CREATING BILINEAR INTERPOLATION FOR SIMPLIFIED CMS MAGNETIC FIELD FOR GEANTV

Ananya
B.Tech (Mechanical Engineering)
IIT Bombay
PH-SFT Group Meeting
February 29th, 2016

Mentor:
John Apostolakis

Supervisor:
Dr. Federico Carminati

OUTLINE

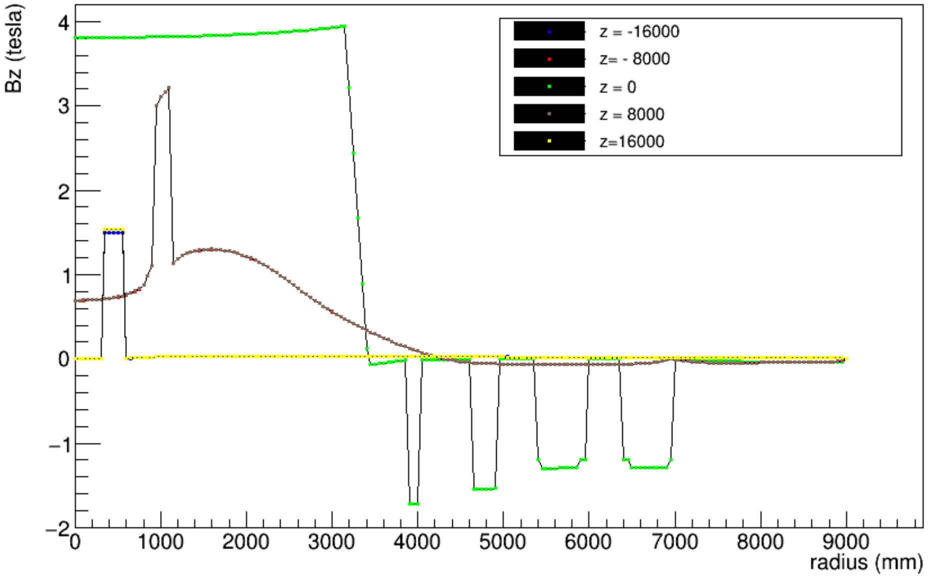
- GeantV and magnetic field tracking
- The CMS magnetic field
  - Approximation using Bilinear Interpolation
  - Vectorization and memory layout
  - Optimization
  - VTune Analysis
- Vectorization of Integration of motion

CONTEXT

- GeantV development using nearly realistic LHC detector
  - Geometry, Field, ~~Hits~~, Physics.
- Need standalone code for CMS-like magnetic field
  - Asked to do simple interpolation.
- Got field values from CMS simulation team.
- Note: Not related to CMS official code for B-field.

