1 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 [1]. 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}.

[1] 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 $T_{ca_a} = T_{ll} + T_{dcl} + T_{bc} + T_{li} + [T_{ra}] + T_{int}$ [2]

- 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) $S_o = T_{gt_s}/T_{gt}$.

[2] 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

1.1 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 [3] . 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	—–	—–

[3]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 [4].

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) 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 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 [5], 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).

[4] 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.

[5] 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* [6]. Taking into account the implementation and the sizes of buffers involved in both the runs, the single node run performs only about 4 times the amount [7] of buffer copying as the 32 node run. This agrees closely with the speedup. As the buffer copying alone takes 19.79 % of running time (without function calling overhead), this has a significant effect on the overall speedup [8] .

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 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. [9]

[6] 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*.

[7] Not to be confused with number of places

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

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

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) [10].

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 %

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 [11].

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.

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"

[10] 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.

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

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.

1.2 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 [12] . 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

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

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	—–	—–

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) 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 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 [13], 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. [14]. 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).

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. 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 4 times the amount [15] of buffer copying as the 32 node run. This agrees closely with the speedup of 3.66. As the buffer copying alone takes 26.06 % of running time (without function calling overhead), this has a significant effect on the overall speedup [16] .

[13]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.

[14]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.

[15]Not to be confused with number of places

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

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 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) [17].

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. [18]

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

[17] 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.

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

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.

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.

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.