA Goddard Laboratory, Maryland

1; as one expects winds to be mostly in the East-West direction. This lead to a very poor utilization of the nodes when there was a small grid size and one of the grid axes declared as serial or local to a node. In such a case the data is distributed among only a few nodes along the news or parallel axis leaving remaining processors with no work [1] The default news:news (square or ravioli) decomposition showed better results. This is because the run time library is able to ensure a better utilzation of nodes with the default decomposition. But this decompostion does not take advantage the fact that the winds are of a East-West nature.

Hence, the message passing model was proposed as an alternate parallelization technique as it offers a greater flexibility in choice of domain decomposition. One does not need to depend on the vagaries of run-time libraries or systems to decide the utilization of nodes. Scalable decompositions may be developed. Additionally, the decomposition can be adapted to ensure that the pole regions are treated differently than regions near the equator. This is because, at the poles the winds move around the pole (entire latitude). This is another disadvantage using data parallel techniques which permit only simple decompositions. Hence, nearest neighbor communication is not possible at the pole for the default square decomposition on CM-5 when using a latitude-longitude grid.

A lasagna decomposition along the N-S axis was adopted as the first test case as seen in Fig. 2. This was postulated to have mostly nearest neighbor communication in the N-S direction. The polar region problem is avoided as the lasagna strips are in E-W direction. The algorithm uses a message-passing version of the data-parallel construct "cshift". The maximum number of "cshifts" is determined at run-time at every time -step for each component. The necessary data from the circulated wind "strips" is collected and a local interpolation is performed at every time step. Two strategies and their merits are discussed in the next section.

2.1 Metrics

We propose some measurements for assessing the goodness of the algorithm as well as giving direction to future work :

- Overall running time
- Ratio of calculation to communication.
- The maximum displacement in terms of nodal distance in the N-S direction.
- Speed-up of interpolation
- Accuracy of calculations

The first criterion shows how well the algorithm performs with respect to the serial code as the problem size grows. The numerical value is of not much interest. But, a very low figure would indicate that the algorithm was ill-conceived. The second may give a clue as to what could be a suitable ratio of the partition dimensions in more complex decompositions. It would also verify if the assumption that the winds are mostly E-W is true. The third criterion indicates how well the algorithm will scale when a global 3-D Semi-lagrangian model is used. As far as the fourth criterion is concerned, speedups will be obtained by comparing the performance of the code on a 32 node CM-5 with the performance on a single node on the CM-5. The last criterion is important as it indicates the presence of errors other than roundoff. This done by finding out if the difference between the interpolated values obtained from both the serial and parallel implementations are homogenous for a given resolution.

3 Design

The problem of parallelizing the bilinear and bicubic interpolation is first considered by using a simple approach. In this lasagna decomposition each node receives an input partition of the latitude, longitude grid and each of the two wind components [Fig 2]. Each node then computes a set of points (ordered pairs) (ii,jj) from the latitude, longitude grid, based on which an interpolation is carried out. In case of the bilinear interpolation, associated with the set of points (ii,jj) there exist four sets of points (ii,jj+2), (ii,jj+3), (ii+1),(jj+2), (ii+1,jj+3) which are required for the interpolation [Fig. 3]. In case of the bicubic interpolation there are sixteen such sets of points [Fig. 4] [2].

In the first approach to interpolation, the algorithm is outlined below.

- Do for all (ii,jj)
  - Find the 4 points (ii,jj+2), (ii,jj+3), (ii+1,jj+2), (ii+1,jj+3)
  - Obtain the values of the wind component at above points.
    - If the value is not available locally, *fetch* from the appropriate neighboring node
  - Find the interpolation point.
  - Compute the value at interpolation point.
- End of do loop

It is obvious that though the above approach is simple, it is impractical to implement because of the huge communication overhead involved. For, every point that is not available locally, a communication call will have to be generated to fetch the value at that point. There are at least two approaches which lower the communication overhead. In both the approaches strips of the wind components are shifted from node to node. The shifting occurs in both the southern and northern directions. The amount of shifting is determined by the approach discussed in the next section.

[2] Hereafter, the discussion will consider only the bilinear interpolation, but the same reasoning holds good for the bicubic interpolation. Appendix A covers the bicubic interpolation algorithm.

3.1 Pre-Determination of Degree of Circulation

If the assumption that the winds in the North-South direction mostly result in nearest neighbor communicationis valid , then the following strategy may be used. For any given global indice (ii, jj), we determine the node it belongs to. The inter-nodal distance is then simply the difference between the owner node and user node. But, this distance has a direction, i.e. the "owner" node may be reached in the North or South Direction [Fig. 5]. Hence, this is converted to the inter-nodal distance between the two nodes when going in the North direction. Moreover, if the inter-nodal distance exceeds half the number of nodes, the shorter path is in the Southern direction. For, either interpolation there are two extreme points in the North-South direction [Fig. 3,4]. The inter-nodal distance is computed for both the points separately, for all the indices on a node. The maximum inter-nodal distance in a direction (both North and South are evaluated) is then found by a global reduction. The two maxima in the North-South direction govern the circulation.

The algorithm for *the pre-determination of degree of circulation* is below

- ny = Number of nodes (processors) used
- Let 'myaddr' denote the address of the node executing the code
- Let 'ownp0' denote nodal address where value 'jj(i,j)+0' resides
- Let 'ownp3' denote nodal address where value 'jj(i,j)+3' resides
- Number of elements in E-W direction for a partition is imp3
- Number of elements in N-S direction for a partition is range
- for j = 1, range
  - for i = 1, imp3
    - ownp0 = jj(i,j) / partition width
    - ownp3 = (jj(i,j) + 3) / partition width
    - Find the inter-nodal distance in *northern direction* from 'myaddr' to both ownp0 and ownp3
      - distp0 = ownp0 - myaddr
      - distp3 = ownp3 - myaddr
      - if (distp0 .lt. 0),distp0 = ny + distp0
      - if (distp3 .lt. 0),distp3 = ny + distp3
    - Find the shortest inter-nodal distance for both ownp0, ownp3, and update the local maximum inter-nodal distances if required
- Find the global max. northern dist. (reduction operation)
- Find the global max. southern dist. (reduction operation)

Expansion of Updation of local maximum inter-nodal distances

- Let distp0 be the inter-nodal distance in *northern direction* from 'myaddr' to ownp0
- Let distp3 be the inter-nodal distance in *northern direction* from 'myaddr' to ownp3
- If distp0 .gt. (ny / 2)
  - distp0 = ny - distp0
  - if (max. southern dist .lt. distp0) max. southern dist = distp0
- Else
  - if (max. northern dist .lt. distp0) max. northern dist = distp0
- If distp3 .gt. (ny / 2)
  - distp3 = ny - distp3
  - if (max. southern dist .lt. distp3) max. southern dist = distp3
- Else
  - if (max. northern dist .lt. distp3) max. northern dist = distp3

3.2 Reconstruction of the Sets of Points Approach

In the second approach we wish to fully reconstruct the four associated sets of points (ii,jj+2), (ii,jj+3), (ii+1),(jj+2), (ii+1,jj+3) in four arrays for every node. Then the four arrays are indexed to obtain the four points required for interpolation on every node. As a wind component has been partitioned as shown in Fig. 2, there is an upper and a lower bound on the index jj in the North-South direction that decides which values of a wind component may be found locally. Thus, at every shift of the wind strips, the four arrays maybe partially reconstructed using the new bounds obtained with every shift. At the end of the shifting the arrays should be fully reconstructed. However, there will be points such that the points (ii,jj) will be local but (x,jj+2) or (x,jj+3) may not be local as seen in figure 3 [3]. Thus it will not be possible to reconstruct *entirely locally* the four associated sets of points for all jj that that fall within the bounds of the node's partition. *If the reconstruction is not done entirely locally, there would be additional communication overhead to fetch the missing values one at a time, as in the first approach.* This hitch may be rectified by initially creating a 3-element wide strip in the Northern direction by fetching those elements from the neighboring nodes [Fig. 6]. The creation of the Northern buffer is created before performing any shifts. The original partition and this northern buffer is now termed a segment. These segments

[3] x maybe ii or ii + 1. This does not matter as in the East-West direction the elements of the wind component are local to a node.

are shifted instead of the original partition. The algorithm for the second approach is outlined as follows, which is executed once for each of the two wind components.

First level refinement for the algorithm.

- Number of elements in E-W direction for a partition is imp3
- Number of elements in N-S direction for a partition is range
- Pre-determine the degree of circulation
- Construct the northern buffer.
- Select elements from segment
- Fetch segments and select elements from southern direction
- Fetch segments and select elements from northern direction
- Perform local interpolation

In this approach, one is guaranteed that if the value at jj(i,j) is resident on a node, then the value at jj(i,j)+3 is resident [4]. Therefore, only the node where jj(i,j) is resident, need be considered for determination of the local maximum inter-nodal distance. This the only deviation from the strategy outlined in the previous subsection.

Algorithm for *construction of northern buffer*

- Copy original wind partition to temporary buffer (depart)
- If partition width less than 3 then
  - While northern buffer incomplete
    - Send depart to southern node
    - Receive partition into northern buffer from northern node
    - Copy received partition into depart
  - End of while
- Else (partition width is .ge. 3)
  - Send original partition to southern node
  - Receive new partition in northern buffer from northern node
- Append northern buffer to original wind partition. This will be called segment.

[4] As there is the 3-element northern buffer appended to the original partition.

Algorithm for *fetching of segments from southern direction*

- Current partition bounds = Original partition bounds
- Copy original segment into temporary buffer (depart)
- For number of hops in southern direction
  - Send upper & lower bounds of current partition to northern node.
  - Receive upper & lower bounds of new partition from southern node.
  - Send 'depart' to northern node.
  - Receive new segment from southern node in buffer 'landing'.
  - Current partition bounds = new partition bounds
  - Copy 'landing' into 'depart'
  - Select elements from landing (new segment)

Algorithm for *fetching of segments from northern direction*

- Current partition bounds = Original partition bounds
- Copy original segment into temporary buffer (depart)
- For number of hops in northern direction
  - Send upper & lower bounds of current partition to southern node.
  - Receive upper & lower bounds of new partition from northern node.
  - Send 'depart' to southern node.
  - Receive new segment from northern node in buffer 'landing'.
  - Current partition bounds = new partition bounds
  - Copy 'landing' into 'depart'
  - Select elements from landing (new segment)

Algorithm for *selection of elements from segment*

- Number of elements in E-W direction for a partition is imp3
- Number of elements in N-S direction for a partition is range
- Input :
  - lower & upper partition bounds [Fig. 6]
  - segment (corresponding to partition as in Fig 6)
  - set of points {(ii,jj)}
  - arrays to hold the four sets of points - w1, w2, w3, w4
- For i = 1, imp3
  - For j = 1, range
    - If (jj(i,j) .le. upper) .and. (jj(i,j) .ge. lower)
      - w1 = segment(ii(i,j), jj(i,j) +2)
      - w2 = segment(ii(i,j), jj(i,j) +3)
      - w3 = segment(ii(i,j)+1, jj(i,j) +2)
      - w4 = segment(ii(i,j)+1, jj(i,j) +3)

Algorithm for *bilinear interpolation.* Fig. 3 explains the terms delx, dely.