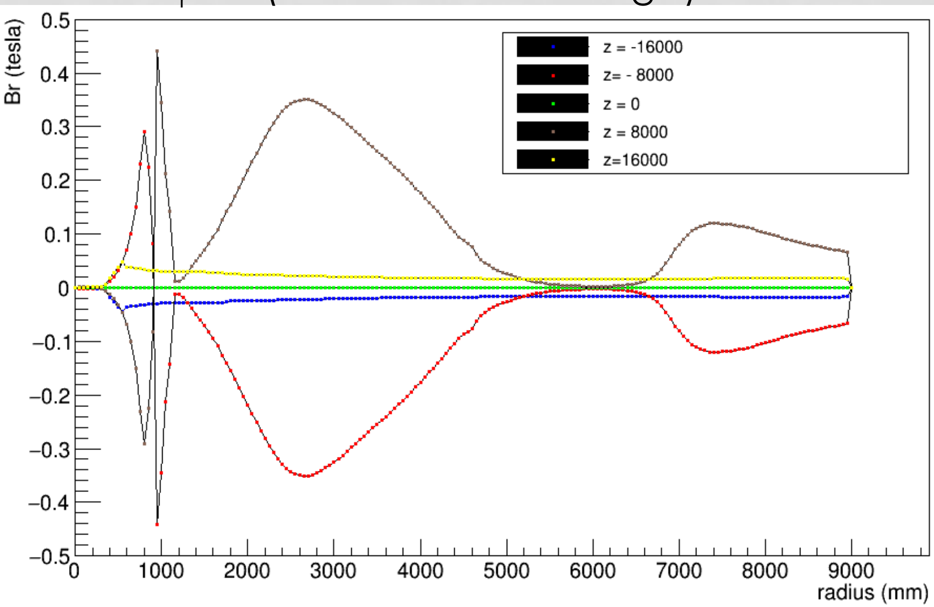
PROBLEM STATEMENT

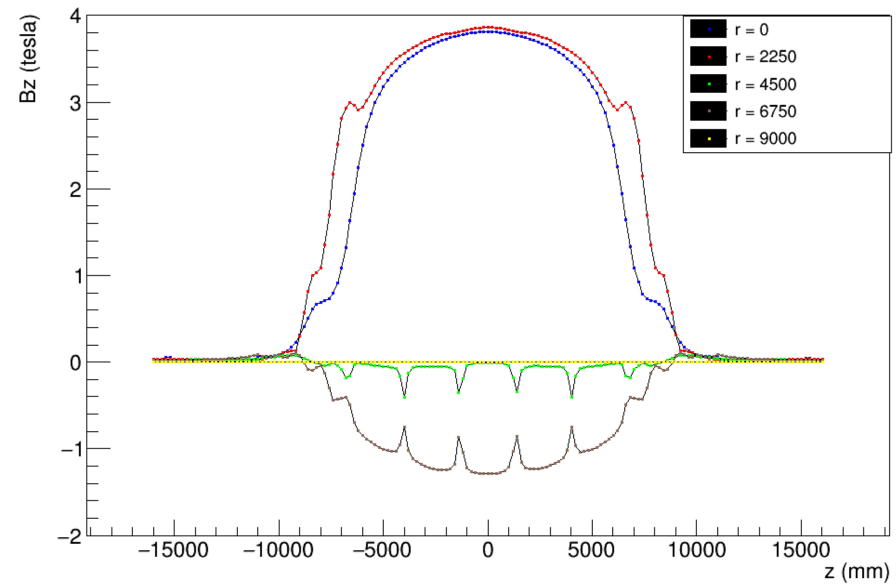
- Start with sample values of 2D CMS field.
- Assume phi-symmetric field.
- Find magnetic field given a point in 3D space.



B_z vs r (for boundaries along Z)

B_r vs r (for boundaries along Z)

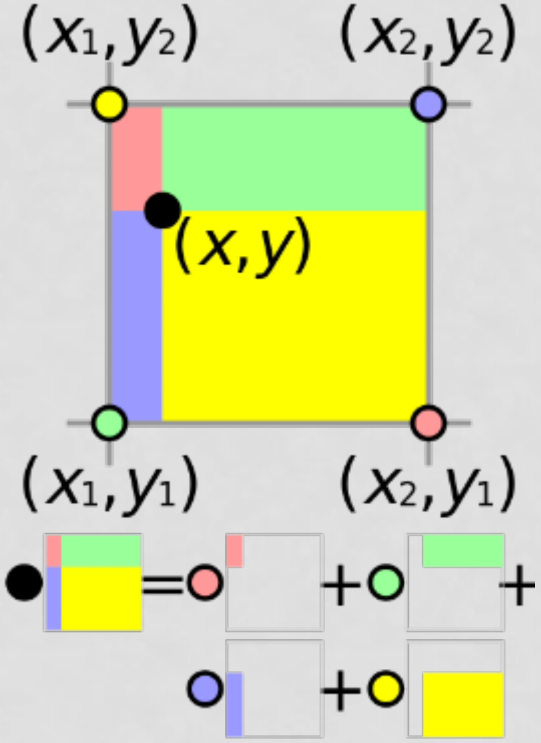




B_z vs z (for boundaries along r)

WHAT DO WE DO?

- Read given 2D map.
- Find corresponding magnetic field using bilinear interpolation on values from map.



CODE IMPROVEMENT

- First Version:
  - 27 multiplications
  - 32 additions/subtractions
  - 7 divisions
  - 8 modulus
  - 5 trigonometric ops
  - 1 sqrt
- Current Version:
  - 29 multiplications
  - 24-28 additions/subtractions
  - 2 floor
  - 3 max/min
  - 1 division
  - 1 sqrt
- Speed enhancement by O(100) (~e-8 s/event)

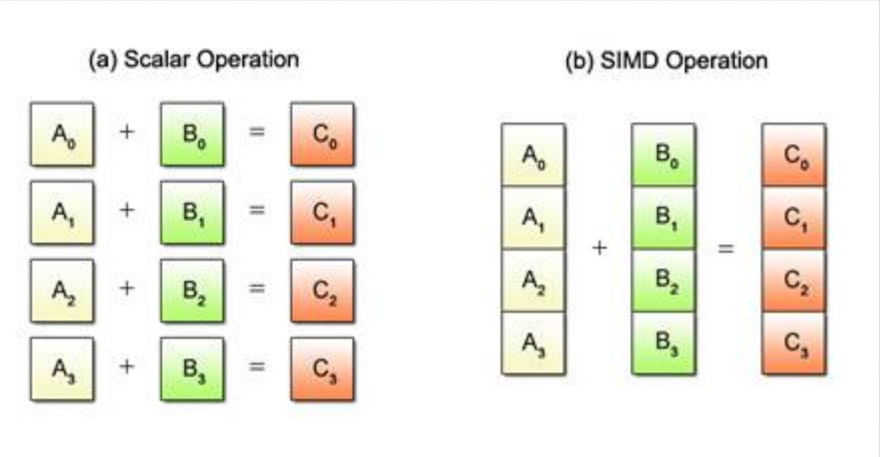
VALIDATING USING UNIT TEST

- Validated predicted magnetic field against given 2D map using unit tests.
- Validation at:
  - Node values
  - Mid-points of nodes
  - Middle of cell
- ~75k points

VECTORIZATION

VECTORIZATION

- "***Vectorization***" *(simplified) is the process of rewriting a loop so that instead of processing a single element of an array N times, it processes (say) 4 elements of the array simultaneously N/4 times.*
- What are we doing?
  - Processing multiple particles/tracks simultaneously



MEMORY LAYOUT

'Standard layout' – used also by Reorder/Reorder2

B_r	B_{phi}	B_z	B_r	B_{phi}	B_z	B_r	B_{phi}	B_z			
	i1			i2			i2+1 (!=i3)				
			B_r	B_{phi}	B_z	B_r	B_{phi}	B_z			
				i3			i4				

