Modelling Flow-Dependent Covariances with Gaussian Integrals

by Carlos Geijo Guerrero (AEMET)

ACCORD All Staff Workshop ; Zalakaros ; April 2025

- WP DA7.1 in the current ACCORD RWP2025 (source and scientific note)
- Idea revisited from previous work (Aladin-Hirlam Newsletters#13, August 2019)
- Modulation of the B covariance matrix of a random field Δ by an "ambient" field $\vec{V}$
  How ? Starting point: the distribution moments generating function with a coupling term

$$Z(S) = \int d\Delta_1 \ldots d\Delta_N \, \mathrm{e}^{-\frac{1}{2}\left(\Delta^T B^{-1} \Delta + \mu \, tr\left[\left(\mathbf{V} \bullet \vec{\nabla} \Delta \right) \left(\mathbf{V} \bullet \vec{\nabla} \Delta \right)^T \right] \right) + \Delta^T S}$$

- In the previous work, computations were done in x-space. Now we try in k-space

$$Z(S)=\int \prod_{k=1\ldots N} d\Delta_k \, e^{-\frac{1}{2}\left(\sum_{k,l}\Delta_k^*\left[B_{k,l}^{-1}\delta_{k,l}+\mu M_{k,l}\right]\Delta_l\right)+\sum_k S_k^*\Delta_k}$$

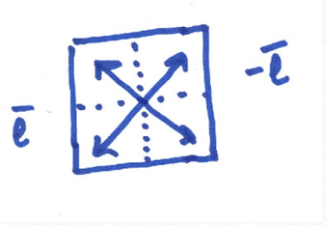
$$M_{k,l}\equiv M\left(\vec{k},\vec{l}\right)=\frac{1}{N}\sum_{\vec{G}}\left(\vec{s}_{\vec{k}}\,\vec{V}^*_{\vec{G}-\vec{k}}\right)\left(\vec{s}_{\vec{l}}\,\vec{V}_{\vec{G}-\vec{l}}\right) \quad ; \quad \vec{k}=\frac{2\pi}{\sqrt{N}}\left(k_x,k_y\right) \quad ; \quad \vec{s}_{\vec{k}}=\begin{pmatrix}\sin(\frac{2\pi}{\sqrt{N}}k_x)\\ \sin(\frac{2\pi}{\sqrt{N}}k_y)\end{pmatrix}$$

- M has several important symmetries

a) Positive definite $\quad \sum_{\vec{l}\,\vec{k}}\Delta^*_{\vec{l}}\; M(\vec{l},\vec{k})\,\Delta_{\vec{k}}>0 \;;\; \forall\; \Delta_{\vec{k}}\neq 0$

b) Hermitian $\quad M\left(\vec{l}_1,\vec{l}_2\right)=M^*\left(\vec{l}_2,\vec{l}_1\right) \;;\; M=M^\dagger$

c) Reality $\quad M\left(-\vec{l}_1,-\vec{l}_2\right)=M^*\left(\vec{l}_1,\vec{l}_2\right) \quad where \quad -\vec{l}=\sqrt{N}-\vec{l}$



- All this allows us to easily calculate Z(S) $\quad Z(S) = \sqrt{(2\pi)^N \|C\|}\ \ e^{\frac{1}{2}(S^*)^T CS} \quad ; \quad C^{-1} = B^{-1} + \mu M$

- And then by standard methods calculate the covariances (***with the correct normalization***)

$$\langle \Delta_k \Delta_l \rangle = \left| \frac{\partial^2}{\partial S_{-k} \partial S_{-l}} \ln Z(S) \right|_{S=0} \quad ; \quad \ln Z(S) = \ln\left(\sqrt{(2\pi)^N \|C\|} \right) + \frac{1}{2} \sum_{k,l} S_{-k} C_{k,l}\ S_l$$

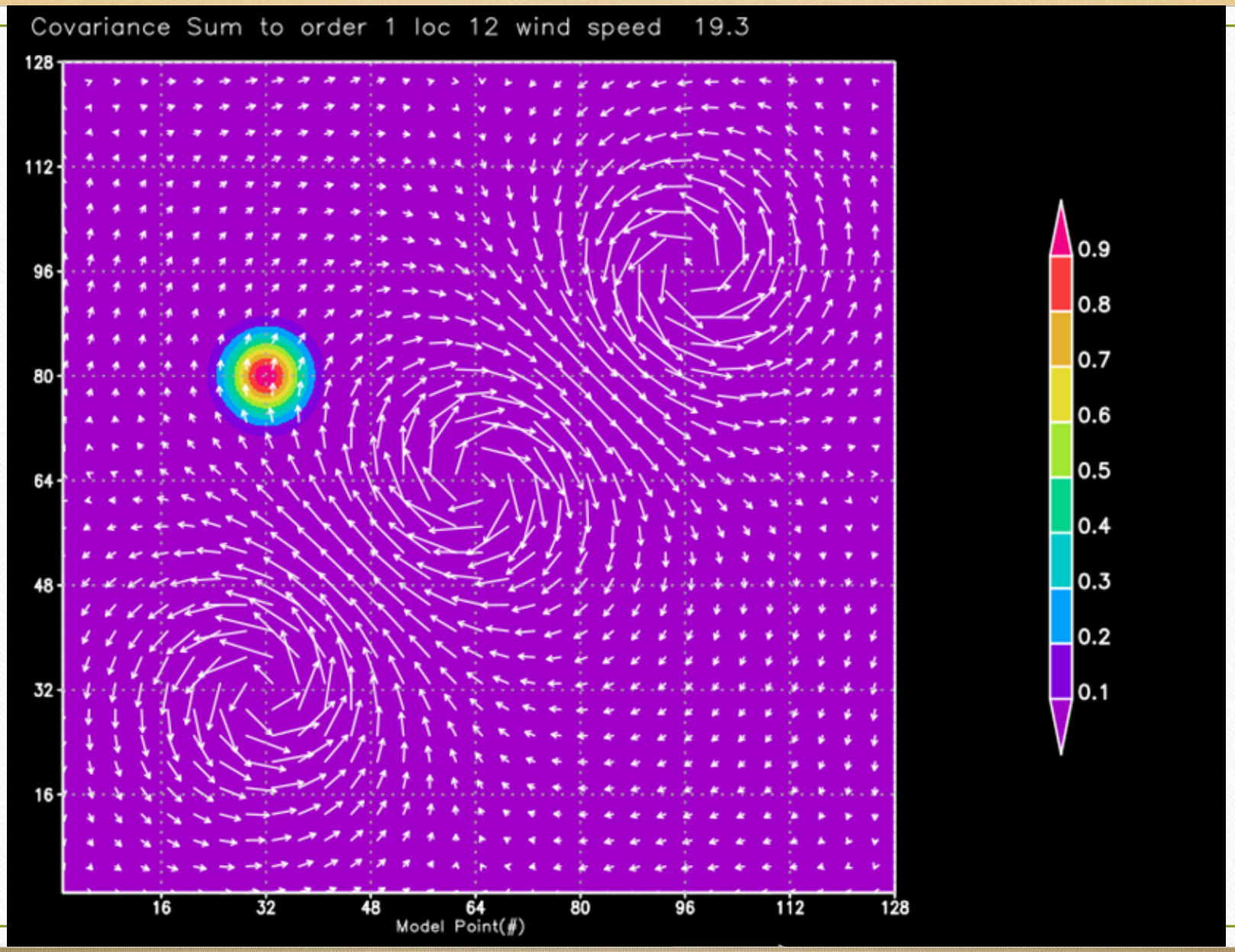
then

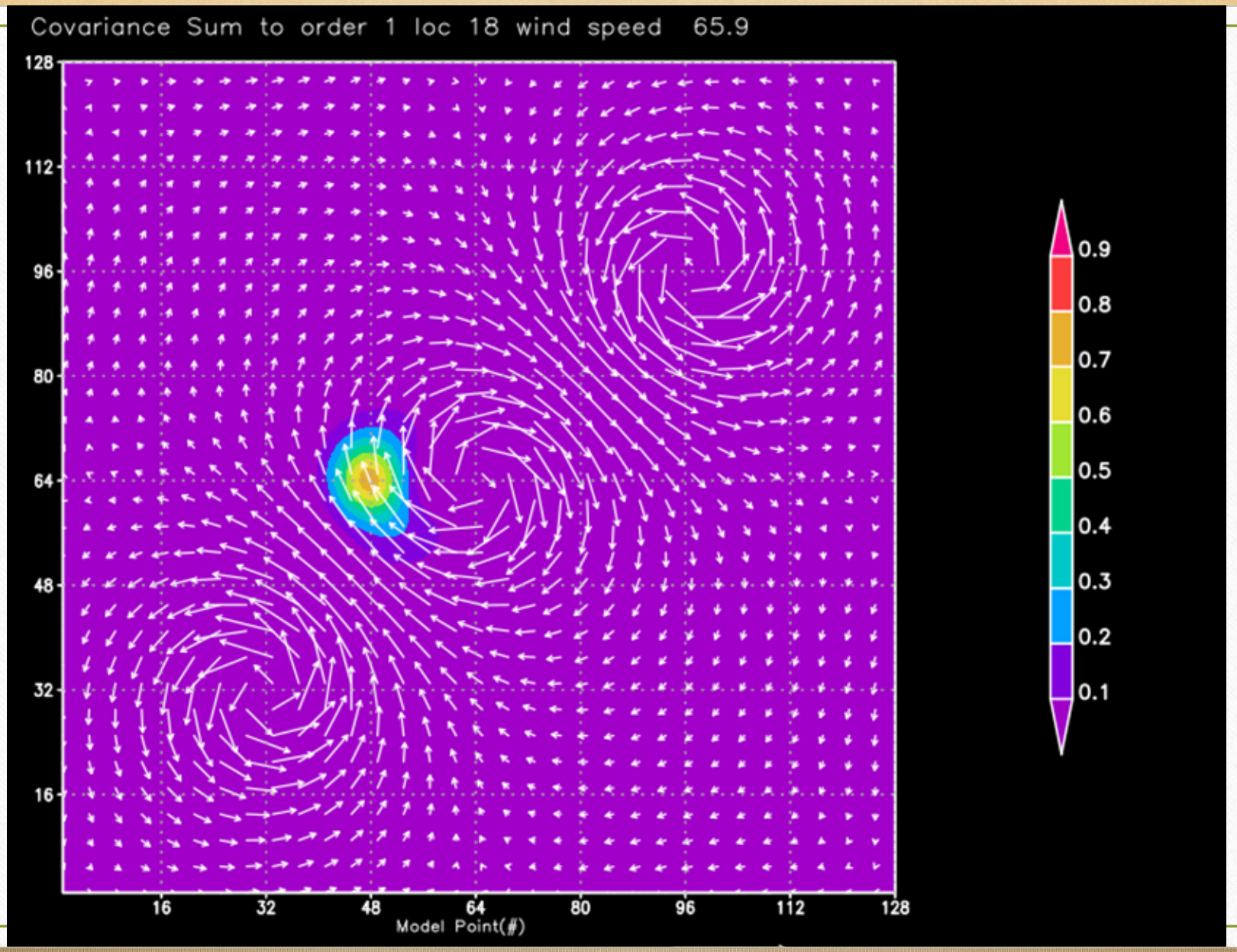
$$\langle \Delta_k \Delta_l \rangle = \frac{1}{2} \frac{\partial^2}{\partial S_{-k} \partial S_{-l}} \left(\sum_{i,j} S_{-i} C_{i,j} S_j \right) = \frac{1}{2} \left(C_{l,-k} + C_{k,-l} \right) = C_{l,-k}$$