- Number of elements in E-W direction for a partition is imp3
- Number of elements in N-S direction for a partition is range
- for j = 1, range
  - for i = 1, imp3
    - value at interpolated point = (1.0 - delx(i,j)) * ((1.0 - dely(i,j)) * w1(i,j) + dely(i,j) * w2(i,j)) + delx(i,j) * ((1.0 - dely(i,j)) * w3(i,j) + dely(i,j) * w4(i,j))

The drawback with this approach is that the communication overhead for creating the northern buffer increases when the width of an individual strip is smaller than the overlap region [Fig. 7].

3.3 Reconstruction of Segment Approach

The third approach recognizes the fact that given the set of computed points (ii,jj) on each node, the node only needs to see a segment of the entire wind component to perform a local interpolation; there is no necessity for reconstructing the associated four sets of points. This segment is not the same as in the second approach, in that its size is not a static quantity. The size of the segment depends on how far back one needs to go along the fluid trajectory to perform an interpolation, and hence is a dynamic quantity. Thus, no extra communication is needed to establish a northern buffer. Additionally, only the *original* partitions of the wind component need be shifted to create the segment. The circulated wind strips are appended as shown in Fig. 8. Some, book-keeping needs to be done in order to the relative position of the partition in the segment and for generation of local indices. The interpolation is then carried out locally. The four points are obtained by the use of the local indices. The algorithm is outlined below.

The *first-level refinement* for the third approach. The individual steps are elaborated subsequently.

- Number of elements in E-W direction for a partition is imp3
- Number of elements in N-S direction for a partition is range
- Pre-determine the degree of circulation
- Copy the partition into a large buffer called segment
- Fetch partitions from the southern direction and append to segment.
- Fetch partitions from the northern direction and append to segment.
- Convert global indices to local indices.
- Perform the interpolation locally.

The pre-determination of the degree of circulation is exactly as outlined in section 3.1.

The algorithm for *fetching of partitions from the southern direction and appending to the segment* is as below.

- Current partition bounds = Original partition bounds
- Copy partition into temporary buffer (depart)
- For number of hops in southern direction
  - Send upper & lower bounds of current partition to northern node.
  - Receive upper & lower bounds of new partition from southern node.
  - Send 'depart' to northern node.
  - Receive new partition from southern node in buffer 'landing'.
  - Update parameters needed for generation of local indices.
  - Move up the previously filled columns in the segment by width of the new partition (southern partitions occupy lower indices in the segments [Fig. 8]).
  - Append the new partition to the 'southern end' of the segment.
  - Update width of segment.
  - Copy 'landing' into 'depart'

The algorithm for *fetching of partitions from the northern direction and appending to the segment* is as below.

- Current partition bounds = Original partition bounds
- Copy partition into temporary buffer (depart)
- For number of hops in northern direction
  - Send upper & lower bounds of current partition to southern node.
  - Receive upper & lower bounds of new partition from northern node.
  - Send 'depart' to southern node.
  - Receive new partition from northern node in buffer 'landing'.
  - Update parameters needed for generation of local indices.
  - Append 'landing' to the 'northern end' of the segment; northern partitions occupy higher indices in the segment [Fig 8].
  - Update width of segment.
  - Copy 'landing' into 'depart'

The local indices are generated with the following considerations :

- The first column in any segment does not necessarily correspond to column '1' in the wind component.
- There maybe a wrap around in the southern direction; i.e. the southern most node communicated may cross the south pole.
- Similarly there may be a wrap around in the northern direction.

The algorithm *generation of local indices* is as below. Fig. 9 illustrates the algorithm.

- co_lower = lower bound of non-wrapped part of segment
- wr_lower = lower bound of wrapped part of segment
- wrapd = width of wrapped part of segment.
- Let 'njj' denote the set of new (local) indices
- If there is no wrap around in either direction
  - for i = 1, imp3
    - for j = 1, range
      - njj(i,j) = jj(i,j) - co_lower + 1
- If there is wrap around in southern direction
  - for i = 1, imp3
    - for j = 1, range
      - if jj(i,j) .ge. wr_lower then njj(i,j) = jj(i,j) - wr_lower + 1
      - Else njj(i,j) = jj(i,j) + wrapd
- If there is wrap around in northern direction
  - for i = 1, imp3
    - for j = 1, range
      - if jj(i,j) refers to the wrapped part then njj(i,j) = jj(i,j) + jmp3 - co_lower + 1
      - Else njj(i,j) = jj(i,j) - co_lower + 1

The algorithm for *the bilinear interpolation* is as below. This is similar to the algorithm in the serial code, excepting that local indices are used to index a segment of the wind component on every processor.

- Let 'seg' denote segment
- Let 'njj' denote new (local) jj
- Number of elements in E-W direction for a partition is imp3
- Number of elements in N-S direction for a partition is range
- for j = 1, range
  - for i = 1, imp3
    value at interpolated point = (1.0 - delx(i,j)) * ((1.0 - dely(i,j)) * seg(ii(i,j), njj(i,j)+2)) + dely(i,j) * seg(ii(i,j), njj(i,j)+3) + delx(i,j) * ((1.0 - dely(i,j)) * seg(ii(i,j)+1, njj(i,j)+2) + dely(i,j) * seg(ii(i,j)+1, njj(i,j)+3))

4 Implementation

The project was implemented on CM-5 (CMOST 7.3) using the native message passing library [2]. This was developed as a proof of concept that the message passing paradigm was a suitable way of implementing a parallel version of the Semi-Lagrangian Transport code. It is hence highly modular and has about five levels of subroutine calls. Though this allows for the easy testing of the code, it conflicts with the requirements of high performance computing in that the overall running time drastically increases. This prompted us in measuring computations without the function calling overhead. Therefore the running time of a production code would be between the timings with the function call overhead and without the overhead. Ideally we would like it to be closer to the timings without the overhead.

The third approach was implemented and instrumented for measuring performance as discussed in the next section.

5 Performance

The timings were obtained on CM-5 on CMOST 7.3 on a 32 node partition. The same code was run on a single node of the CM-5 in order to measure the speedup [5]. The performance is discussed for both bicubic and bilinear interpolations with respect to the five criteria discussed in the Introduction. They are as follows :

- The overall running time.
- The maximum displacement in terms of nodal distance in the N-S direction.
- Ratio of calculation to communication.
- Speed-up of interpolation
- Accuracy of calculations

Before discussing the performance issues, we will first define the following terms :

- Averaged Maximum Number of hops to northern nodes NI.
- Averaged Maximum Number of hops to southern nodes SI.
- Total Number of hops in both directions $TI = (NI + SI)$.
- The total communication time per iteration of fetching or gathering partitions from southern (or northern nodes) T_{exp}. This is the sum of the time for sending and receiving a bound of the partition plus sending and receiving a partition. This time is derived by measuring the communication time for various message sizes.
- Time required for bicubic (or bilinear) interpolation on the 32 node partition (without function overhead) T_{int}.
- Time required for bicubic (or bilinear) interpolation on the single node (without function overhead) T_{int_s}.
- Speedup of bicubic (or bilinear) interpolation $S_{int} = T_{int_s}/T_{int}$.
- Time required for reduction of address calculations on the 32 node partition (without function overhead) T_{ra}. This overhead is a result of the implementation of the code for the Bicubic Interpolation alone and does not show up at the design stage.
- Time required for reduction of address calculations on the single node (without function overhead) T_{ra_s}.

[5]The flow of program execution within a subroutine is however, not necessarily the same. This results in less than ideal speedup for some parts of the code.

- Speedup of reduction of address calculations $S_{ra} = T_{ra_s}/T_{ra}$.
- Time required for conversion of the global indices to local indices on the 32 node partition (without function overhead) T_{li}.
- Time required for conversion of the global indices to local indices on the single node (without function overhead) T_{li_s}.
- Speedup of conversion to local indices $S_{li} = T_{li_s}/T_{li}$.
- Time taken for node-to-node communication (circulation of partitions of winds and the bounds of the partition) on the 32 node partition T_{co_n}.
- The product of TI and T_{exp} is $T_{co_{ne}}$. This is approximately equal to T_{co_n} .
- Time required for book-keeping and buffer copying in creation of segment (no function calling overhead or node-to-node comunnication is accounted for) on the 32 node partition T_{bc}.
- Time required for book-keeping and buffer copying in creation of segment (no function calling overhead is accounted for) on the single node T_{bc_s}. There is no communication required on the single node for creation of segment.
- Time required for creation of segment on the 32 node partition $T_{cs} = T_{bc} + T_{co_n}$.
- Speedup of buffer copying in creation of segment $S_{bc} = T_{bc_s}/T_{bc}$. This indicates how well the buffer copying alone scales when comparing the timings on the 32 node and the single node in CM-5. The next speedup includes the cost of the node-to-node communication which is a part of the "creation of segment" function It is a "gathering" process. .
- Speedup of creating a segment $S_{cs} = T_{bc_s}/T_{bc}$. This includes the node-to-node communication overhead in the computation of the speedup.
- Time taken for global maximum operation on the 32 node partition T_{co_g}.
- Time taken for finding local maxima in determining degree of circulation (no global reduction involved) on 32 node partition T_{dcl}.
- Time taken for determining degree of circulation $T_{dc} = T_{dcl} + T_{co_g}$.
- Time taken for finding local maxima in determining degree of circulation on single node partition. Note the single node does not need a global maximum operation T_{dcl_s}.
- Speedup of finding local maxima (calculations) $S_{dcl} = T_{dcl_s}/T_{dcl}$. This indicates how well the computation alone scales when comparing the timings on the 32 node and the single node in CM-5. The next speedup includes the cost of the global maximum operation.