Minimum Memory

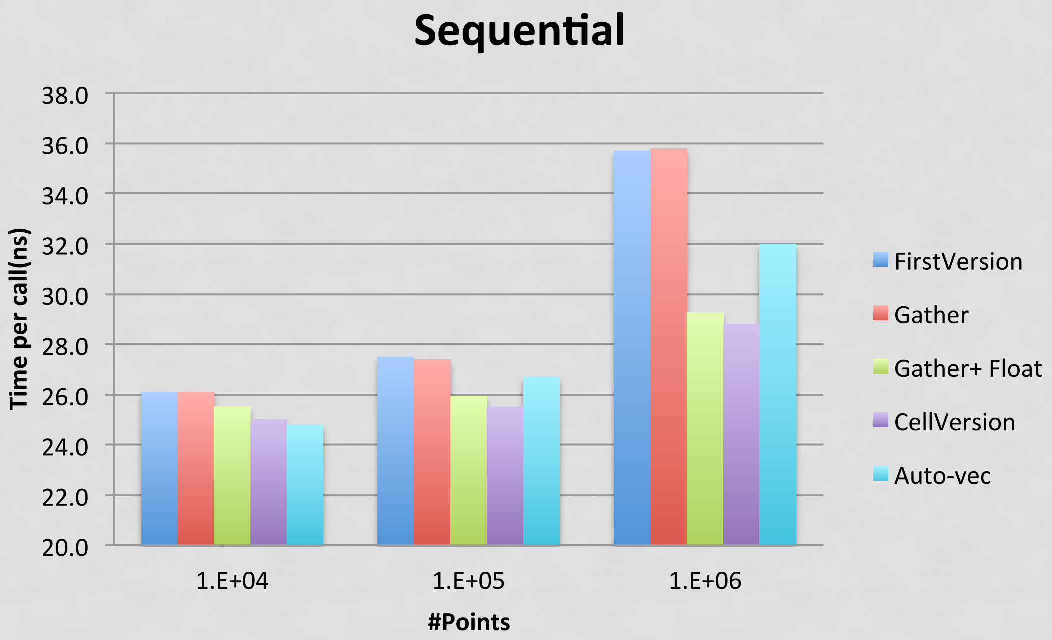
1 MagCellStruct (CellVersion)

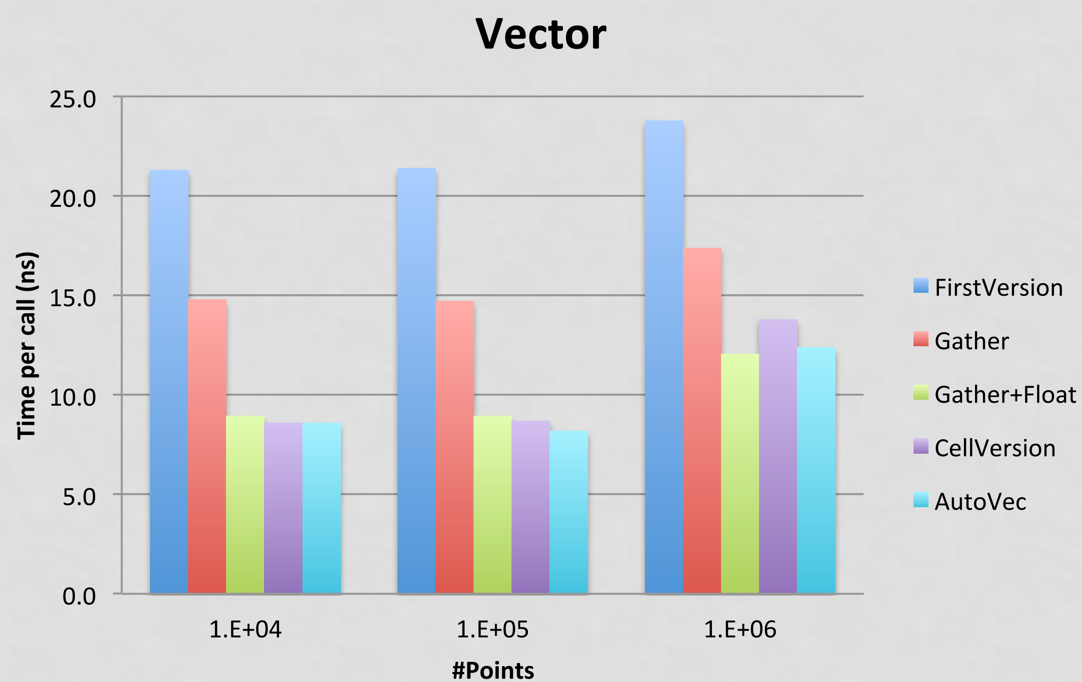
B_r	B_{phi}	B_z	B_r	B_{phi}	B_z	B_r	B_{phi}	B_z	B_r	B_{phi}	B_z
	i1			i2			i3			i4	