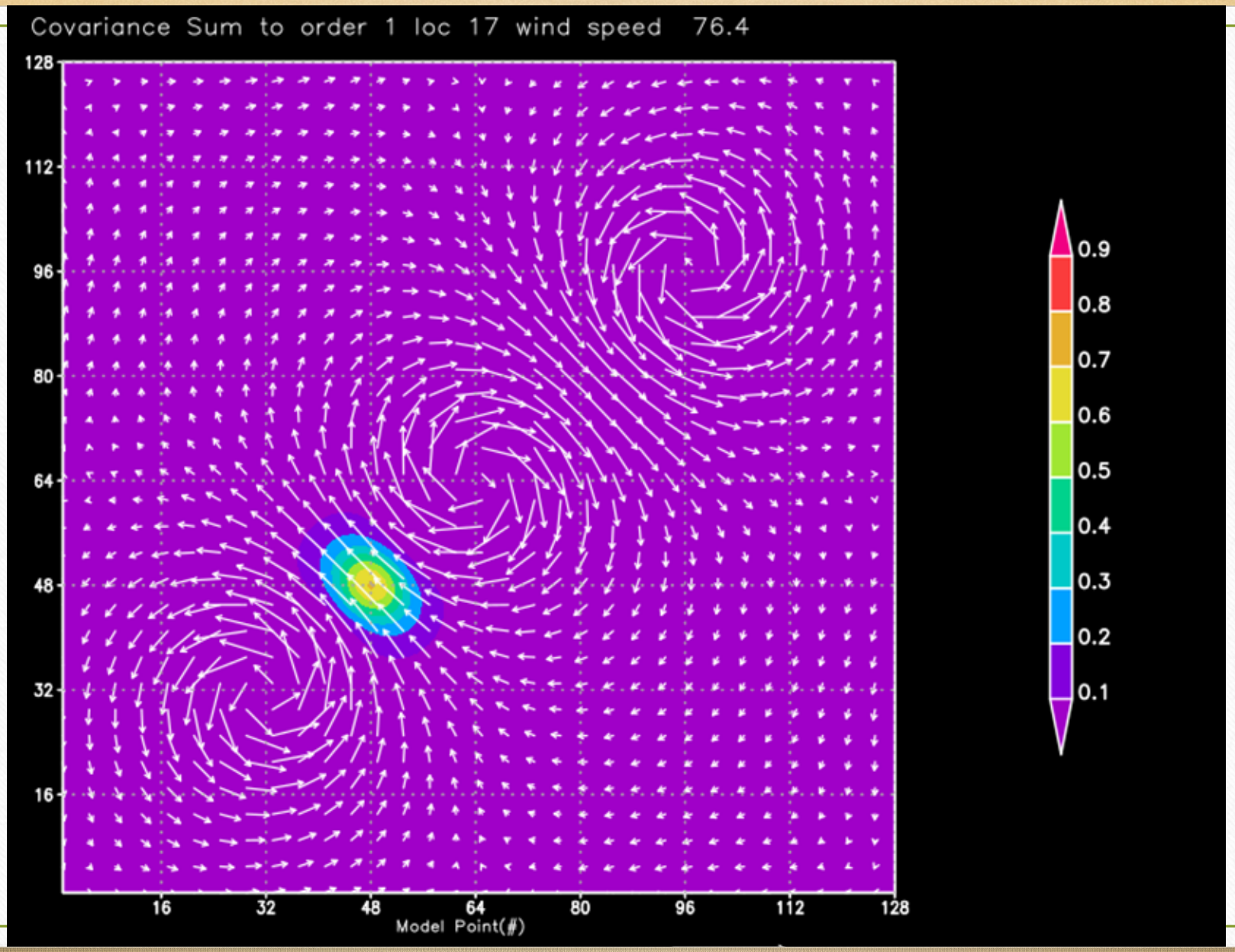
- It "only" remains to compute C . The proposed algorithm here is the "C.G. Neumann series"

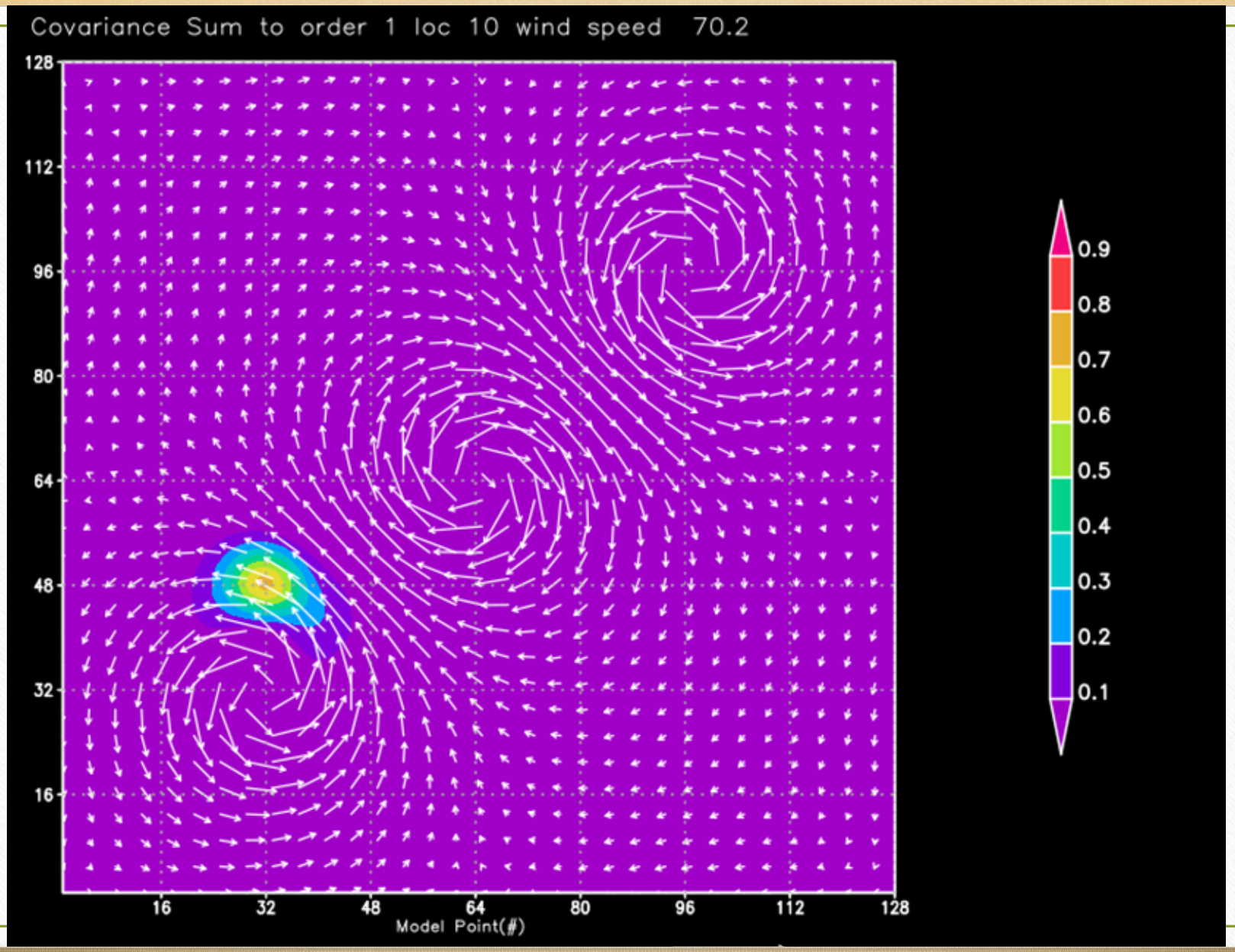
$$C = B\left(1+\mu\, MB\right)^{-1} = B\left(1-\mu\, MB+\mu^2 MBMB-\ldots\right) = \sum_{n=0}\left(-\mu\right)^n\, B(MB)^n$$

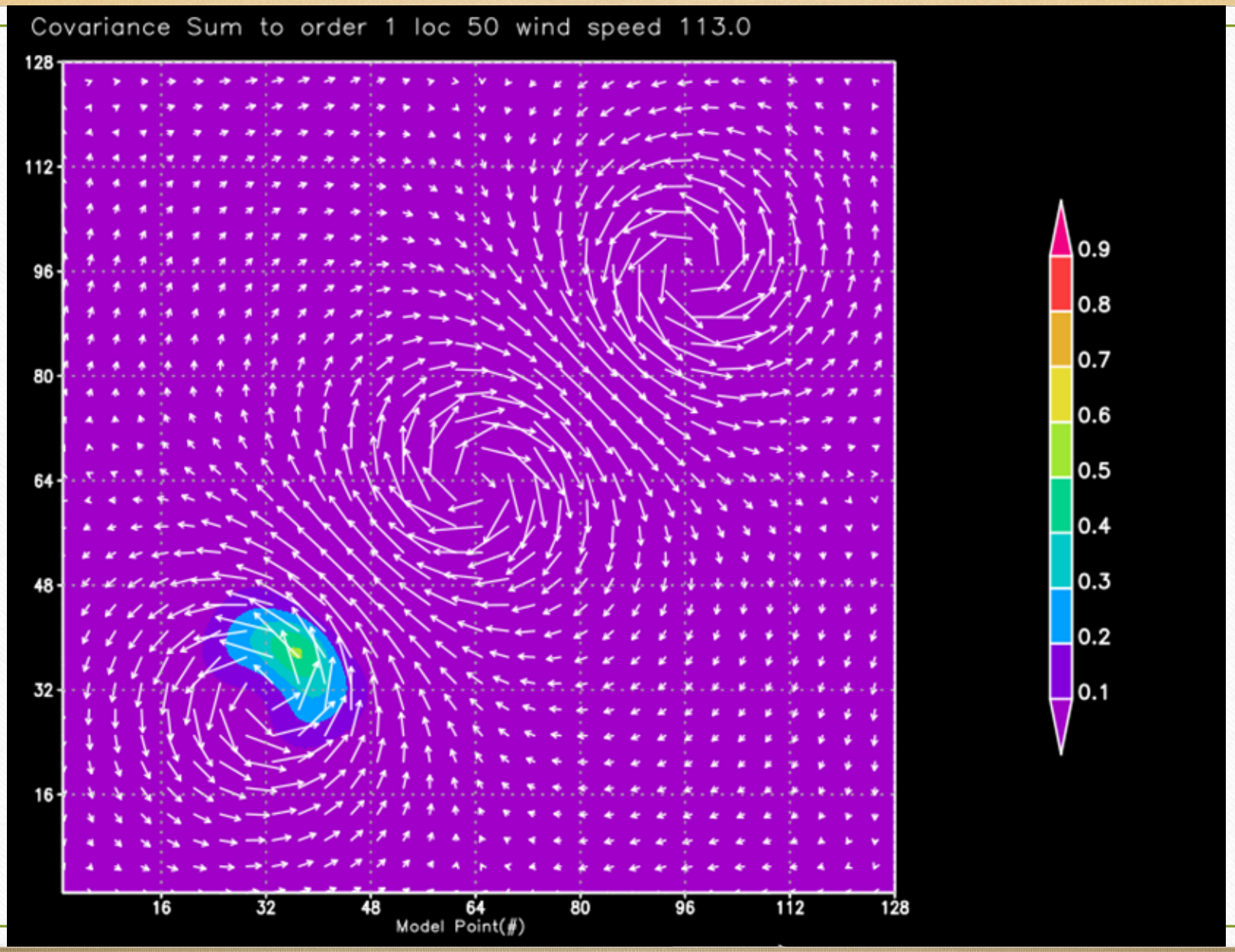
$$\langle \Delta_k \Delta_l \rangle = B_k \delta_{k,l} - \mu\, M_{k,-l}\, B_k B_l + \mu^2 \left(\sum_m B_m\, M_{k,m}\, M_{m,-l} \right) B_k B_l$$
$$- \mu^3 \left(\sum_{m,n} B_m\, B_n\, M_{k,m}\, M_{m,n}\, M_{n,-l} \right) B_k B_l + \ldots$$