- Speedup of finding degree of circulation $S_{dc} = T_{dcl_s}/T_{dc}$. This speedup includes the cost of the global maximum operation.
- Time taken for latitude, longitude index calculations (no function calling overhead) on the 32 node partition T_{ll}.
- Time taken for latitude, longitude index calculations (no function calling overhead) on the single node T_{ll_s}.
- Speedup of latitude, longitude index calculations $S_{ll} = T_{ll_s}/T_{ll}$.
- Total calculation (overall) time (with function calling overhead and timing overhead) on the 32 node partition T_{ca_o}. This is the sum of T_{ca_a} and function calling overhead.
- Total communication time (includes both global communication and node to node communication) on the 32 node partition $T_{co} = T_{co_g} + T_{co_n}$
- Ratio of calculation time (overall) to communication time on the 32-node CM-5 $R_a = T_{ca_o}/T_{co}$
- Total calculation time (adjusted, i.e. no function calling overhead) on the 32 node partition $T_{ca_a} = T_{ll} + T_{dcl} + T_{bc} + T_{li} + [T_{ra}] + T_{int}$ [6]
- Ratio of calculation time (adjusted) to communication time on the 32-node CM-5 $R_a = T_{ca_a}/T_{co}$.
- Total calculation time (adjusted, i.e. no function calling overhead) on the single node $T_{ca_{as}} = T_{ll_s} + T_{dcl_s} + T_{bc_s} + T_{li_s} + [T_{ra_s}] + T_{int_s}$.
- Speedup (adjusted) $S_a = T_{ca_{as}}/(T_{ca_a} + T_{co})$.
- Total (Overall) running time T_{ot} (includes function calling overhead, communication overhead and timing overhead) on a 32 node partition on CM5. $T_{gt} = T_{ca_o} + T_{co}$. This is also the sum of T_{ca_a}, T_{co} and function calling overhead.
- Total (Overall) running time T_{ot_s} (includes function calling overhead and timing overhead) on a single node on CM5. This is the sum of $T_{ca_{as}}$ and function calling overhead.
- Speedup (overall) $S_o = T_{gt_s}/T_{gt}$.

[6] T_{ra} is shown in square brackets as there is no reduction of address calculations for Bilinear Interpolation. Similarly T_{ra_s} is shown in square brackets for the definition of $T_{ca_{as}}$.

The first table shows the non-linear nature of the communication overhead versus message size. The overhead decreases as message size increases from 0 to 8 bytes. Somewhere between a size of 8 to 16 bytes, the overhead starts increasing again and again becomes lower at 32 bytes. The machine also exhibits a high latency; for example messages of size 64 bytes takes nearly the same amount of time as a message of 256 bytes for a synchronous send and receive.

Table 1 : Communciation Time vs Message Size

Message Size (bytes)	Communication Time (ms)
0	0.226
1	0.200
2	0.176
4	0.172
8	0.155
16	0.221
32	0.218
64	0.232
256	0.250
512	0.300
1280	0.428
2048	0.799
5120	1.584
7168	1.751
10000	2.591
22528	5.218

Table 2 shows the message size used at a given resolution. From Tables 1 and 2, we derive the values for T_{exp}. T_{exp} is the sum of the times for communicating the bound of a partition (4 bytes long) and the partition itself (message size at given resolution).

Table 2 : Message Size Used At Given Resolution

Resolution (bytes)	Message Size (bytes)
(8x64)	256
(64x64)	2048
(128x128)	5120
(128x256)	7168
(256x512)	22528

5.1 Bicubic Interpolation

The reconstruction of segment approach was adopted for the bicubic interpolation. Table 3 shows the overall results with the function calling overheads. Table 4 shows the overall results without function calling overhead [7]. Tables 5 to 9 show the functional breakup of the timings in Table 4.

Table 3 : Overall Results

Resolution (I,J)	T_{ot} ms	T_{ot_s} ms	S_o	T_{ca_o} ms	T_{co} ms	R_o
(8,64)	14.62	190.00	13.0	11.86	2.76	4.29
(64,64)	63.60	1146.20	18.0	58.41	5.18	11.26
(128,128)	211.19	4500.30	21.2	213.76	8.17	24.94
(128,256)	396.70	9122.00	23.0	388.00	8.66	44.78
(256,512)	1740.40	——	–	1718.5	21.91	78.43

Table 4 : Results Without Function Calling Overhead

Resolution (I,J)	T_{ca_a} ms	R_a	$T_{ca_{as}}$ ms	S_a
(8,64)	4.85	1.75	85.1	11.18
(64,64)	26.42	5.09	522.8	17.5
(128,128)	92.04	11.27	1985.00	19.8
(128,256)	173.45	20.01	4034.00	22.15
(256,512)	714.75	32.62	——	——

Table 5 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{ll} ms	T_{ll_s} ms	S_{ll}
(8,64)	0.58	15.16	26.14
(64,64)	3.20	92.34	28.86
(128,128)	12.76	364.7	28.58
(128,256)	27.57	732.5	26.56
(256,512)	99.24	——	——

[7] The function calling overhead does not apply to the timing of the communication routines.

Table 6 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{co_g} ms	T_{dcl} ms	T_{dc} ms	T_{dcl_s} ms	S_{dcl}	S_{dc}
(8,64)	0.0774	0.59	0.667	13.82	23.42	20.70
(64,64)	0.0829	3.30	3.38	77.26	23.41	22.88
(128,128)	0.0930	12.99	13.08	305.68	23.53	23.37
(128,256)	0.0910	26.29	26.38	613.21	23.32	23.24
(256,512)	0.1070	108.8	108.9	——	—–	—–

Table 7 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{co_n} ms	T_{bc} ms	T_{cs} ms	T_{bc_s} ms	S_{bc}	S_{cs}
(8,64)	2.69	1.51	4.2	5.52	3.66	1.31
(64,64)	5.11	6.81	11.92	33.65	4.94	2.82
(128,128)	8.08	17.31	25.39	128.77	7.44	5.07
(128,256)	8.57	24.41	32.98	293.8	12.04	8.91
(256,512)	21.8	87.35	109.15	——	—–	—-

Table 8 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{li} ms	T_{li_s} ms	S_{li}	T_{ra} ms	T_{ra_s} ms	S_{ra}
(8,64)	0.274	2.16	7.88	0.85	15.03	17.68
(64,64)	1.55	13.77	8.88	5.42	89.55	16.52
(128,128)	4.04	53.17	13.16	20.93	360.56	17.22
(128,256)	5.55	109.65	19.75	42.80	734.56	17.16
(256,512)	17.2	——	——	90.92	——	—–

Table 9 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{int} ms	T_{int_s} ms	S_{int}
(8,64)	1.04	34.02	32.7
(64,64)	6.13	216.25	35.28
(128,128)	24.02	777.20	32.15
(128,256)	46.81	1550.9	33.12
(256,512)	311.29	——	——

Table 10 : Experimental Confirmation of Node-to-Node Communication Time T_{co_n}

Resolution (I,J)	NH	SH	TH	T_{exp} ms	$T_{co_n e}$ ms
(8,64)	4.0	1.5	5.5	0.422	2.32
(64,64)	4.2	1.5	5.7	0.971	5.53
(128,128)	3.1	2.0	5.1	1.756	8.96
(128,256)	2.0	2.0	4.0	1.923	7.69
(256,512)	2.0	2.0	4.0	5.390	21.56

Since the code has 5 levels of subroutine calling, there is approximately a factor of 2 difference between the overall timings with and without the function calling overhead. A good production code would have a *overall timing* somewhere between these two figures [8].

The *inter-nodal distance* seems to be fairly static for both components (north, south directions inclusive). On an average this is 4 ”cshifts” in the Northern direction or 1.5 ”cshifts” in the Southern for a resolution of 8 x 64. This appears to stabilize at 2 “cshifts” in either direction at higher resolutions as seen from Table 10. This is close to the assumption that the winds generate mostly nearest neighbor communication.

The *ratio of calculation to communication* (R_o or R_a) (Tables 3,4) falls with increasing resolution because of two factors, namely :

- High latency in the communciation overhead. Due to high latency, it takes almost the same time to send and receive a message of 256 bytes as a message of size 64 bytes.
- The decreasing total number of cshifts TI. The total number of “cshifts” falls from 5.5 to 4.

It is obvious that this algorithm is dominated by overheads other than communication at high resolutions, which will be established shortly [9].

It is also noticed since the function calling overhead is high, the ratio R_a is less than R_o. Consequently, the *speedup* S_a is lower than S_o. However, S_o itself is much lower than ideal. At low resolutions, this is primarily due to the high communication overhead [10], but does not hold true at higher resolutions. Hence, a complete picture of speedup will not emerge without a detailed discussion at the functional level of the code. We will first tabulate the functions in order of increasing speedup for the resolution of 8x64 (Table 11) from the data in Tables 3 to 7 and then analyze the reasons for the performance. We will do this similarly for a higher resolution of 128x256 (Table 12).

[8] It is not desirable to eliminate totally the function calling overhead except where it is in excess. For example, one should inline the calls to the actual interpolation routine as it is called within an inner loop of a doubly nested loop.

[9] Communication overhead is only 1.28 % at a resolution of 128x256.

[10] At a resolution of 8 x 64, the ratio R_o is as low as 4.29.

Table 11 : Functional Breakup of Timings at a Resolution of 8 x 64:

Function	Speedup	%age of $T_{ca_a} + T_{co}$	%age of T_{ot}
Creation of Segment (cs)	1.31	53.98 %	28.1 %
Buffer Copying in above (bc)	3.66	19.79 %	10.3 %
Conversion to Local Indices (li)	7.88	3.65 %	1.9 %
Reduction of Address Calculation (ra)	17.68	11.14 %	5.8 %
Degree of Circulation (dc)	20.70	8.83 %	4.6 %
Computation alone in above (dcl)	23.42	7.68 %	4.0 %
Latitude, Longitude Index Calculations (ll)	26.14	7.68 %	4.0 %
Interpolation (int)	32.7	13.45 %	7.0 %

The "creation of segment" function shows a very low speedup primarily because it involves node-to-node communication with high latency. But, this does not account for all the loss in speedup (speedup of buffer copying is also at a low of 3.66). This is because there are more places in the 32-node run where there is *buffer copying* [11]. The term buffer copying includes memory to memory copying and the book-keeping associated with it. There is some load imbalance in the memory to memory copying, especially at the low resolutions, due to nodes at the poles having to deal with a larger partition (8x5 instead of 8x2) as shown in Figure 2. These nodes also have to handle with the pole-wrap problem [12] which are absent in other nodes, leading to an additional load imbalance. The load imbalance is apparent from the fact that at the next higher resolution (64x64), though the grid points have increased by 8, the amount of time spent in buffer copying increases by 4. Otherwise the buffer copying scales with number of grid points. In the single node we find that the time spent in buffer copying scales well with the number of grid points.

Taking into account the implementation and the sizes of buffers involved in both the runs, the single node run performs only about 6 times the amount [13] of buffer copying as the 32 node run. This is slightly more than the speedup of 3.66. However, due to the load imbalance the drop in speedup can be expected . As the buffer copying alone takes 19.79 % of running time (without function calling overhead), this has a significant effect on the overall speedup [14] .

The speedup of "conversion to local indices" is also low. This could be due to an error in measurement of a very small interval of time (0.27 ms) in the 32-node run. There is also the possibility that due to a load imbalance that affects the nodes corresponding to

[11]Though both the single node and the 32 node run the same code, because the single node has no communication to perform, it does not need to perform the buffer copying associated with the sending and receiving of partitions. In other words the single node program traces a different path through the code. We will hereafter refer to this as a *run*.