1 MagCellArray (AutoVec)

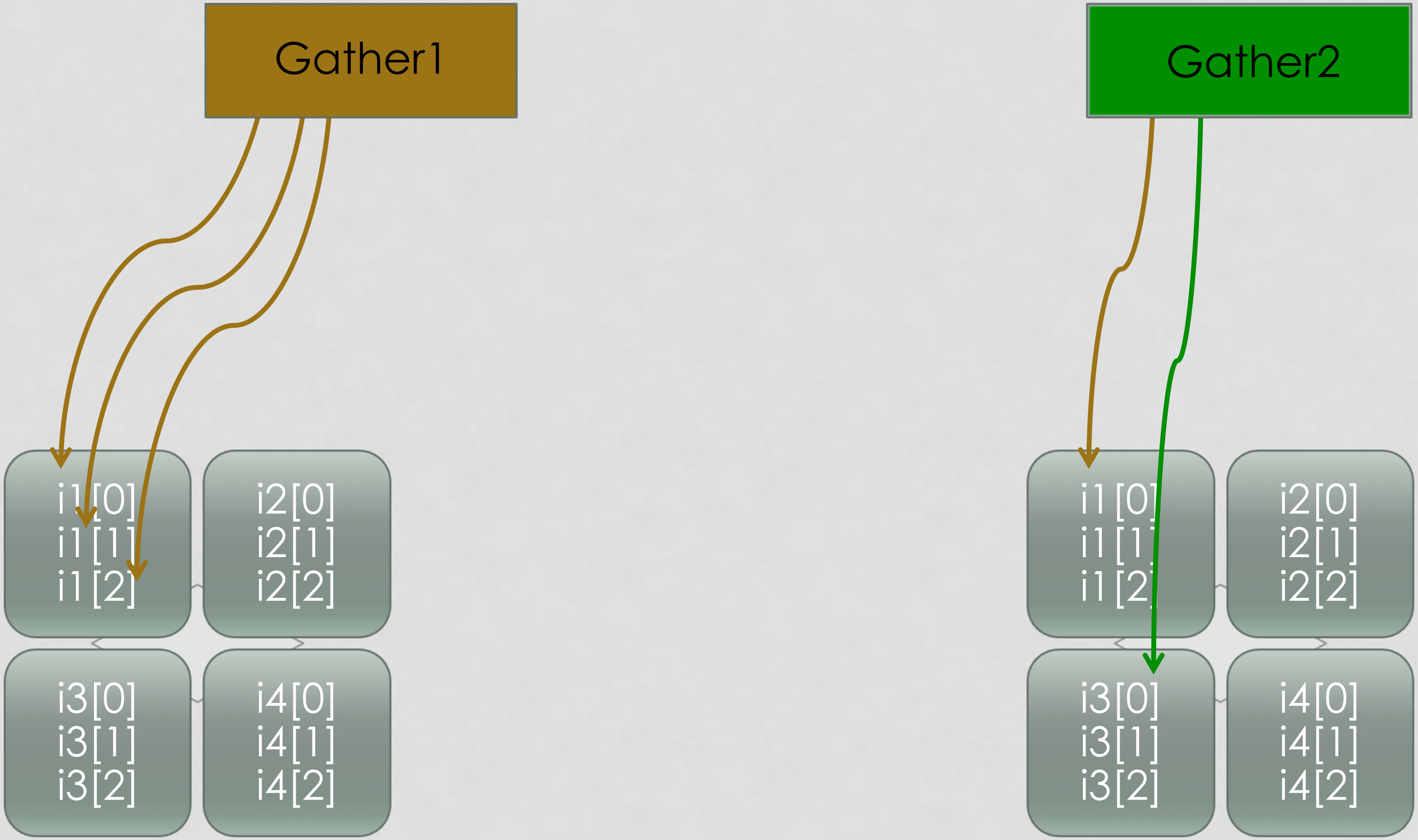
B_{r1}	B_{r2}	B_{r3}	B_{r4}	B_{phi1}	B_{phi2}	B_{phi3}	B_{phi4}	B_{z1}	B_{z2}	B_{z3}	B_{z4}
	B_r				B_{phi}				B_z		