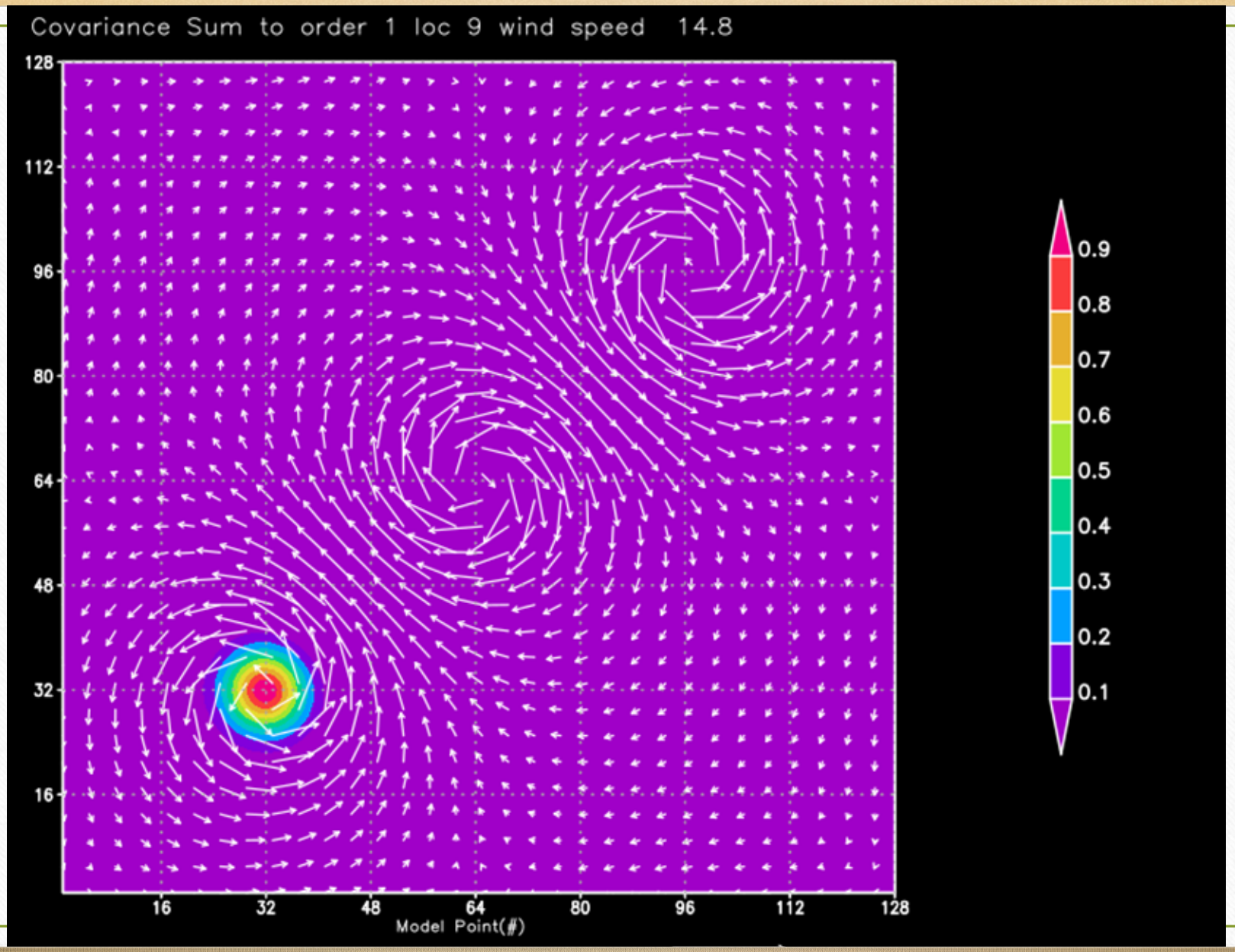

Covariance Sum to order 1 loc 12 wind speed 19.3
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

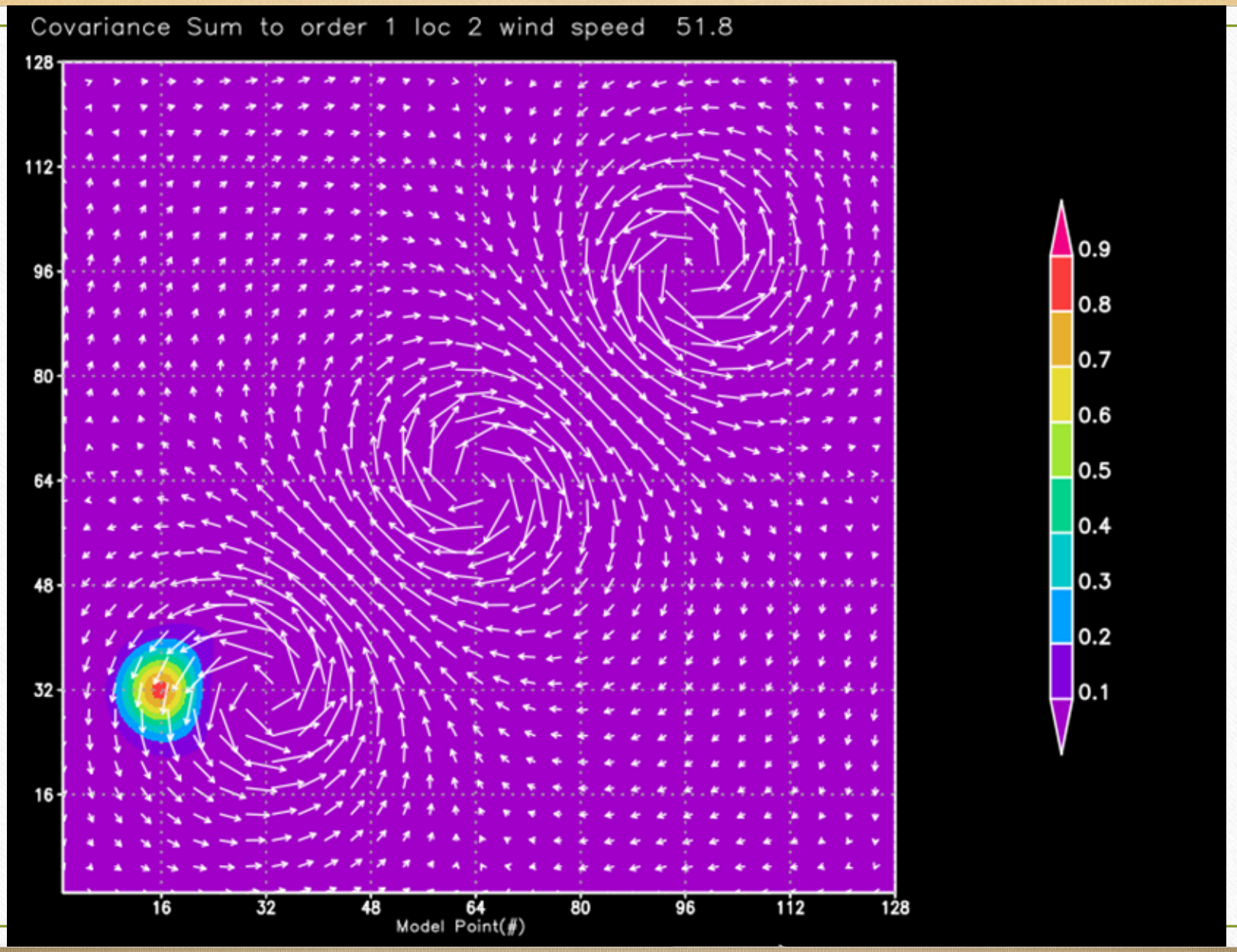

Covariance Sum to order 1 loc 18 wind speed 65.9
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

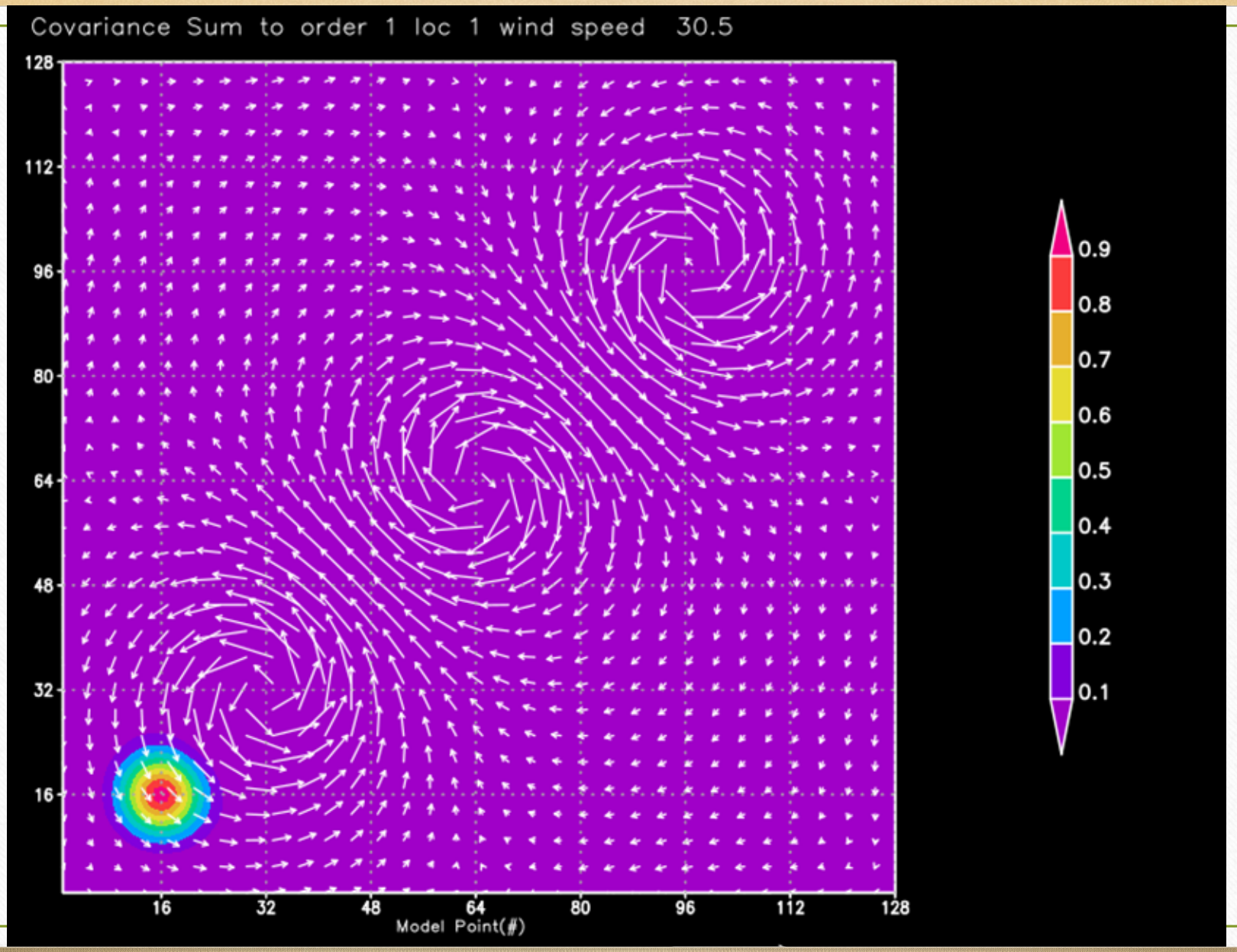

Covariance Sum to order 1 loc 17 wind speed 76.4
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