[12]Fetching of partition by wrapping around the pole, needs more book-keeping.

[13]Not to be confused with number of places

[14]Inclusive of the communication cost, it takes the lion's share of the running time at 53.98 %.

the polar regions which have to translate indices that cross the poles, that the speedup is low. This code involves testing of a condition in the innermost loop of a doubly nested loop. This does not occur in the single node run where only a straight forward translation to local indices takes place. But, the impact of this function is slight as it constitutes only 3.65 % of the running time (which is the least of all the functions).

The "reduction of address calculations" constitutes about 11.14 % of the running time. The number of times it is called for the 32-node run is equal to the number of grid points in a partition. There are 16 grid points at a resolution of 8x64. Therefore the approximate time per call is only 53 μs which spans 8 integer operations. The measurement of such a small time is prone to errors, which may have led to a somewhat moderate speedup of 17 to 18. [15]

The "degree of circulation" has a moderate speedup of 20.7. When the global maximum operation is left out, the speedup is about 23.4, which is fairly high. As the global maximum operation is a constant overhead, its effect at higher resolutions becomes negligible. As the code involves testing of conditions and as the quantities involved are dynamically varying with both time and space, it is difficult to pinpoint how the speedup is affected. Additionally this function has no significance for the single node run.

The "calculation of latitude, longitude indices" shows a good speedup of 26.1. This involves a mix of integer and floating point arithmetic.

The actual "bicubic interpolation" shows more than ideal speedup. The timing of this routine has the same drawback as the "reduction of address calculations". As the time intervals involved are very small, about 65 μs, the measurements are prone to errors. However, it may be concluded that this function has a good speedup even after discounting for any errors.

The overall speedup S_a is therefore poor because functions which have poor speedups (< 16) constitute 57.6 % of running time (with no function calling overhead), those with medium speedup ($16 \leq speedup \leq 23$) constitute 20.0 % and those with good speedups 22.2 % (> 23) [16].

Table 12 : Functional Breakup of Timings at a Resolution of 128x256:

Function	Speedup	%age of $T_{ca_a} + T_{co}$	%age of T_{ot}
Creation of Segment (cs)	8.91	18.1 %	8.31 %
Buffer Copying in above (bc)	12.04	13.40 %	6.15 %
Reduction of Address Calculation (ra)	17.16	23.53 %	10.8 %
Conversion to Local Indices (li)	19.75	3.05 %	1.4 %
Degree of Circulation (dc)	23.24	14.49 %	6.65 %
Computation alone in above (dcl)	23.32	14.42 %	6.62 %
Latitude, Longitude Index Calculations (ll)	26.56	15.14 %	6.95 %
Interpolation (int)	33.1	25.70 %	11.8 %

[15] Even the timings for the single node run suffer from the same defect.

[16] The classification of poor, medium and good speedups may seem arbitrary. But this distinction serves the purpose considering that timings were obtained on a 32 node machine.

At this resolution cache miss penalties play an important role as seen from Tables 5 to 9 where there is a jump in speedups from the 128x128 to 128x256 resolution [17].

Using the same reasoning as in the case of 8x64 resolution for the "creation of segment" function, the single-node run has about 6 times the amount of *buffer copying* as the 32 node-run. However, the speedup of buffer copying is 12.04 which is two times more than expected. This is most likely due to caching effects. It can also be seen that there is an increase of 1-1/2 times in T_{bc} compared with a two-fold increase in the number of grid points from a resolution of 128x128. This indicates that the load imbalance due to unven partition size and pole wrapping is becoming less severe compared to a resolution of 8x64.

The node-to-node communication is still expensive as the speedup inclusive of this cost falls to 8.91 from 12.04. However, the adverse effect of the "creation of segment" function is less on the overall speedup as it only takes 18.1 % of the running time compared to 53.98 % at the 8x64 resolution.

The speedup of "conversion to local indices" increases to 19.75. This is certainly due to cache misses, because at a resolution of 128x128 the speedup is only 13.2. At the 128x128 resolution the accuracy of the timing measurements should be good as there are 512 iterations taking place. The speedup is not better than 19.75 probably because of the load imbalance problem at the nodes corresponding to the poles.

The measurement of the total time for the "reduction of calculations" suffers from the drawback discussed earlier. The speedup for this function shows little variance (Table 8). But since it now constitutes 10.8 %, it does little to contribute to a better speedup.

The "degree of circulation" has a fairly high speedup of 23.24. The global maximum operation now constitutes a negligible amount of the time spent in the function, as seen in the difference in speedups S_{dcl} and S_{dc}.

The "calculation of latitude, longitude indices" shows a good speedup of 26.56.

The actual "bicubic interpolation" as before shows more than ideal speedup due to the drawback discussed earlier.

Thus the overall speedup S_a is better because functions which take major part of running time (55.35 %) have good speedups (> 23). About 12 % of the code has medium speedup ($16 \leq speedup \leq 23$) and the rest has a poor speedup (< 16).

An *accuracy* of upto (but not including) the fifth decimal place was obtained between the parallel and serial implementations.

Table 10 confirms the *accuracy of the node-to-node communication overhead* . The results from the secondary experiment, $T_{co_{ne}}$ and T_{co_n} match closely.

[17] It was not possible to run at a resolution of 256x256 on the single node due to lack of memory.

5.2 Bilinear Interpolation

The reconstruction of segment approach was also adopted for the bilinear interpolation. Table 13 shows the overall results with the function calling overheads. Table 14 shows the overall results without function calling overhead [18] . Tables 15 to 19 show the functional breakup of the timings in Table 14. We obtain very similar results for certain functions in the code, because the code is essentially the same excepting for the actual interpolation routine. There is also no "reduction of address calculation" function in this code.

Table 13 : Overall Results

Resolution (I,J)	T_{ot} ms	T_{ot_s} ms	S_o	T_{ca_o} ms	T_{co} ms	R_o
(8,64)	10.50	96.88	9.23	8.22	2.27	3.62
(64,64)	41.12	589.6	14.34	36.63	4.49	8.176
(128,128)	128.46	2280.0	17.74	112.03	6.43	18.97
(128,256)	240.47	4717.0	19.60	232.05	8.42	27.56
(256,512)	955.20	——	–	930.68	24.53	37.95

Table 14 : Results Without Function Calling Overhead

Resolution (I,J)	T_{ca_a} ms	R_a	$T_{ca_{as}}$ ms	S_a
(8,64)	3.37	1.48	47.31	8.39
(64,64)	16.17	3.60	290.3	14.05
(128,128)	52.63	8.18	1142.0	19.34
(128,256)	98.37	11.68	2349.1	22.03
(256,512)	381.45	15.55	——	—–

Table 15 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{ll} ms	T_{ll_s} ms	S_{ll}
(8,64)	0.61	14.72	24.09
(64,64)	2.95	91.87	31.14
(128,128)	11.19	364.8	32.60
(128,256)	22.53	733.0	32.60
(256,512)	91.23	—–	—–

[18] The function calling overhead does not apply to the timing of the communication routines.

Table 16 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{co_g} ms	T_{dcl} ms	T_{dc} ms	T_{dcl_s} ms	S_{dcl}	S_{dc}
(8,64)	0.0756	0.60	0.67	11.55	19.40	17.21
(64,64)	0.0797	3.27	3.35	69.09	21.12	20.64
(128,128)	0.0907	12.88	12.97	276.50	21.47	21.31
(128,256)	0.0890	25.72	25.81	556.30	21.63	21.55
(256,512)	0.0946	102.7	102.79	——	—–	—–

Table 17 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{co_n} ms	T_{bc} ms	T_{cs} ms	T_{bc_s} ms	S_{bc}	S_{cs}
(8,64)	2.20	1.47	3.67	5.50	3.74	1.50
(64,64)	4.41	5.98	10.39	34.59	5.78	3.33
(128,128)	6.34	14.50	20.84	128.69	8.88	6.18
(128,256)	8.33	24.66	32.99	296.50	12.02	8.99
(256,512)	24.43	87.54	111.97	——	—–	—-

Table 18 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{li} ms	T_{li_s} ms	S_{li}
(8,64)	0.28	2.19	7.96
(64,64)	1.56	13.63	8.74
(128,128)	3.86	53.13	13.76
(128,256)	5.44	108.9	20.02
(256,512)	17.2	——	——

Table 19 : Functional Breakup of Table 2 Timings

Resolution (I,J)	T_{int} ms	T_{int_s} ms	S_{int}
(8,64)	0.410	13.36	32.59
(64,64)	2.40	81.11	33.8
(128,128)	10.21	318.78	31.22
(128,256)	19.25	654.56	32.81
(256,512)	82.81	——	——

Table 20 : Experimental Confirmation of Node-to-Node Communication Time T_{co_n}

Resolution (I,J)	NH	SH	TH	T_{exp} ms	T_{co_ne} ms
(8,64)	3.33	1.67	5.0	0.422	2.11
(64,64)	3.33	1.67	5.0	0.971	4.86
(128,128)	2.00	1.93	3.93	1.756	6.90
(128,256)	2.0	2.0	4.0	1.923	7.692
(256,512)	2.0	2.0	4.0	5.390	21.56

Since the code has 5 levels of subroutine calling, there is approximately a factor of 2 difference between the overall timings with and without the function calling overhead. A good production code would have a *overall timing* somewhere between these two figures. We would like the timings to be closer to the value of $T_{ca_a} + T_{co}$ than T_{ot}.

The *inter-nodal distance* seems to be fairly static for both components (north, south directions inclusive). On an average this is 3.33 "cshifts" in the Northern direction or 1.67 "cshifts" in the Southern for a resolution of 8 x 64. This appears to stabilize at 2 "cshifts" in either direction at higher resolutions as seen from Table 10. This is close to the assumption that the winds generate mostly nearest neighbor communication.

The *ratio of calculation to communication* (R_o or R_a) (Tables 13,14) falls with increasing resolution because of two factors, namely :

- High latency in the communciation overhead. As observed earlier, it takes almost the same time to send and receive a message of 256 bytes as a message of size 64 bytes.

- The decreasing total number of cshifts TI. The total number of "cshifts" falls from 5.0 to 4.

It is obvious that this algorithm is dominated by overheads other than communication at high resolutions, which will be established shortly [19].

It is also noticed since the function calling overhead is high, the ratio R_a is less than R_o. Consequently, the *speedup* S_a is lower than S_o. However, S_o itself is much lower than ideal. At low resolutions, this is primarily due to the high communication overhead [20], but does not hold true at higher resolutions. Hence, a complete picture of speedup will not emerge without a detailed discussion at the functional level of the code. [21]. We will first tabulate the functions in order of increasing speedup for the resolution of 8x64 from the data in Tables 13 to 19 (Table 21) and then analyze the reasons for the performance. We will do this similarly for a higher resolution of 128x256 (Table 22).