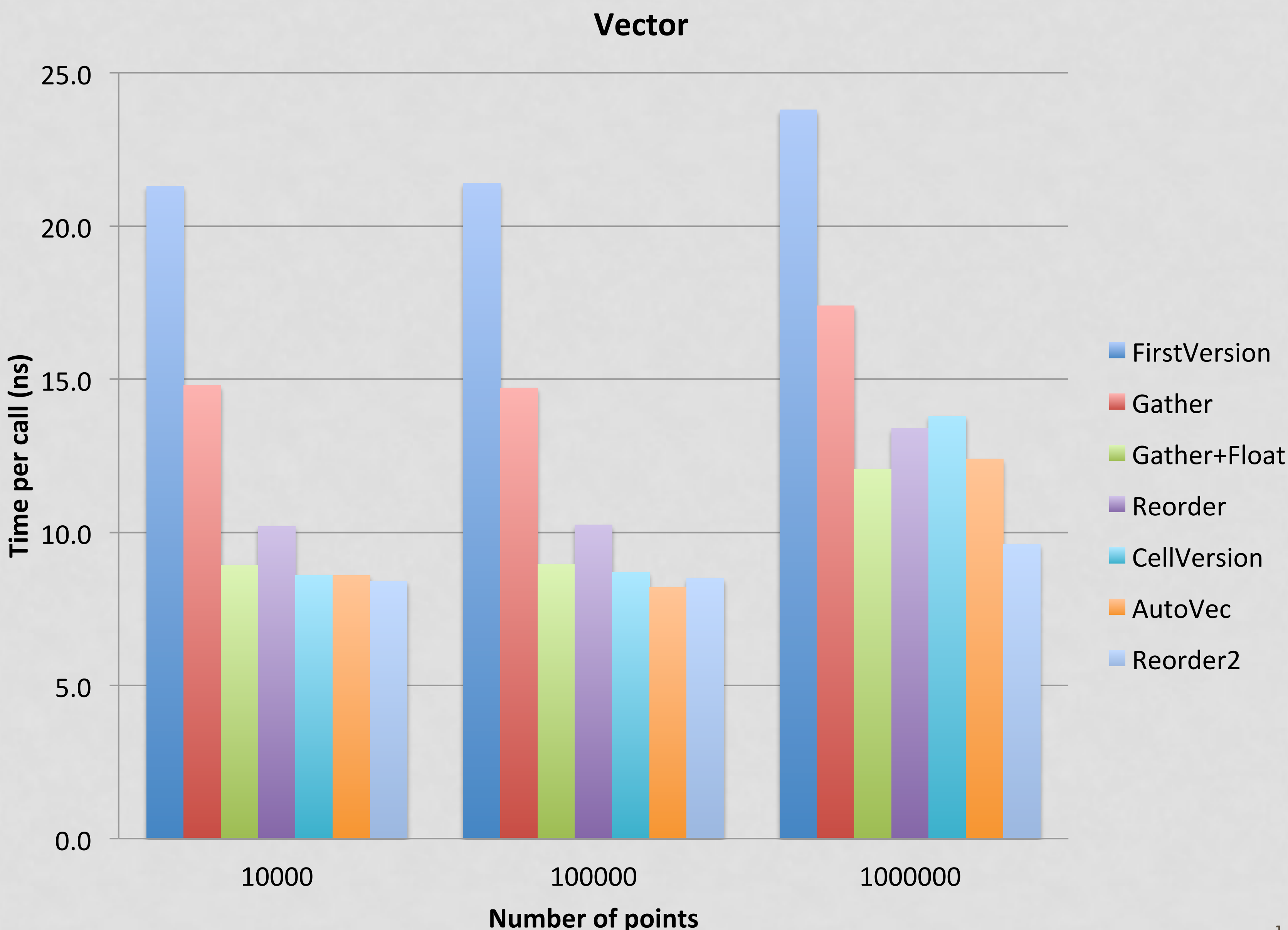
More memory


Sequential
38.0
36.0
34.0
32.0
30.0
28.0
26.0
24.0
22.0
20.0
Time per call(ns)
1.E+04
1.E+05
1.E+06
#Points
FirstVersion
Gather
Gather+ Float
CellVersion
Auto-vec