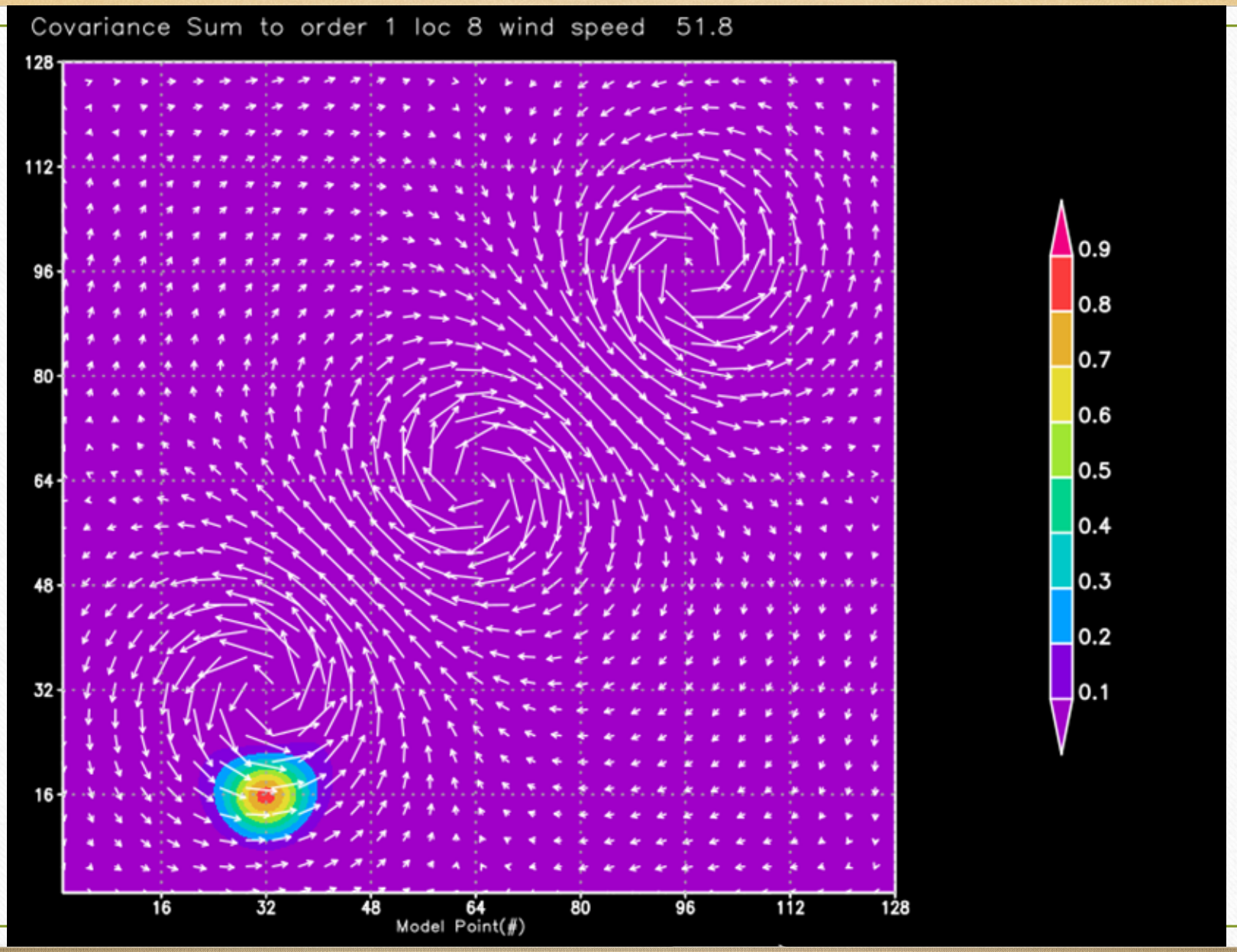

Covariance Sum to order 1 loc 10 wind speed 70.2
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

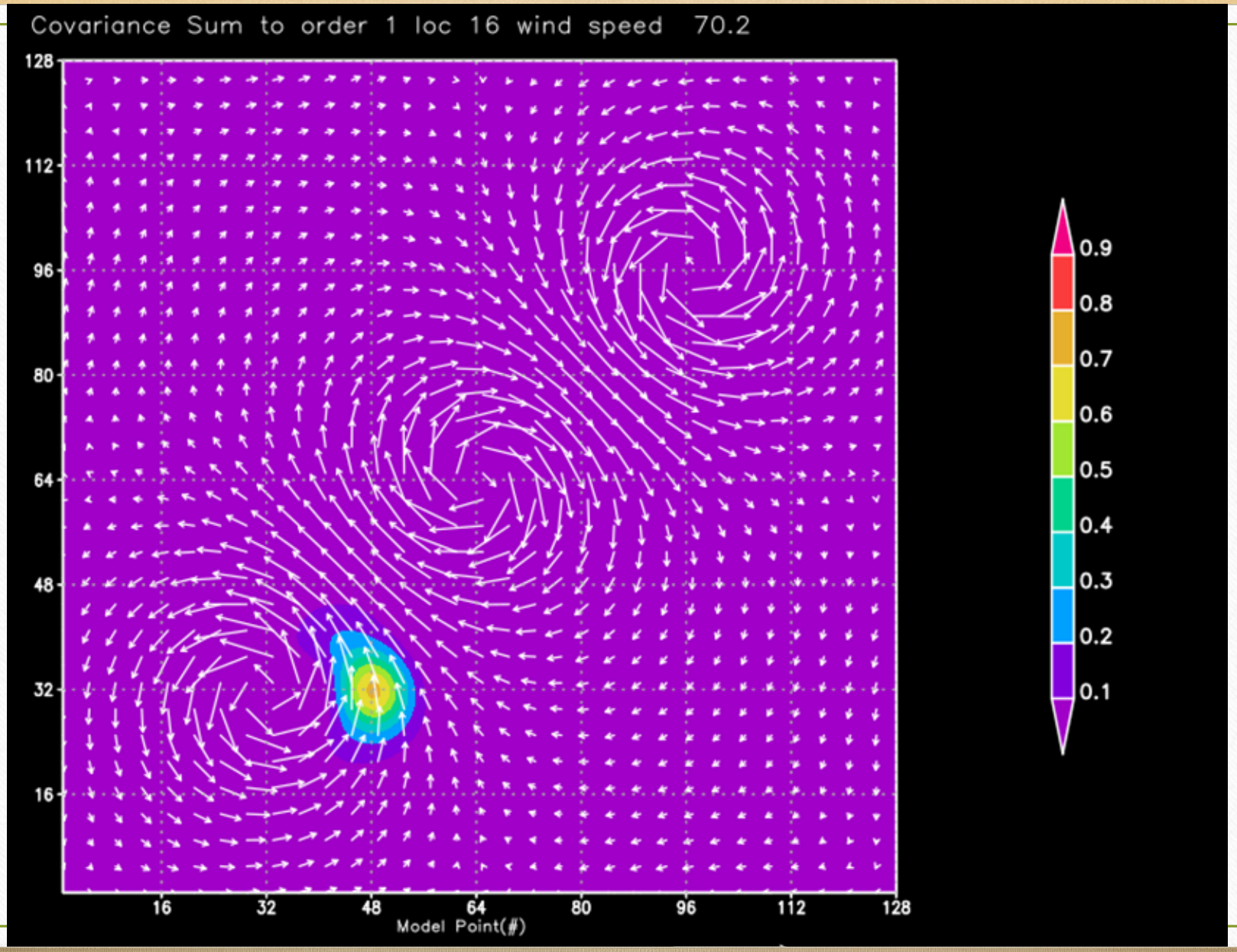

Covariance Sum to order 1 loc 50 wind speed 113.0
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

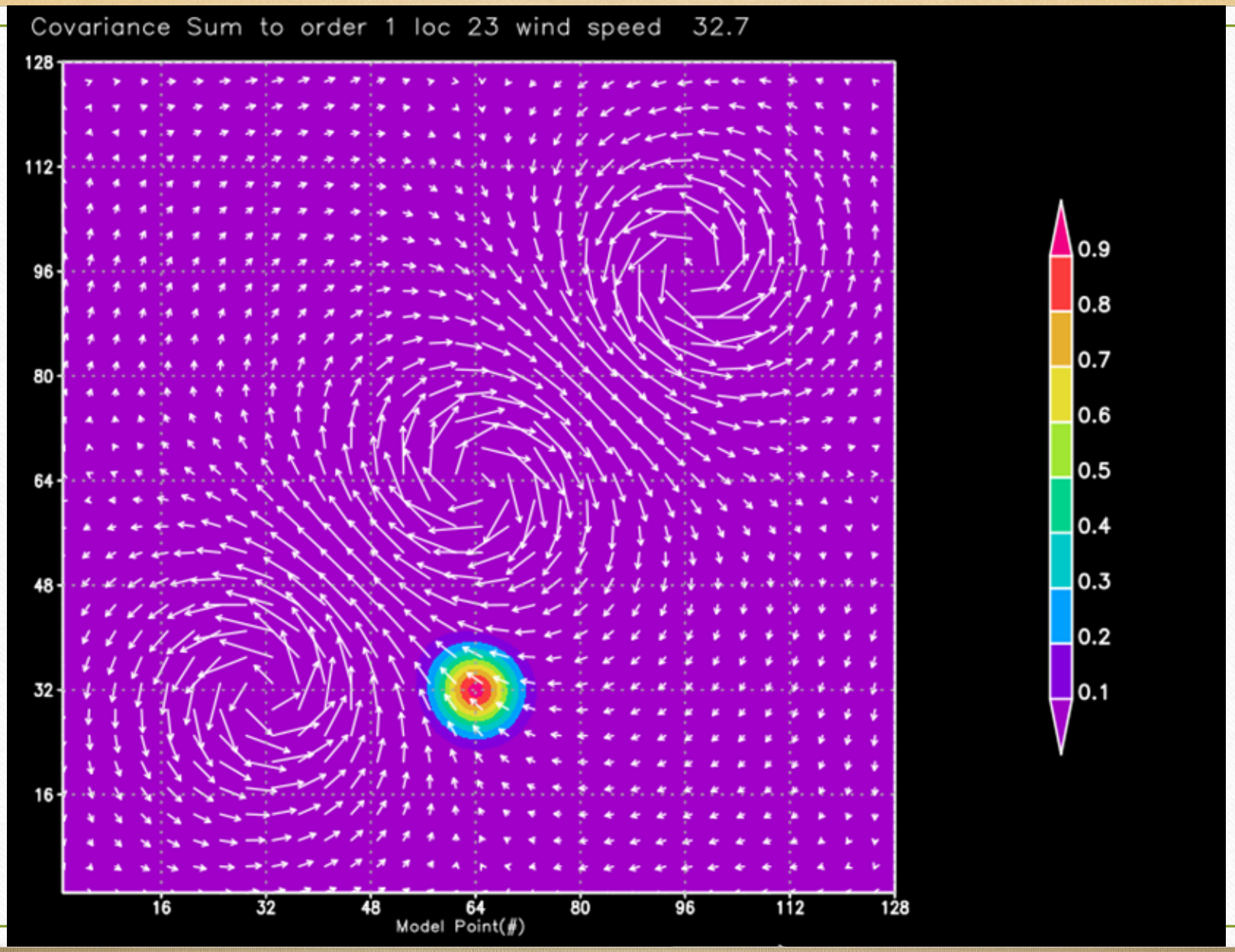

Covariance Sum to order 1 loc 9 wind speed 14.8
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