[19] Communication overhead is only 2.64 % at a resolution of 128x256.

[20] At a resolution of 8 x 64, the ratio R_o is as low as 3.62, whereas at 128x256 R_o reaches a high of 37.95.

[21] Another notable aspect is that the overall speedups for the Bilinear Interpolation is lower than that for Bicubic Interpolation. This is because the actual interpolation routine for the Bilinear Interpolation involves only 6 floating-point multiplications compared to 36 for the Bicubic Interpolation.

Table 21 : Functional Breakup of Timings at a Resolution of 8 x 64:

Function	Speedup	%age of $T_{ca_a} + T_{co}$	%age of T_{ot}
Creation of Segment (cs)	1.50	65.07 %	34.95 %
Buffer Copying in above (bc)	3.74	26.06 %	14.00 %
Conversion to Local Indices (li)	7.96	4.96 %	2.67 %
Degree of Circulation (dc)	17.21	11.88 %	6.38 %
Computation alone in above (dcl)	19.40	10.64 %	5.7 %
Latitude, Longitude Index Calculations (ll)	24.09	10.82 %	5.8 %
Interpolation (int)	32.59	7.27 %	3.90 %

The "creation of segment" function shows a very low speedup primarily because it involves node-to-node communication with high latency. As discussed in the case of the Bicubic Interpolation, this does not account for all the loss in speedup. There is some load imbalance in the memory to memory copying, especially at the low resolutions, due to nodes at the poles having to deal with a larger partition (8x5 instead of 8x2) as shown in Figure 2. These nodes also have to handle with the pole-wrap problem which are absent in other nodes, leading to an additional load imbalance. The load imbalance is apparent from the fact that at the next higher resolution (64x64), though the grid points have increased by 8, the amount of time spent in buffer copying increases by 4. Otherwise the buffer copying scales with number of grid points. In the single node we find that the time spent in buffer copying scales well with the number of grid points.

The speedup of buffer copying is also low, as there are more places in the 32-node run where there is buffer copying. Taking into account the implementation and the sizes of buffers involved in both the runs, the single node run performs only about 6 times the amount [22] of buffer copying as the 32 node run. This is slightly more than the speedup of 3.66, but this is expected due to load imbalance. As the buffer copying alone takes 26.06 % of running time (without function calling overhead), this has a significant effect on the overall speedup [23] .

The speedup of "conversion to local indices" is also low, like the Bicubic Interpolation. This could be due to an error in measurement of a very small interval of time (0.28 ms) in the 32-node run. There is also the possibility that due to a load imbalance that affects the nodes corresponding to the polar regions which have to translate indices that cross the poles, that the speedup is low. But, the impact of this function is slight as it constitutes only 4.96 % of the running time (which is the least of all the functions).

The "degree of circulation" has a moderate speedup of 17.21. When the global maximum operation is left out, the speedup is about 19.40, which is medium. As the global maximum operation is a constant overhead, its effect at higher resolutions becomes negligible. As the code involves testing of conditions and as the quantities involved are

[22] Not to be confused with number of places

[23] Inclusive of the communication cost, it takes the lion's share of the running time at 65.07 %.

dynamically varying with both time and space, it is difficult to pinpoint how the speedup is affected. Additionally this function has no significance for the single node run.

The "calculation of latitude, longitude indices" shows a good speedup of 24.09 which involves a mix of integer and floating point arithmetic.

The actual "bilinear interpolation" shows more than ideal speedup. The timing of this routine has the same drawback as the "reduction of address calculations". As the time intervals involved are very small, the measurements are prone to errors. However, it may be concluded that this function has a good speedup even after discounting for any errors.

The overall speedup S_a is therefore poor because functions which have poor speedups (< 16) constitute 70.02 % of running time (with no function calling overhead), those with medium speedup ($16 \leq speedup \leq 23$) constitute 11.88 % and those with good speedups 18.1 % (> 23) [24].

Table 22 : Functional Breakup of Timings at a Resolution of 128x256:

Function	Speedup	%age of $T_{ca_a} + T_{co}$	%age of T_{ot}
Creation of Segment (cs)	8.99	30.89 %	13.72 %
Buffer Copying in above (bc)	12.02	23.09 %	10.25 %
Conversion to Local Indices (li)	20.02	5.09 %	2.20 %
Degree of Circulation (dc)	21.55	24.17 %	10.73 %
Computation alone in above (dcl)	21.63	24.08 %	10.70 %
Latitude, Longitude Index Calculations (ll)	32.60	21.09 %	9.37 %
Interpolation (int)	32.81	18.02 %	8.01 %

As discussed in the case of the Bicubic Interpolation, at this resolution, the jumps in speedups compared to the 128x128 resolution is due to cache misses. [25]

Using the same reasoning as in the case of 8x64 resolution, the single-node run has about 6 times the amount of buffer copying as the 32 node-run for the "creation of segment" function. However, the speedup of buffer copying is 12.04 which is two times more than expected. This is most likely due to caching effects. It can also be seen that there is an increase of 1.7 times in T_{bc} compared with a two-fold increase in the number of grid points from a resolution of 128x128. This indicates that the load imbalance due to unven partition size and pole wrapping is becoming less severe compared to a resolution of 8x64.

The node-to-node communication is still expensive as the speedup inclusive of this cost falls to 8.99 from 12.02. As the "creation of segment" function still takes 30.89 % (13.72 % with function calling overhead) of the running time, it affects the overall speedup adversely.

[24] The classification of poor, medium and good speedups may seem arbitrary. But this distinction serves the purpose considering that timings were obtained on a 32 node machine.

[25] It was not possible to run at a resolution of 256x256 on the single node due to lack of memory.

The speedup of "conversion to local indices" increases to 20.02. This is certainly due to cache misses, because at a resolution of 128x128 the speedup is only 13.76. At the 128x128 resolution the accuracy of the timing measurements should be good as there are 512 iterations taking place. The speedup is not better than 20.0 because of the load imbalance problem at the nodes corresponding to the poles.

The "degree of circulation" has a fairly high speedup of 21.55 The global maximum operation now constitutes a negligible amount of the time spent in the function, as seen in the difference in speedups S_{dcl} and S_{dc}.

The "calculation of latitude, longitude indices" shows more than ideal speedup of 32.60. This is due to errors in measurement of timings.

The actual "bicubic interpolation" as before shows more than ideal speedup due to the drawback discussed earlier.

Thus the overall speedup S_o or S_a is better than at a resolution of 8x64. The speedup is only medium because functions which take major part of running time (60.15 %) have poor or medium speedups (≤ 23). About 30.89 % of the code has poor speedup (< 16) , another 29.26 % has medium speedup ($16 \leq speedup \leq 23$) and the balance 39.85 % the rest has a good speedup (> 23).

An *accuracy* of upto (but not including) the fifth decimal place was obtained between the parallel and serial implementations.

Table 20 confirms the *accuracy of the node-to-node communication overhead* . The results from the secondary experiment, $T_{co_{ne}}$ and T_{co_n} match closely.

6 Conclusions

A reasonable speedup maybe achieved by the message passing version. This problem is hard to parallelize because of the following reasons. At low resolutions, the speedup is poor because of very high communication overhead [26], some load imbalance, and a fair amount of book-keeping and memory management involved. At higher resolutions the amount of book-keeping and memory management dominate the communication [27] and load imbalance overheads. But, unfortunately, these overheads are required for the working of the parallel algorithm and are difficult to reduce. Hence, there is only a medium speedup. If the algorithm were to be ported to a machine with a low latency communication network, the book keeping and memory managment overheads will still result in only a medium speedup. However, there is scope for improving the overall running time by reducing the function call overhead.

[26] For the bicubic interpolation, T_{co} takes 57 % as much time as T_{ca_a}. For the bilinear interpolation, T_{co} takes 68 % as much time as T_{ca_a}, which is even worse.

[27] T_{co} is now only about 30 % of T_{bc} for either the bicubic or the bilinear interpolations. T_{bc} itself takes about 14 % and 24.9 % of T_{ca_a} for the bicubic and bilinear interpolations respectively.

7 Further Work

In the short term further testing needs to be done to verify the accuracy and stability of the parallel code. The scaling of the performance of the parallel code with the number of processors needs to be observed.

In the long term more, more scalable decompositions such as the igloo decomposition have to be studied. A different approach will be needed. The "cshift" operation will no longer be applicable. A key strategy to make the igloo decomposition feasible would be to find a near-optimal ratio of the partition dimensions. This is has to be ascertained by initial experiments.

References

[1] Thinking Machines Co., 1993: CM Fortran Reference Guide **Ch. 12**

[2] Thinking Machines Co., 1993: CMMD User's Guide and Reference Manual

[3] McDonald, A.; Bates, J.R., 1989: Semi-lagrangian Integration of a Gridpoint Shallow Water Model on the Sphere. *J. American Meteor. Soc.*, **117**, 130-137

[4] Robert, A., 1981: A stable numerical integration scheme for the primitive meteorological equations. *Atmos. Ocean*, **19**, 35-46

[5] Robert, A., 1982: A semi-Lagrangian and semi-implicit numerical integration scheme for the primitive meteorological equations. *J. Meteor. Soc. Japan*, **60**, 319-324

[6] Staniforth, A.; Cot'e, J., 1991: Semi-Lagrangian Integration Schemes for Atmospheric Models - A Review. *Monthly Weather Review*, **119**, 2206-2203

8 Appendix A: Bicubic Interpolation

The bicubic interpolation is used where more accuracy is desired than the bilinear interpolation. [28] The bicubic interpolation used here is a two-step one-dimensional lagrange polynomial (cubic) interpolation.

The inputs to the bicubic interpolation consists of sixteen values arranged on a 3 cell x 3 cell grid as shown in figure 10. Using the sixteen values four cubic interpolations can be carried out using the four points available at each value of ii (in the North-South direction). Then another cubic interpolation can be performed on the four resultant values in the East-West direction (all of which reside on a straight line which is dely units away from the line y = jj + 3) to obtain the desired value.

Let $P(y) = a_0 + a_1 y + a_2 y^2 + a_3 y^3$ be a polynomial of third degree. The standard equation for the lagrange polynomial interpolation is given by,

$$P(y) = \sum_{i=0}^{3} f_i L_i(y) \tag{1}$$

where,

- y is the point at which interpolation takes place.
- P(y) represents the desired value.
- f_i are the known values on the grid at points y_i.
- $L_i(y)$ are the Lagrange polynomials such that $L_i(y) = \Pi^3_{k \neq i, k=0}(y - y_k)/(y_i - y_k)$

As shown in figure 10, considering the North-South direction. Let the position of point of interpoation be considered relative to the lower right corner of the *grid square* in which it is located. It is Δ_y units away from the lower right corner in North-South direction.