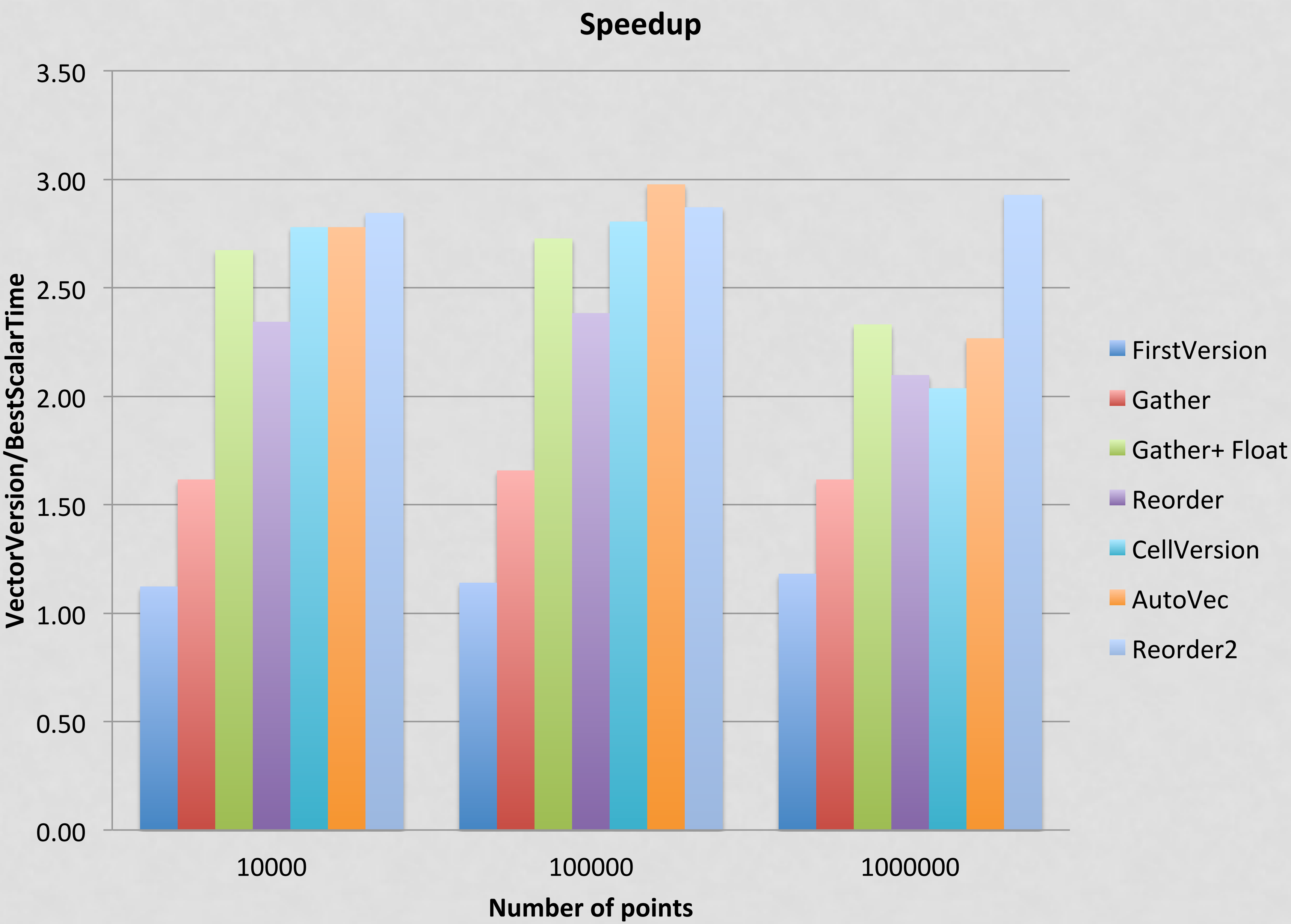

Vector
25.0
20.0
15.0
10.0
5.0
0.0
Time per call (ns)
1.E+04
1.E+05
1.E+06
#Points
FirstVersion
Gather
Gather+Float
CellVersion
AutoVec

DIFFERENT ORDERING OF GATHER





Vector
25.0
20.0
15.0
10.0
5.0
0.0
Time per call (ns)
10000
100000
1000000
Number of points
FirstVersion
Gather
Gather+Float
Reorder
CellVersion
AutoVec
Reorder2


Speedup
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
VectorVersion/BestScalarTime
10000
100000
1000000
Number of points
FirstVersion
Gather
Gather+ Float
Reorder
CellVersion
AutoVec
Reorder2

OBSERVATIONS

- Speedup factor of ~3
- Semi-realistic benchmark:
  - Half the points are new; the other half are ‘moved’ near to previous values.
  - Exponential random distribution.
  - Time reduced by ~5%. Likely effect is from cache.
- Difference in performance from changing doubles to floats:
  - 3-20% for sequential
  - 30-40% for vector version
- Difference in performance from changing order of memory operations:
  - 5-7% for sequential
  - 5-20% for vector version

VTUNE ANALYSIS

		Elapsed Time	Instructions Retired	CPI Rate	Back-end Bound	Memory Bound	Core Bound	Port Utilization
Sequential (nRep = 200)	Reorder2 (Haswell Xeon)	8.166	31.7B	0.947	0.786	0.282	0.505	0.327
Vector (nRep = 500)	Reorder2 (Haswell Xeon)	7.665	33.8B	0.776	0.697	0.143	0.555	0.463

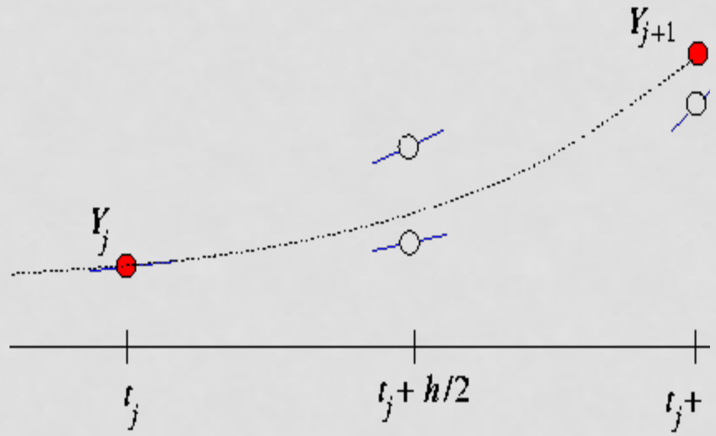
PART 2: VECTORIZATION OF INTEGRATION

INTEGRATING MOTION

		Vecotrizable
Magnetic Field	$\vec{B}$	✓
Force	$\vec{F} = \frac{1}{m}\vec{p} \times \vec{B}$	✓
Equation of motion (ODE)	$\frac{d\vec{\mathbf{x}}}{ds} = \frac{\vec{\mathbf{p}}}{\lvert\vec{\mathbf{p}}\rvert}$ $\frac{d\vec{\mathbf{p}}}{ds} = \frac{1}{\mathbf{p}}\overrightarrow{\mathbf{F}(v)}$	✓

Vectorizable

Runge Kutta

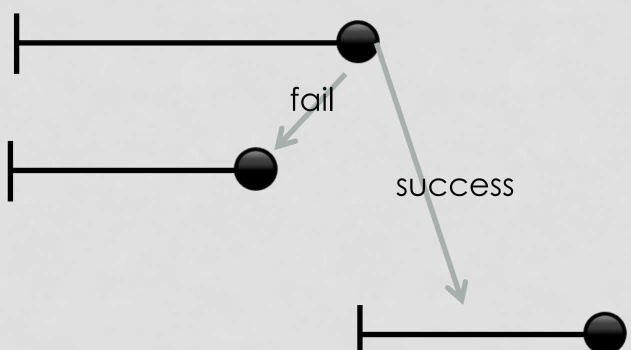


Input : y_o
Output: y_1, Δy_1

✓

Driver

Error Control
Adaptive Stepsize



✗

(Not Naively vectorizable)

WHY NOT?

- Different step lengths:

- Different magnetic field
- Different number of iterations for a success step

VECTORIZATION OF INTEGRATION DRIVER

- Takes a buffer stream of 16 particles/tracks.
- Starts working with 4 in Vc vector .
- As soon as integration is over for one track, insert a new track in its place.