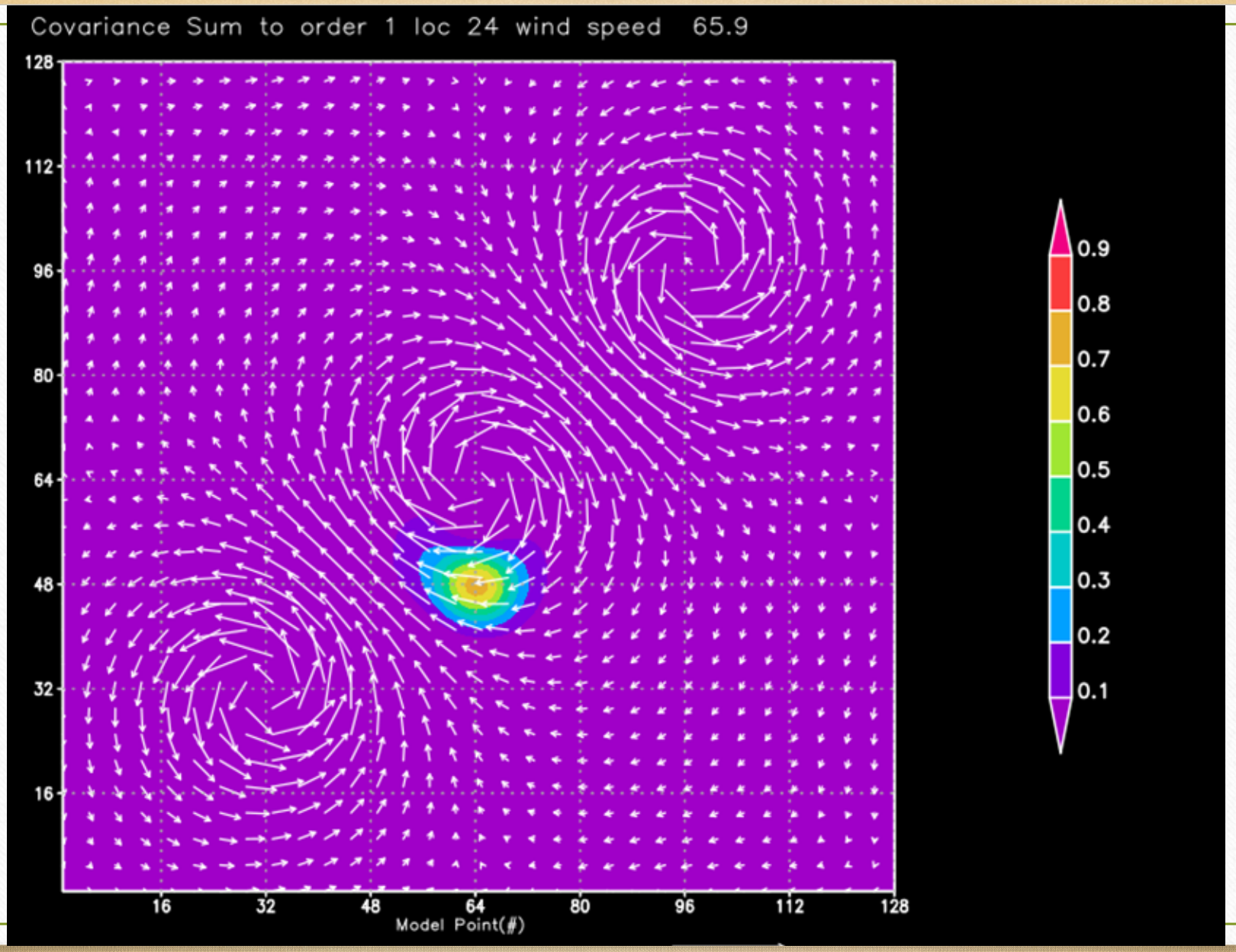

Covariance Sum to order 1 loc 2 wind speed 51.8
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

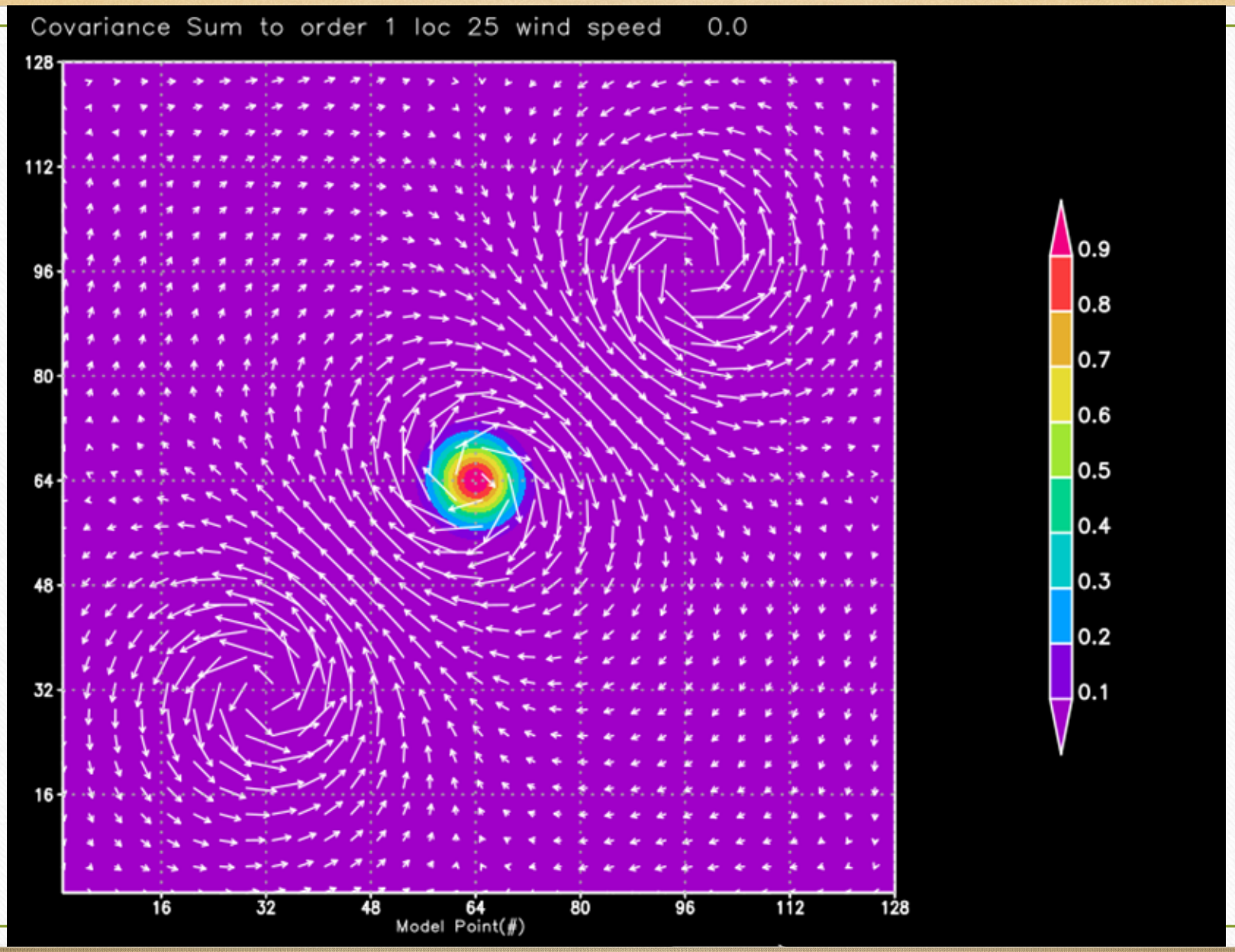

Covariance Sum to order 1 loc 1 wind speed 30.5
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

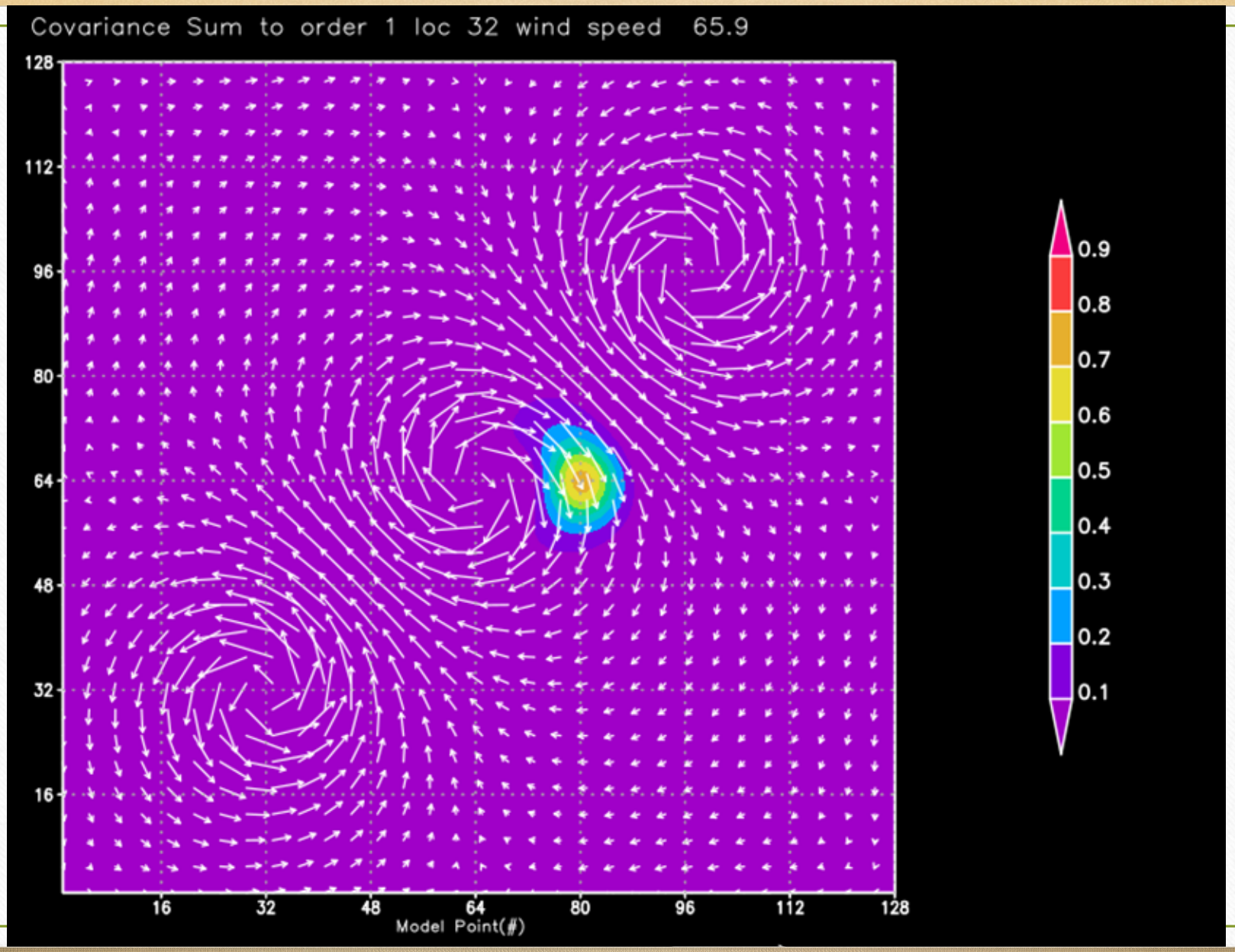

Covariance Sum to order 1 loc 8 wind speed 51.8
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