In our usage [figure 10], let

- y_i represent jj+2, jj+3, jj+1, jj+4, $0 \leq i \leq 3$
- f_i corresponds to the value of w_k at y_i, $i = (k-1)mod4$, $1 \leq k \leq 16$

As the grid points are evenly spaced, the computation of the Lagrange polynomials can be simplified as below,

$$L_0 = [(y - y_1)(y - y_2)(y - y_3)]/[(y_0 - y_1)(y_0 - y_2)(y_0 - y_3)]$$

$$L_0 = [(1-\Delta_y)(1+\Delta_y)(2-\Delta_y)]/(-1)(1)(-2) = 0.5*(2.0-\Delta_y)*(1.0-\Delta_y^2) = \Delta_{y2}*(1.0-\Delta_y^2)$$

[28]The bilinear interpolation is more economical than the bicubic interpolation and is hence used where more accuracy is not required.

where, $\Delta_{y2} = 0.5 * (2.0 - \Delta_y)$

Similarly,

$$L_1 = (\Delta_y^2 + \Delta_y) * \Delta_{y2}$$

$$L_2 = (\Delta_y^2 - \Delta_y) * \Delta_{y2}/3$$

$$L_3 = (\Delta_y^2 - 1.0) * \Delta_y/6$$

A similar reasoning holds for East-West direction. Let the subscript 'y' be used for quantities in the North-South direction and 'x' for East-West. The algorithm for the bicubic interpolation is as follows.

- Inputs
  - Sixteen values at the grid points as in figure 10
  - Relative positions Δ_x, Δ_y
- Compute terms Δ_{y2} and Δ_y^2 - 2 multiplications
- Compute terms Δ_{x2} and Δ_x^2 - 2 multiplications
- Compute Lagrange polynomials L_{iy} - 6 multiplications
- Compute Lagrange polynomials L_{ix} - 6 multiplications
- Perform four lagrange interpolations in North-South direction - 4 x 4 multiplications
- Perform the final lagrange interpolation in East-West direction - 4 multiplications. This obtains the desired value in a total of 36 multiplications.
- Return the desired value

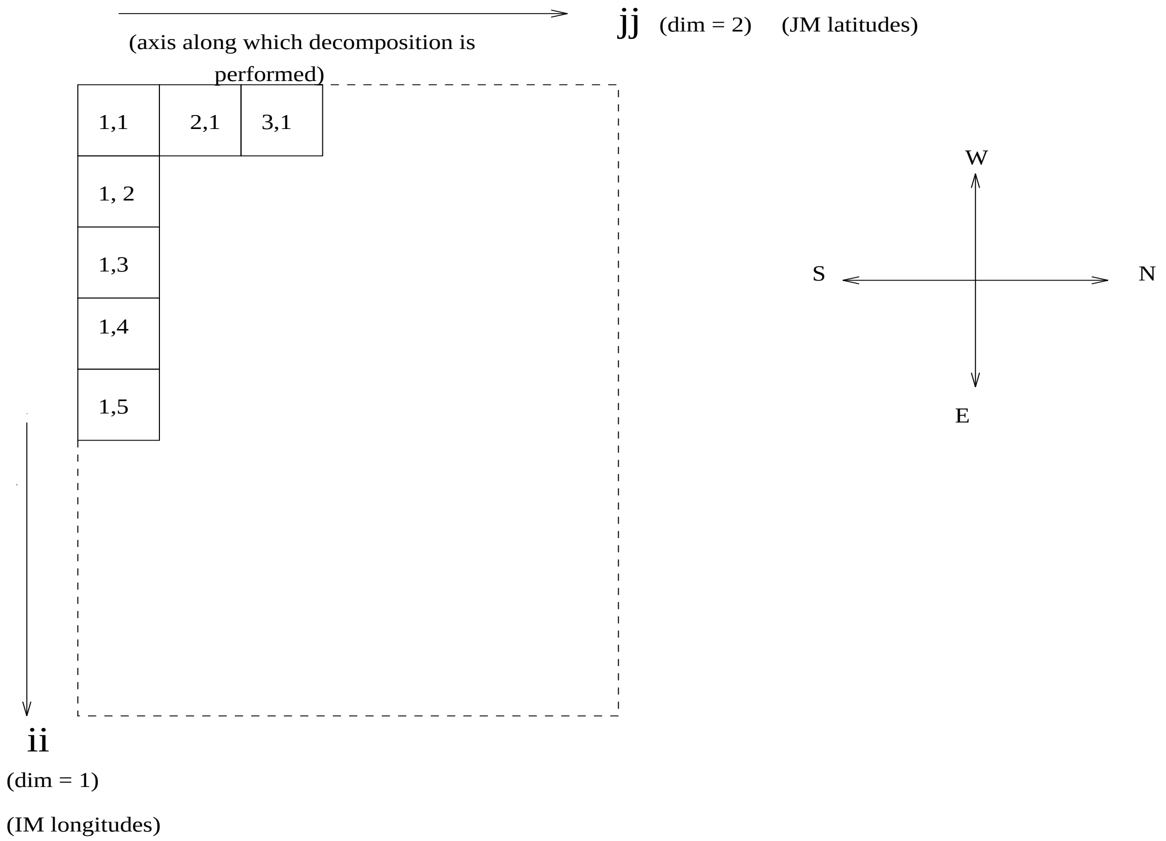


Fig. 1 Original Grid : Dimensions And Orientation. Also shown is the axis along which lasagnia decomposition is performed.

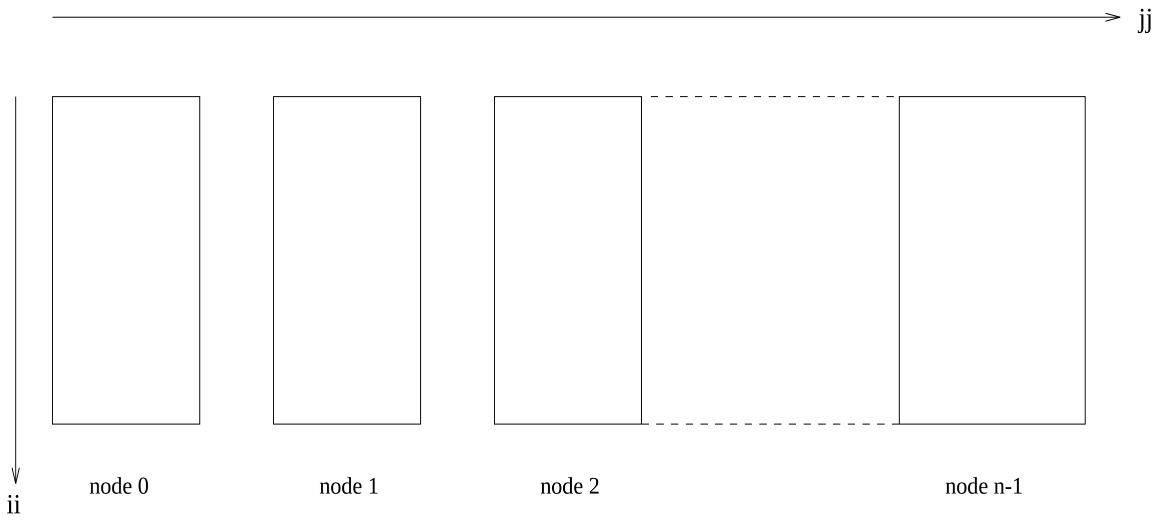
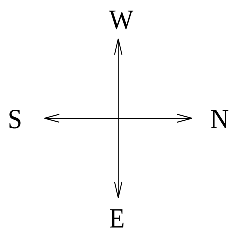


Fig. 2 Decomposed grid. Each node has a partition of the original data. As the original grid had dimensions which were not a power of 2, the last node picks up a larger partition than the rest.



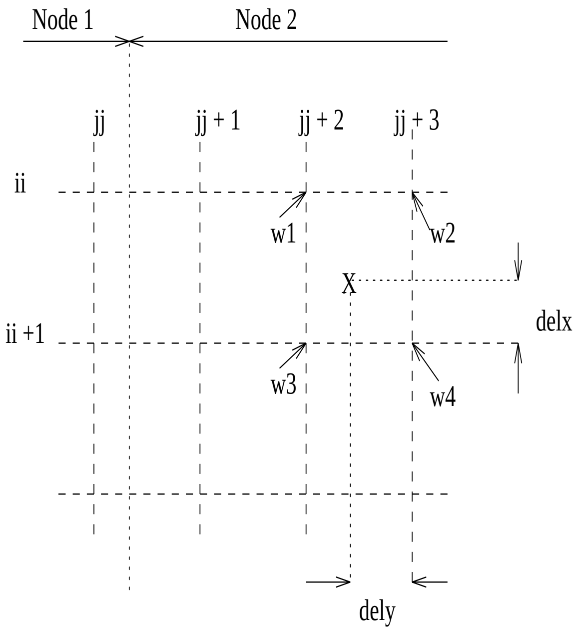


Fig. 3 Bilinear Interpolation. The 'X' marks the point which is found on interpolation. The figure also shows the case where the value at jj is resident on a node (node 1) but values at jj + 1 onwards are off-node.

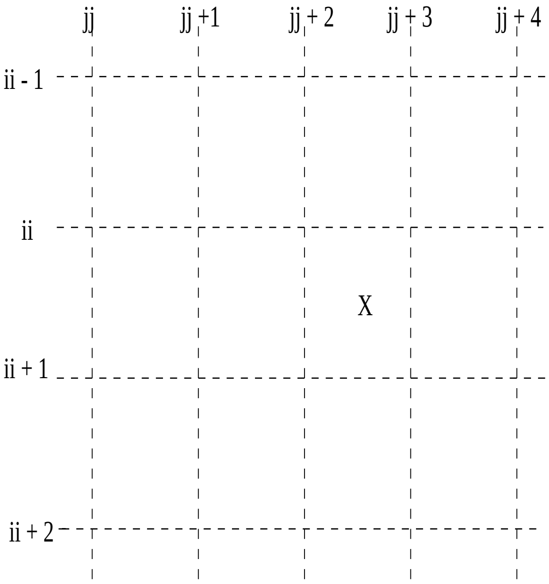


Fig. 4 Bicubic Interpolation. The 'X' marks the point which is found on interpolation.