Preliminary results (100 steps)		
	Sequential	Vectorized
#CashKarp Calls	435	172
#OneGoodStep Calls	324	94

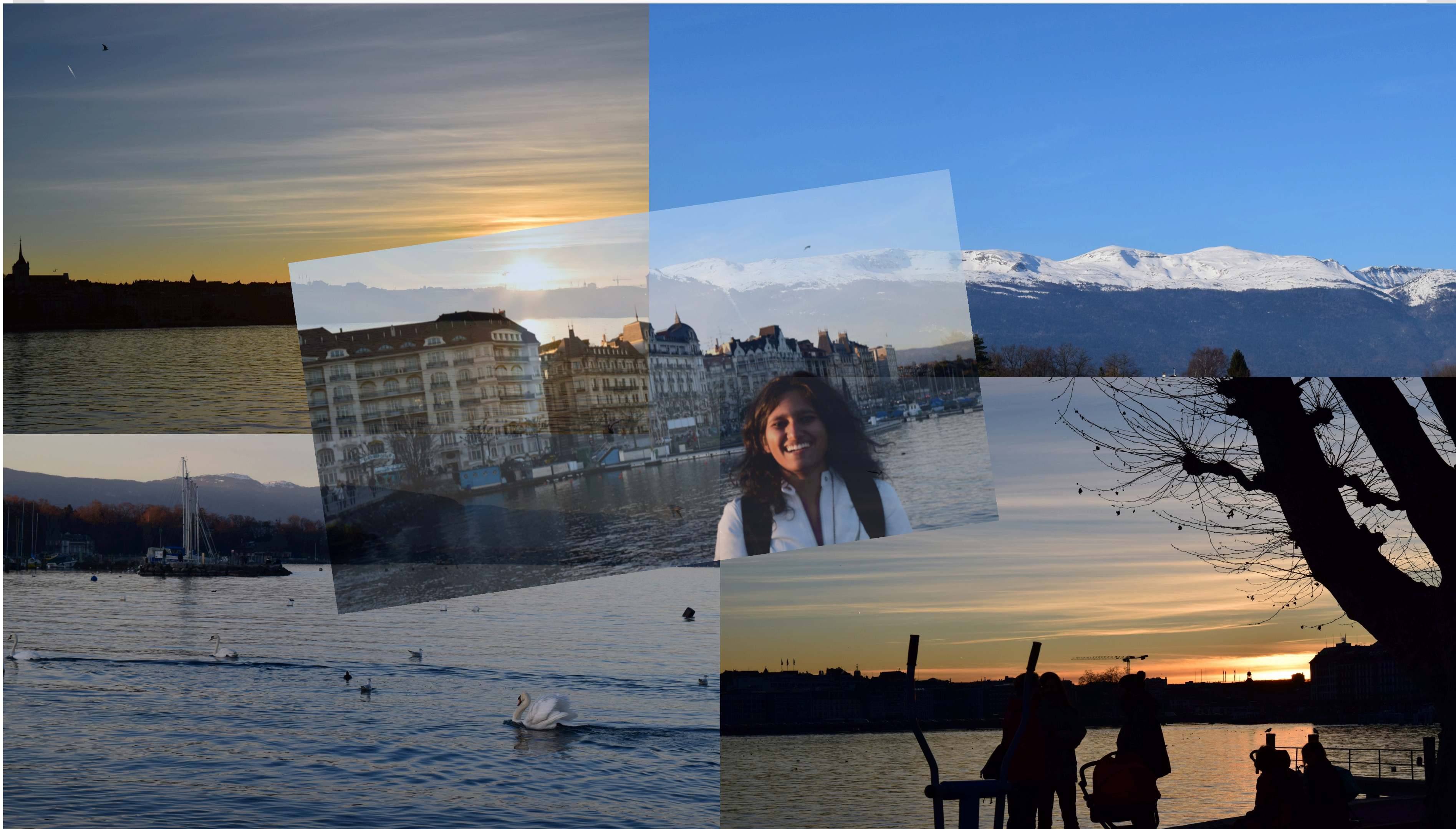
CONCLUSIONS

- Working sequential and vector bilinear interpolation approximation of CMS magnetic field
- Vector speedup of 3x on Haswell (Xeon)
- Demonstrated effect of memory layout and sensitivity to order of data points' access
- First version of integration driver