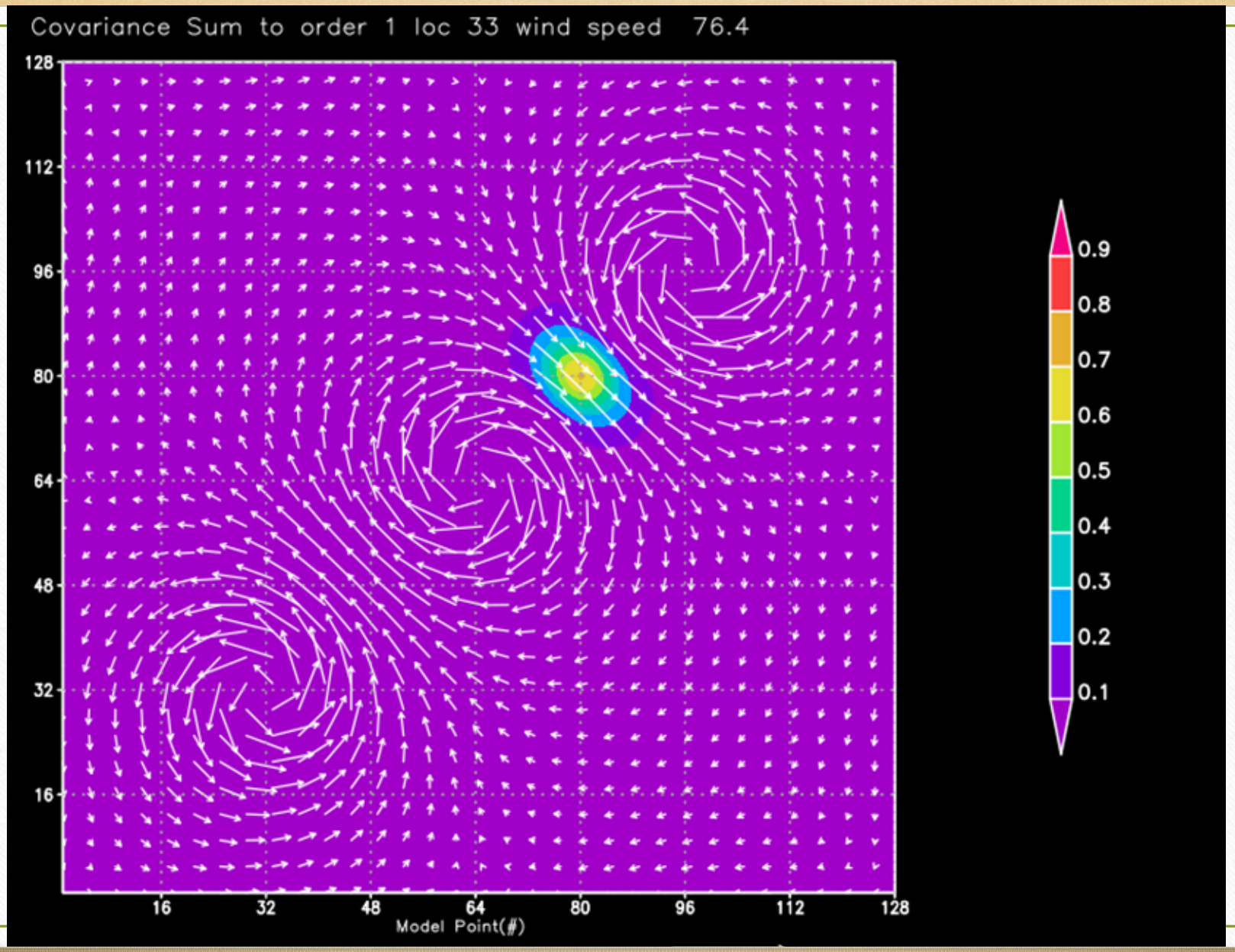

Covariance Sum to order 1 loc 16 wind speed 70.2
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

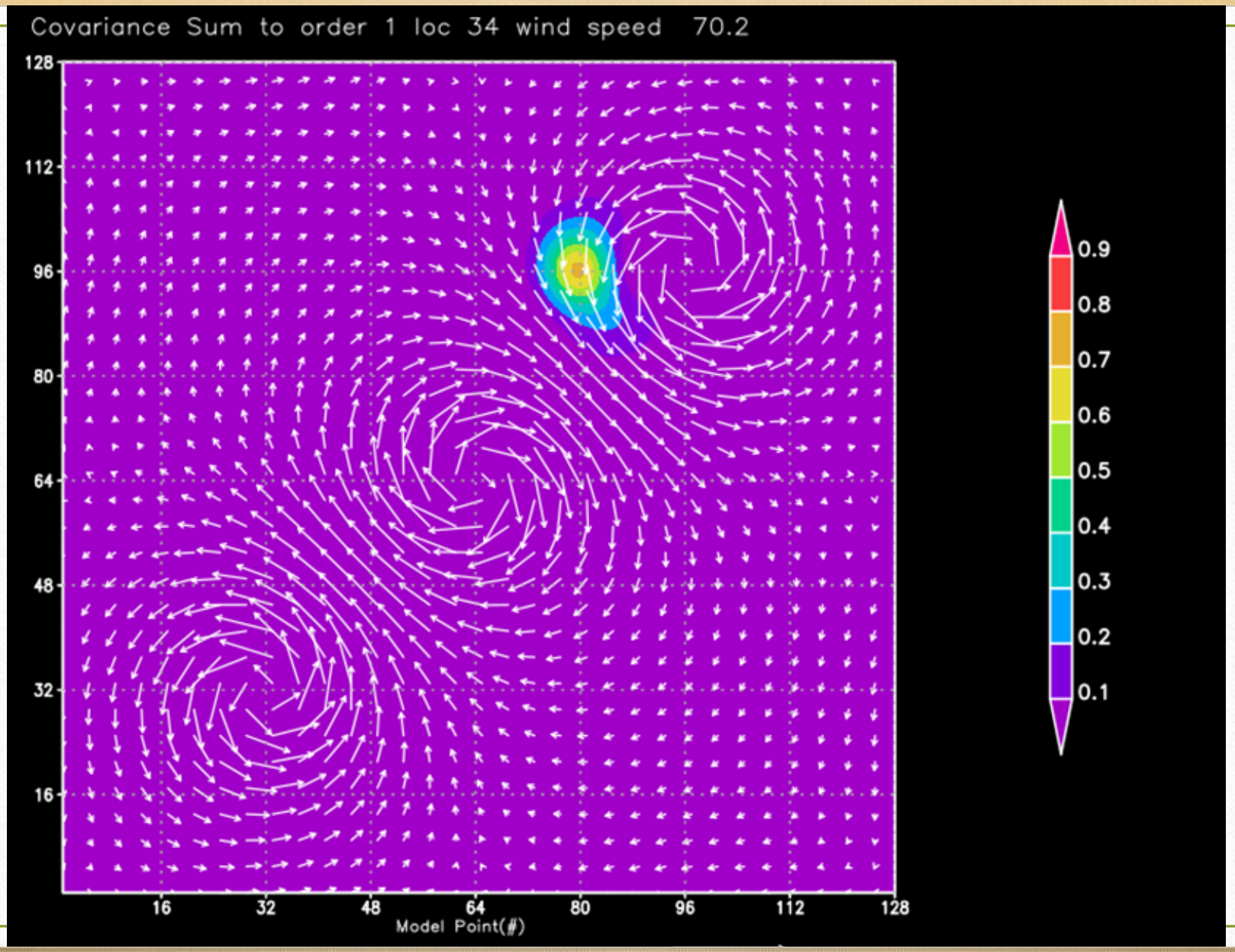

Covariance Sum to order 1 loc 23 wind speed 32.7
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

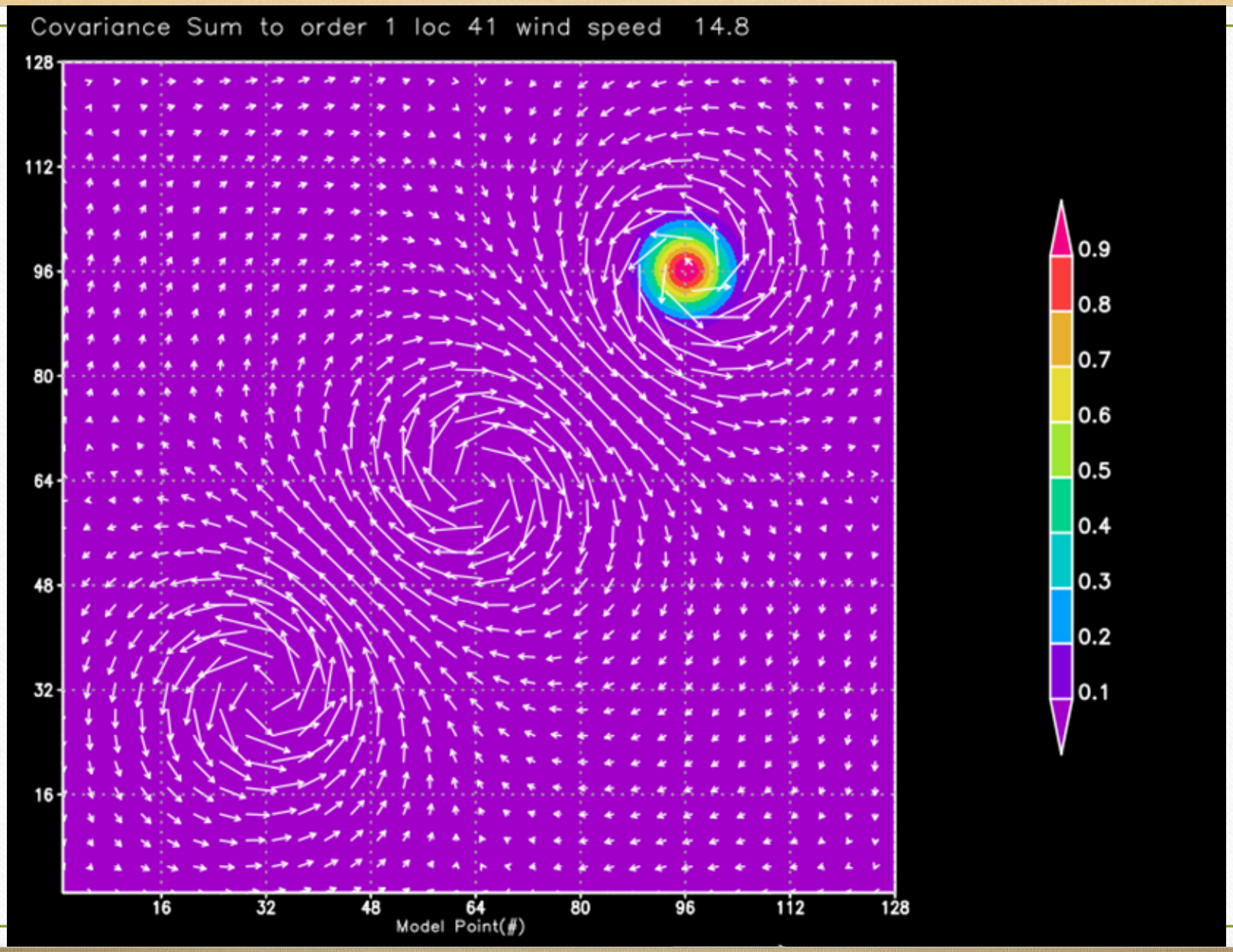

Covariance Sum to order 1 loc 24 wind speed 65.9
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