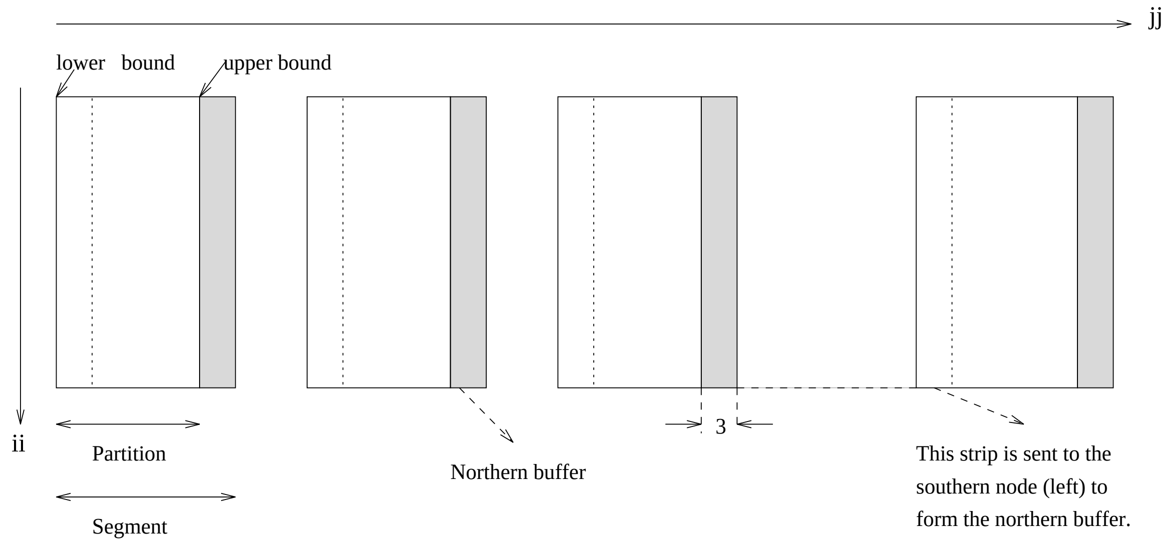
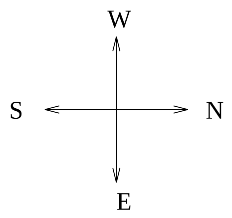


Fig. 6. A fixed width (3 elements wide for bilinear interpolation) is constructed on every node that overlaps with the northern neighbor.



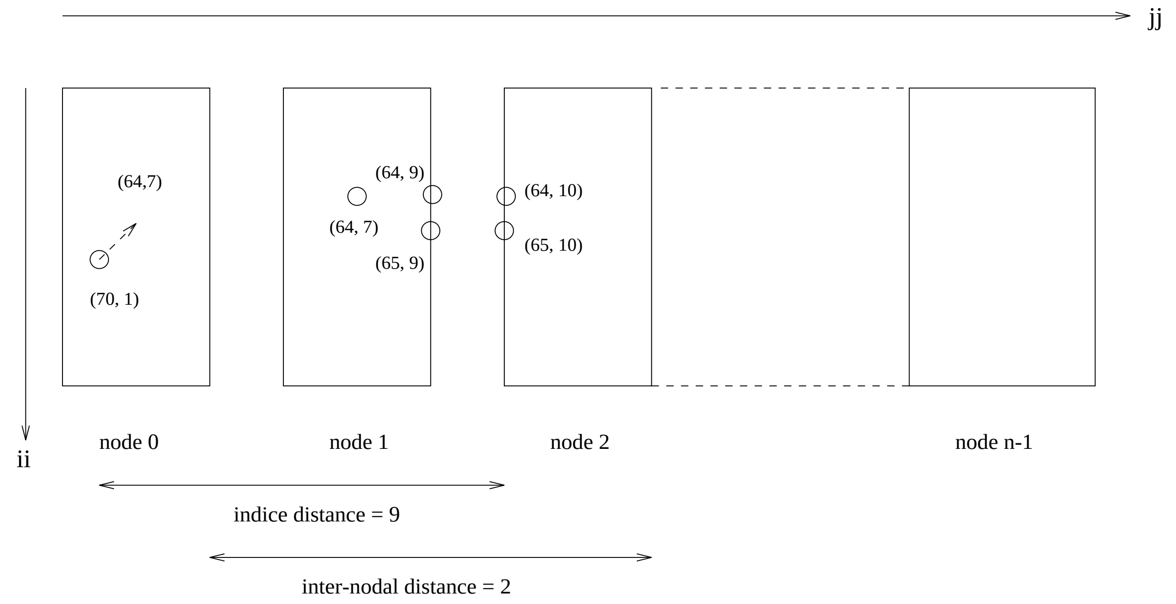
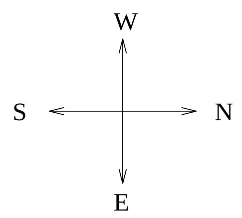


Fig. 5 The figure shows how the inter-nodal distance is computed. In node 0, the global indice jj has the value 64 at location (70,1) and global indice ii has the value 7. To be able to perform the interpolation the farthest indice needed is jj = 10 (jj + 3) in the Northern direction for the bilinear interpolation. However, if jj were such that the points were more than half-way around the globe, the southern most point jj + 2 is furthest. For simplicity we use jj instead of jj + 2 in our calculations. These two are the most extreme points for the bilinear interpolation and are used in the calculation of the maximum inter-nodal distance for each jj.



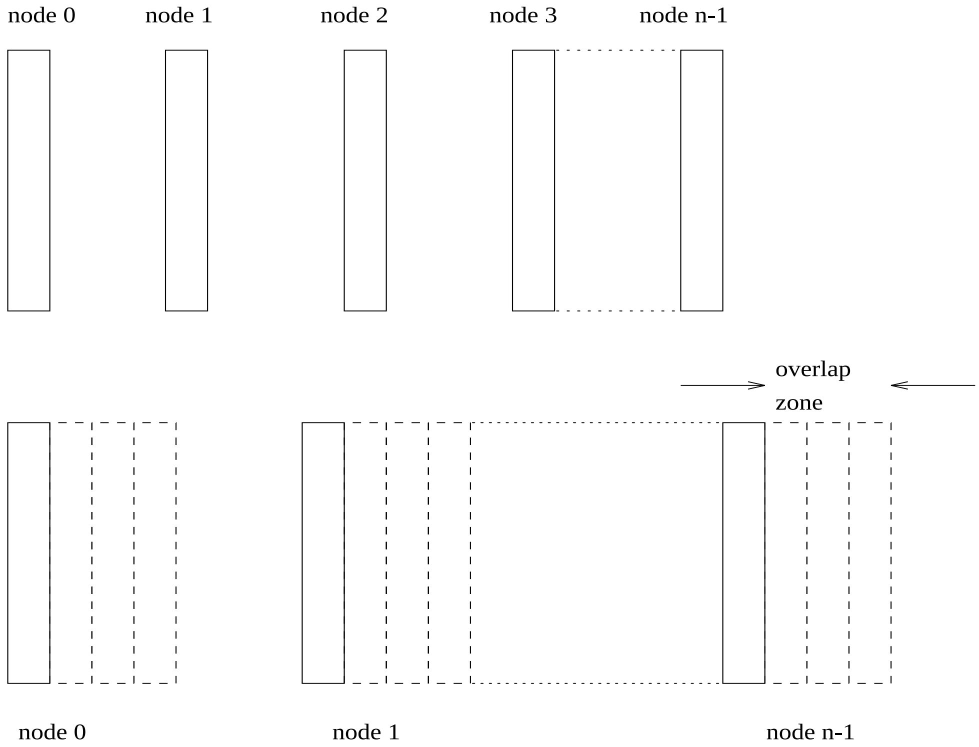


Fig. 7 The top half of figure shows a case where an individual partition is smaller than the overlap zone. In this case two or more communication steps have to be carried out only for establishing the overlap zone as shown in the bottom half of the figure. This maybe avoided as subsequent communication steps fetch data which include the data in the overlap zone. An improved algorithm is explained in the next figure.

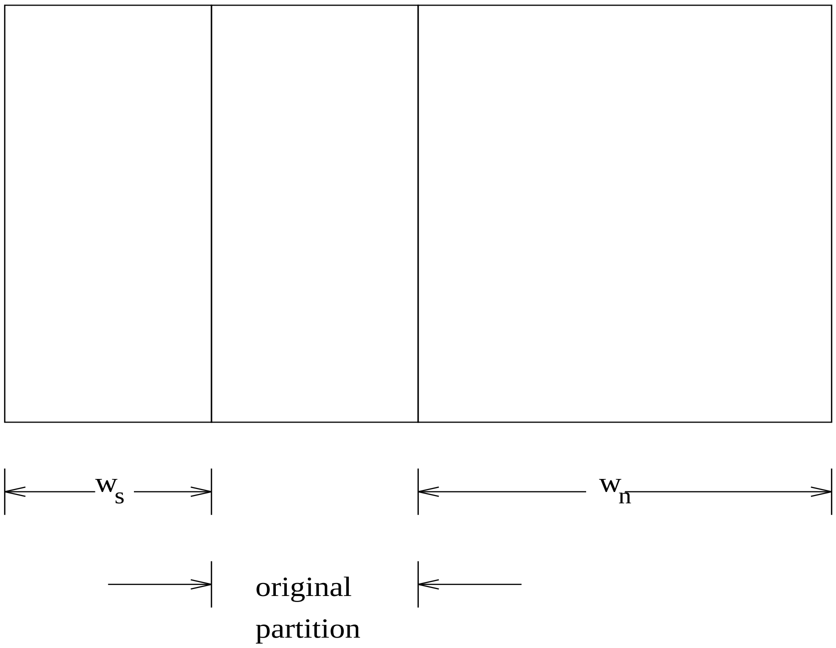


Fig. 8 The figure shows the resulting block in a node after all communication steps. There is no northern buffer created, however. Each node adds on a strip of width w (with the suffix s for southern and n for northern) in the North-South directions, depending on how far along the fluid trajectory one has to traverse to be able to perform an interpolation. An individual strip width is the same as that of a partition, i.e. w (s or n) is a multiple of the partition width.