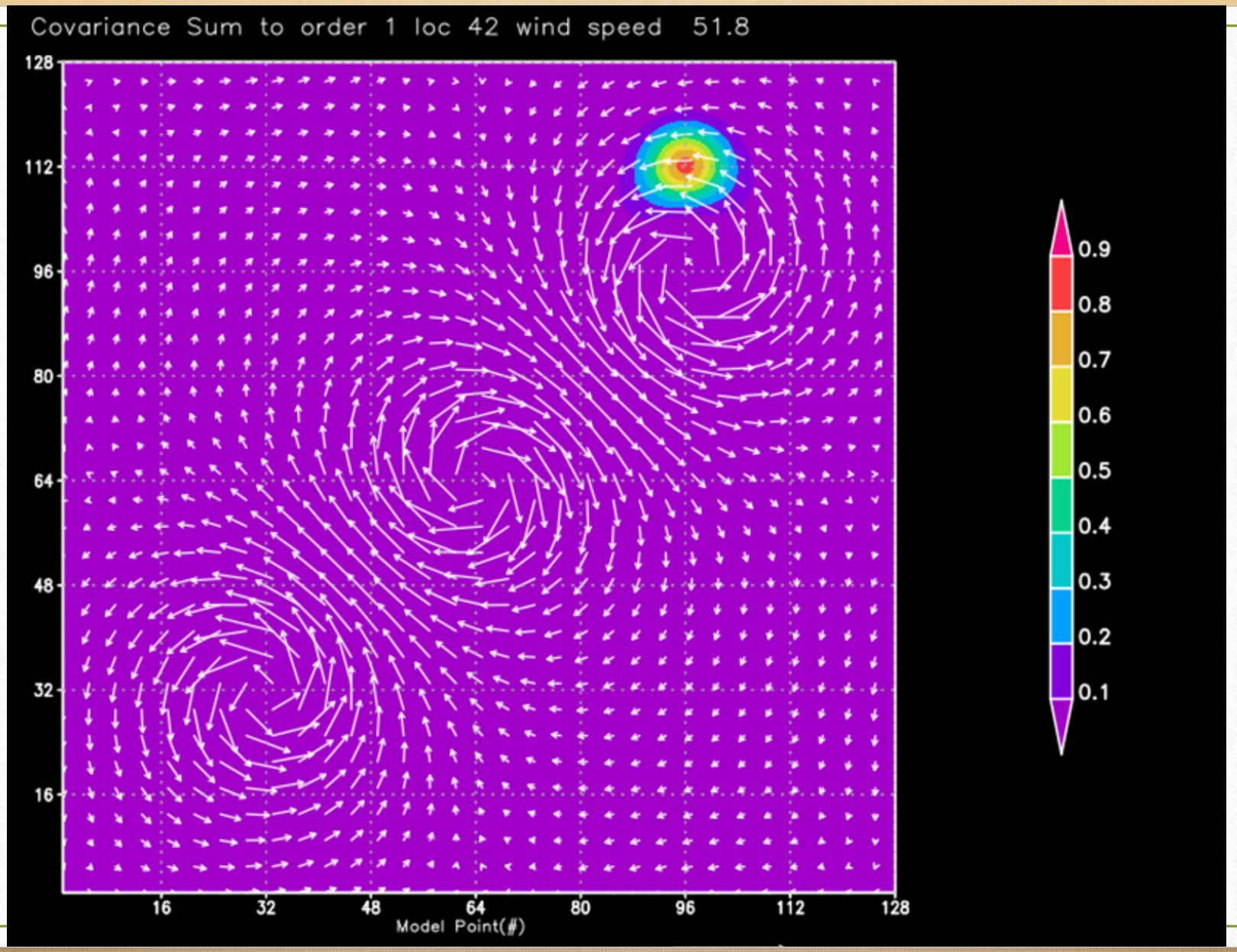

Covariance Sum to order 1 loc 25 wind speed 0.0
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Covariance Sum to order 1 loc 32 wind speed 65.9
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Covariance Sum to order 1 loc 33 wind speed 76.4
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Covariance Sum to order 1 loc 34 wind speed 70.2
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Covariance Sum to order 1 loc 41 wind speed 14.8
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

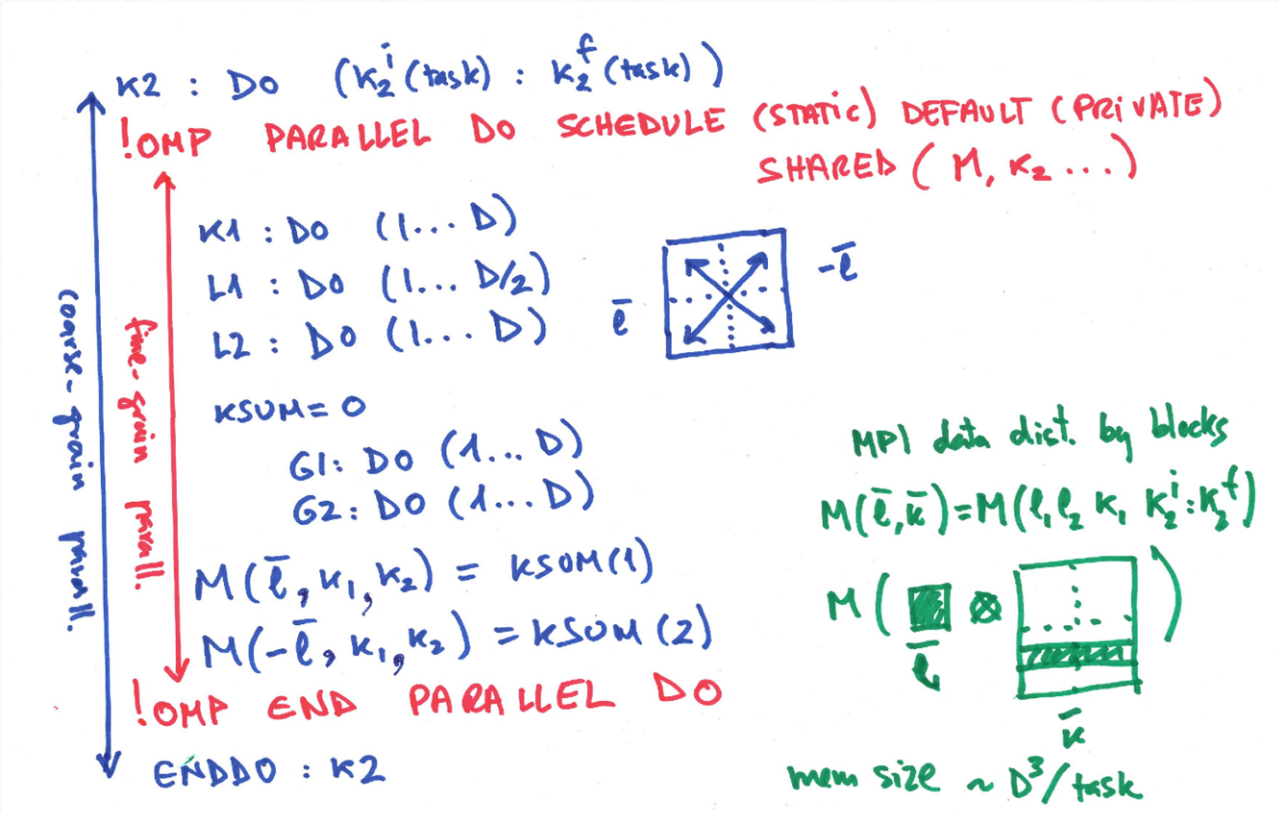

Covariance Sum to order 1 loc 42 wind speed 51.8
128
112
96
80
64
48
32
16
16
32
48
64
80
96
112
128
Model Point(#)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

- A prototype has been implemented on AEMET's HPC Atos-Bull facility
- The partition of AEMET's HPC used in this work consists of 48 nodes, each of these nodes has about **370 GB** RAM and contains 2 sockets with 64 cores per socket and 2 threads per core, giving a total of **256 logical CPUs per node**. Processor type is AMD EPYC (2.25 GHz, 225 W). Inter-node connectivity is provided by Infiniband HDR/HDR100 (100 Gb/s)
- The software source code is written in Fortran (compiler GCC x85_64-RedHat distribution version 8.3.1, 2019), parallelization is done using **MPI** (open-mpi 4.0.5 release, Bull.2.0 Jan 26, 2021 repo revision) and **OpenMP** (implementation as that embedded in the GCC compiler, probably 4.5) API standards.
- The **multi-dimensional Fourier transforms** (in the present case 4D) are computed with the **FFTW** ("Fastest Fourier Transform in the West") package (version 3.3.10 Sep 15, 2021)

Sketch of the solution implemented for the computation of M

The computation of M involves a six-level deep nested loops block . (coarse grain / fine grain solution)

- FFTW is very efficient at computing 4D FFTs, but imposes constraints on how data has to be distributed among tasks
- In this work a simple "distribution along one index" has been adopted (**max number tasks <= $\sqrt{N}$**)
- These constraints make not trivial to fully exploit M symmetries in this computation



Performance of various parallelization combinations (MPI-tasks / OpenMP threads)

128 x 128 domain (computation to 1st order)

- Scales linearly with the number of CPUs
- In the **computation of M**: small sensitivity to the precise distribution of CPUs among tasks and threads, slight advantage for more "fine grain" parallelization
- **Wall time (including FFT)** for a given number of CPUs, better the more MPI tasks

Nodes - Tasks/Node - Threads/Task	CPUs/Node - Total CPUs	Total Tasks (iter. / task)	CPU-Time M-matrix (secs)	Wall time (secs)
8 - 16 - 16	256 – 2048	128 (1)	45,5	**53**
8 - 8 - 32	256 - "	64 (2)	45,2	59
8 - 4 - 64	256 - "	32 (4)	45	65
8 - 2 - 128	256 - "	16 (8)	44	75
8 - 1 - 256	256 - "	8 (16)	29	133
16 - 8 - 32	256 – 4096	128 (1)	23	**31**
16 - 4 - 64	256 - "	64 (2)	22.8	35
16 - 2 - 128	256 - "	32 (4)	22	40
16 - 1 - 256	256 - "	16 (8)	14.8	67
32 - 4 - 64	256 – 8192	128 (1)	11,5	**19**
32 - 2 - 128	256 - "	64 (2)	11.4	23
32 - 1 - 256	256 - "	32 (4)	8.3	37

Performance of various parallelization combinations (MPI-tasks / OpenMP threads)
256 x 256 domain (problem 2^6=64 times bigger)

- **Computation of M**: scales slightly worse than linear (CPU time > x 64)
- **Wall time (including FFT):** scales slightly better than linear (CPU time < x 64)
- Times far away from being practical (~ 15 minutes)

Nodes - Tasks/Node - Threads/Task	CPUs/Node – Total CPUs	Total Tasks (iter. / task)	CPU-Time M-matrix (secs)	Wall time (secs)
32 - 8 - 32	256 – 8192	256 (1)	824	**886**
32 - 4 - 64	256 - "	128 (2)	828	918
32 - 2 - 128	256 - "	64 (4)	814	974