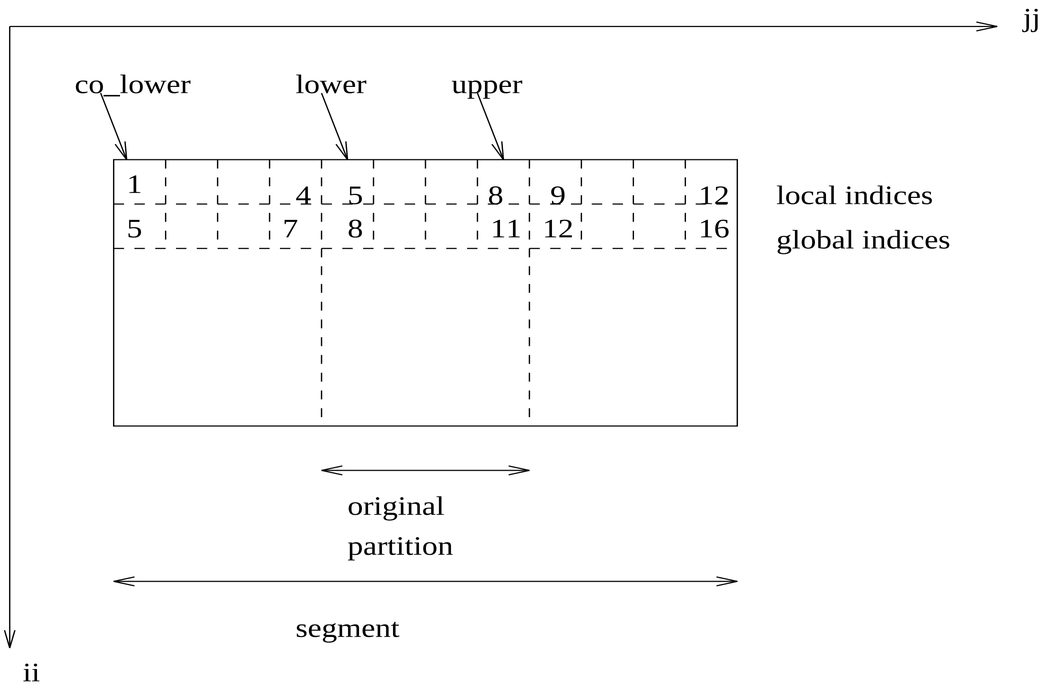


Fig 9 a. This figure shows a node which has no wrap around after the segment has been fully reconstructed in either north or south directions. Global indices translate to local indices in the North - South direction according to the formula

Local index = Global Index - co_lower + 1

(10, 7) $\equiv$ (10, 7 - 5 + 1) = (10, 3)

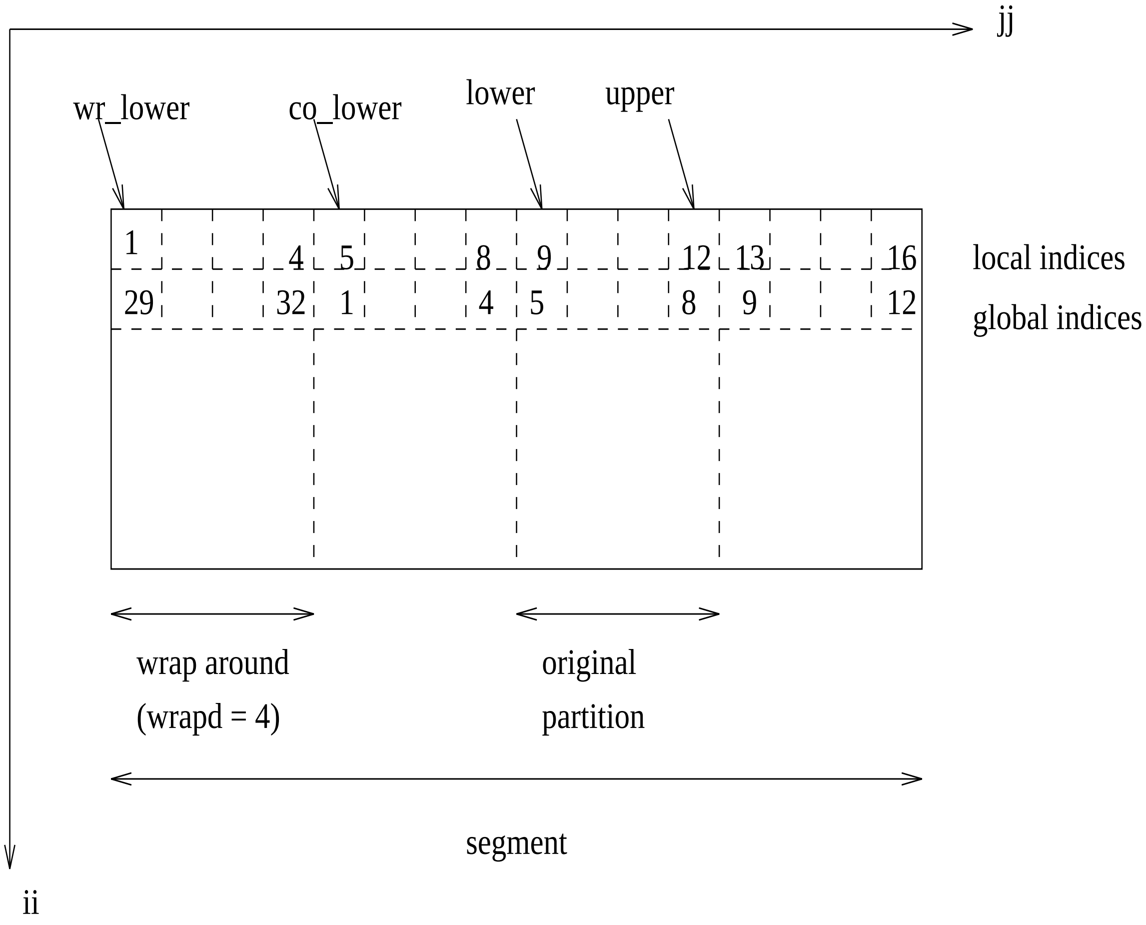


Fig 9 b. This figure shows a node which has a wrap around in the southern direction after the segment was fully reconstructed. Global indices translate to local indices in one of two ways.

Case a) The global index refers to a value in the wrapped part .

Local Index = Global Index - wr_lower + 1

$$(10, 30) \equiv (10, 30 - 29 + 1) = (10, 2)$$

Case b) The global index refers to a value in the non-wrapped part. .

Local Index = Global Index + wrapd

$$(10, 9) \equiv (10, 9 + 4) = (10, 13)$$

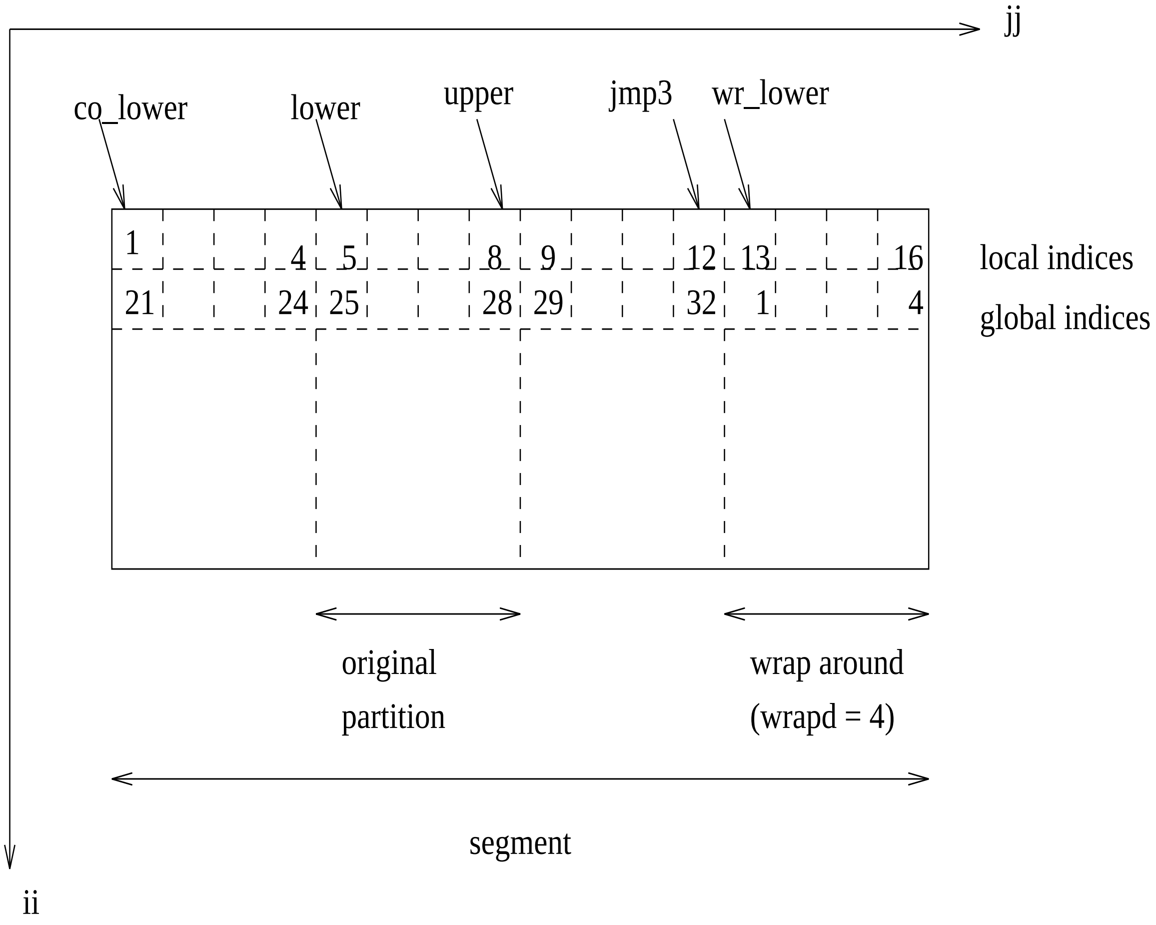


Fig 9 c. This figure shows a node which has a wrap around in the northern direction after the segment was fully reconstructed. Global indices translate to local indices in one of two ways.

Case a) The global index refers to a value in the wrapped part .

Local Index = Global Index + jmp3 - co_lower + 1

$$(10, 3) \equiv (10, 3 + 32 - 21 + 1) = (10, 15)$$

Case b) The global index refers to a value in the non-wrapped part. .

Local Index = Global Index + co_lower + 1

$$(10, 21) \equiv (10, 21 - 21 + 1) = (10, 1)$$

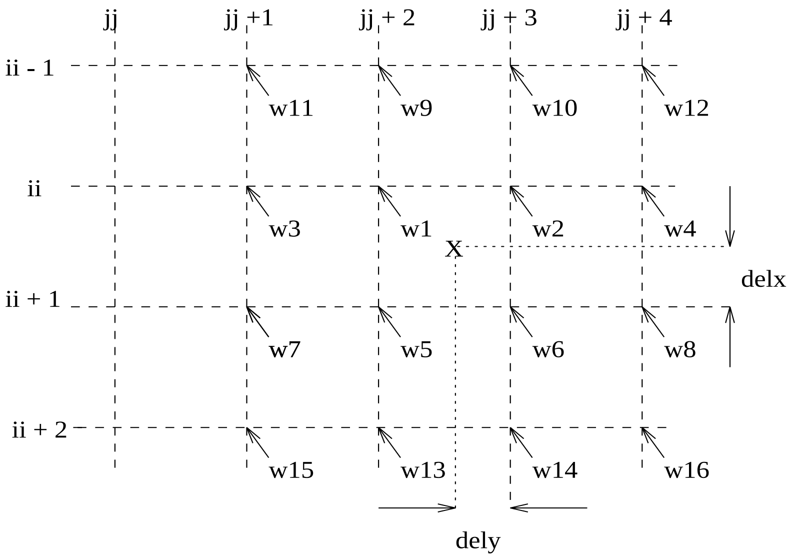


Fig. 10 Bicubic Interpolation. The 'X' marks the point which is found on interpolation.