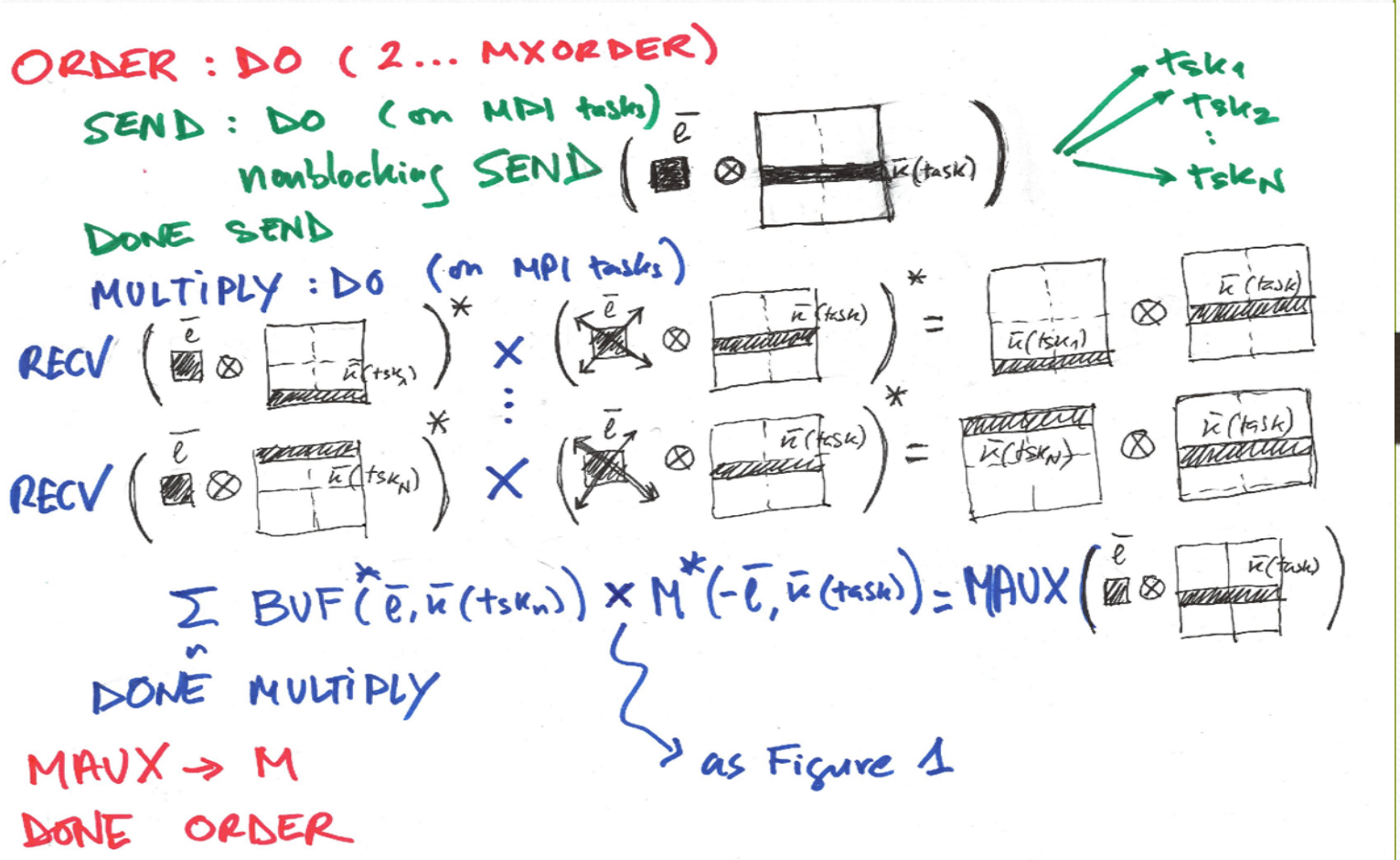
Effect of higher order terms

$$< \Delta_k \Delta_l >= B_k \delta_{k,l} - \mu M_{k,-l} B_k B_l + \mu^2 \left(\sum_m B_m M_{k,m} M_{m,-l} \right) B_k B_l - \mu^3 \left(\sum_{m,n} B_m B_n M_{k,m} M_{m,n} M_{m,-l} \right) B_k B_l + \cdots$$

- It is first necessary to code a practical way of carrying out the multiplications of M with itself
- After this, we study the convergence and tune the value of μ

Sketch of the solution implemented for the computation of the powers of M

CPU time ~ ORDER x CPU (M)



$\|MB\|$	μ	0	| 1 |	2	| 3 |	4	| 5 |	6	| 7 |
	$2.1\ 10^{-4}$	1.0	0.2	$6.6\ 10^{-2}$	$2.3\ 10^{-2}$	$8.3\ 10^{-3}$	$3.0\ 10^{-3}$	$1.1\ 10^{-3}$	$4.1\ 10^{-4}$
Fröben =9156 **L(p=1) =4750**	10^{-3}	1.0	0.98	1.5	2.5	4.2	7.3	12.8	22.5
	$5.0\ 10^{-4}$	1.0	0.49	0.37	0.31	0.26	0.23	0.20	0.17

- In the first row we have used the (sufficient) convergence criteria (ambiguous) $\|\mu MB\| < 1$ to give a value to μ Convergence is very fast but the effect is not noticeable

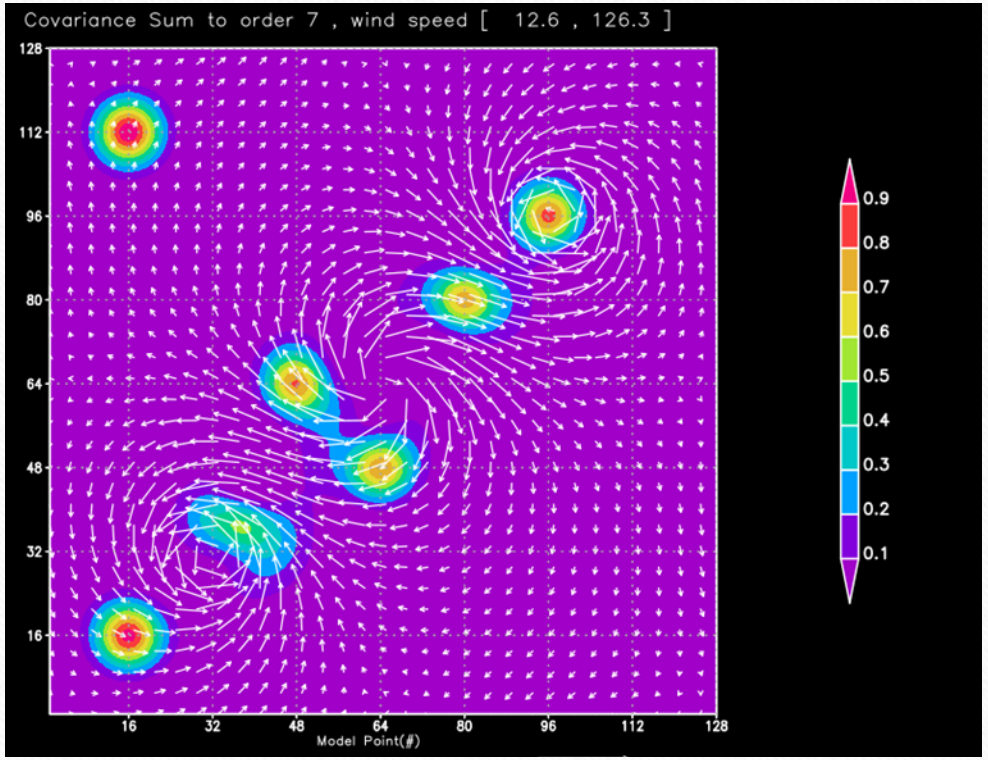
- In the second row we try 5μ . Convergence is still possible but many terms would be required (not practical)

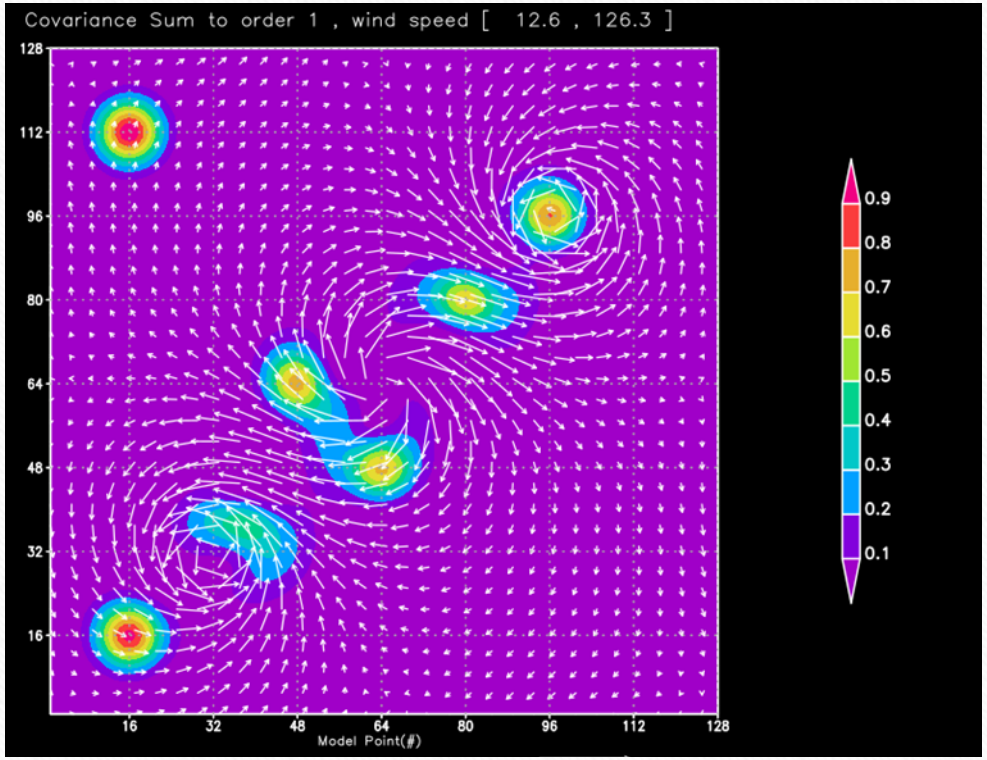
- In the third row we have tried ~2μ . Series decreases monotonously (Leibniz's rule for alternating series), and the effect is noticeable (animation at the beginning of the presentation)

Most of the modulation is already reached with just the first term !

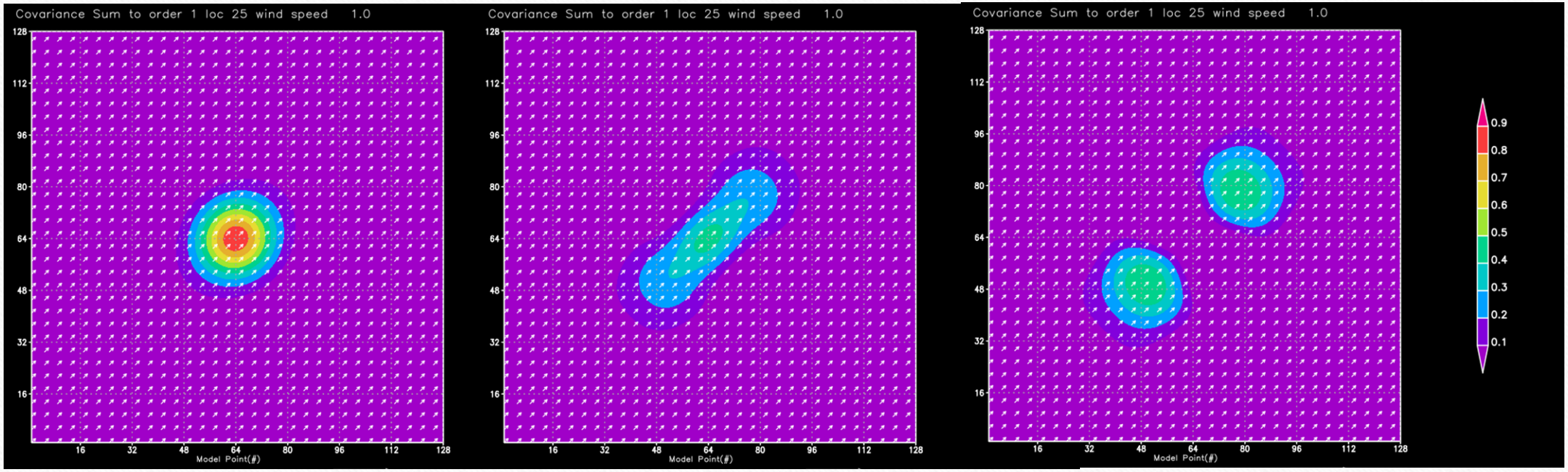
$$< \Delta_k \Delta_l >= B_k \delta_{k,l} - \mu M_{k,-l} B_k B_l$$

$\|MB\|$	μ	0	| 1 |	2	| 3 |	4	| 5 |	6	| 7 |
	$5.0\ 10^{-4}$	1.0	0.49	0.37	0.31	0.26	0.23	0.20	0.17
		-51		+6		+3		+3	





From this analysis we can get a "rule of thumb" that works fine : take for μ the value that, at the point of interest with maximum spread of variance this is limited to, say, 50% so as to prevent tear off of the covariance function.



Modelling Flow-Dependent Covariances with Gaussian Integrals

Recap and some conclusions

- It has been shown how to give flow-dependency to **B** by introducing a positive quadratic form that depends on an external arbitrary **V** field
- The idea is appealing, in the first place, for its potential in EDA
- Complicated issues regarding the performance of the coding of the algorithm on CPUs have been boarded
- Although the possibilities for optimization are, by no means, exhausted (GPUs ?), the results of this work strongly suggest that practical exploitation of this idea is at the moment out of reach