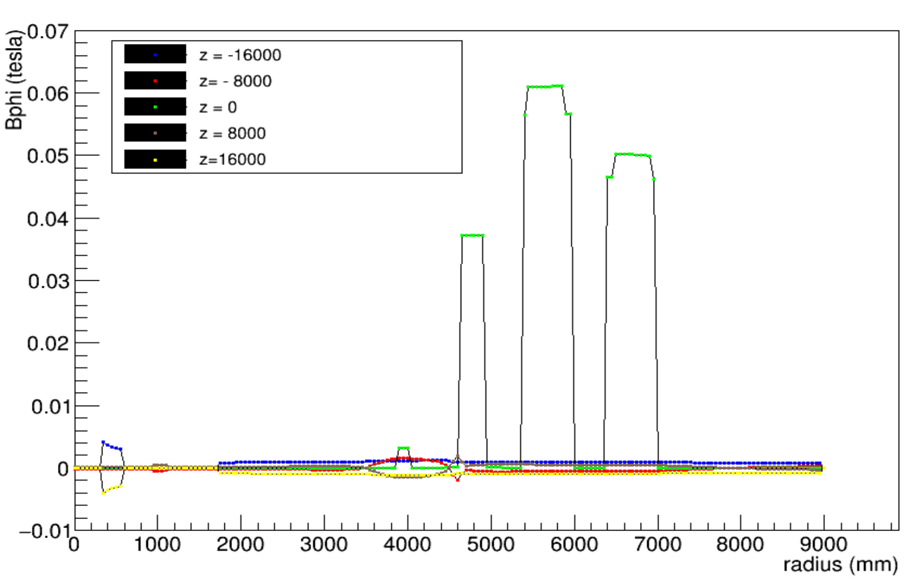
REFERENCES

- Bilinear Interpolation (Wikipedia) https://en.wikipedia.org/wiki/Bilinear_interpolation
- Stack Overflow http://stackoverflow.com
- https://software.intel.com/en-us/node/544392
- https://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch04s03.html
- Numerical Recipes in C, The Art of Scientific Computing
- https://www.kernel.org/pub/linux/kernel/people/geoff/cell/ps3-linux-docs/CellProgrammingTutorial/BasicsOfSIMDProgramming.html [Saul Teukolsky, William T. Vetterling, William H. Press, Brian P. Flannery]

THANK YOU!

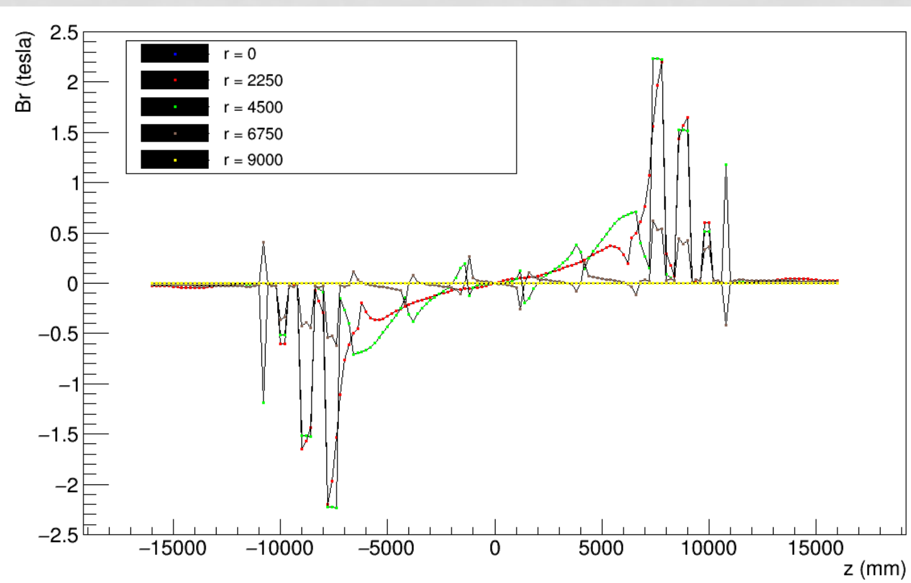


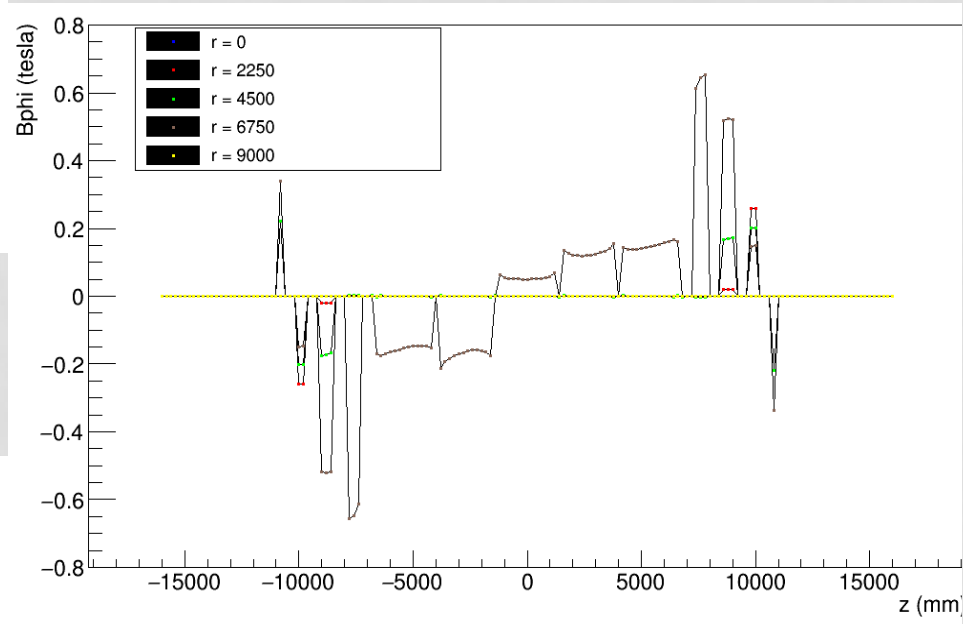
BACKUP SLIDES



B_{phi} vs r (for boundaries along Z)

B_r vs z (for boundaries along r)





B_{phi} vs z (for boundaries along r)

Sequential					
#points	Initial	VcGather	VcGather+ Float	CellVersion	AutoVec
1.E+04	26.1	26.1	25.5	25.0	24.8
1.E+05	27.5	27.4	26.0	25.5	26.7
1.E+06	35.7	35.8	29.3	28.8	32.0

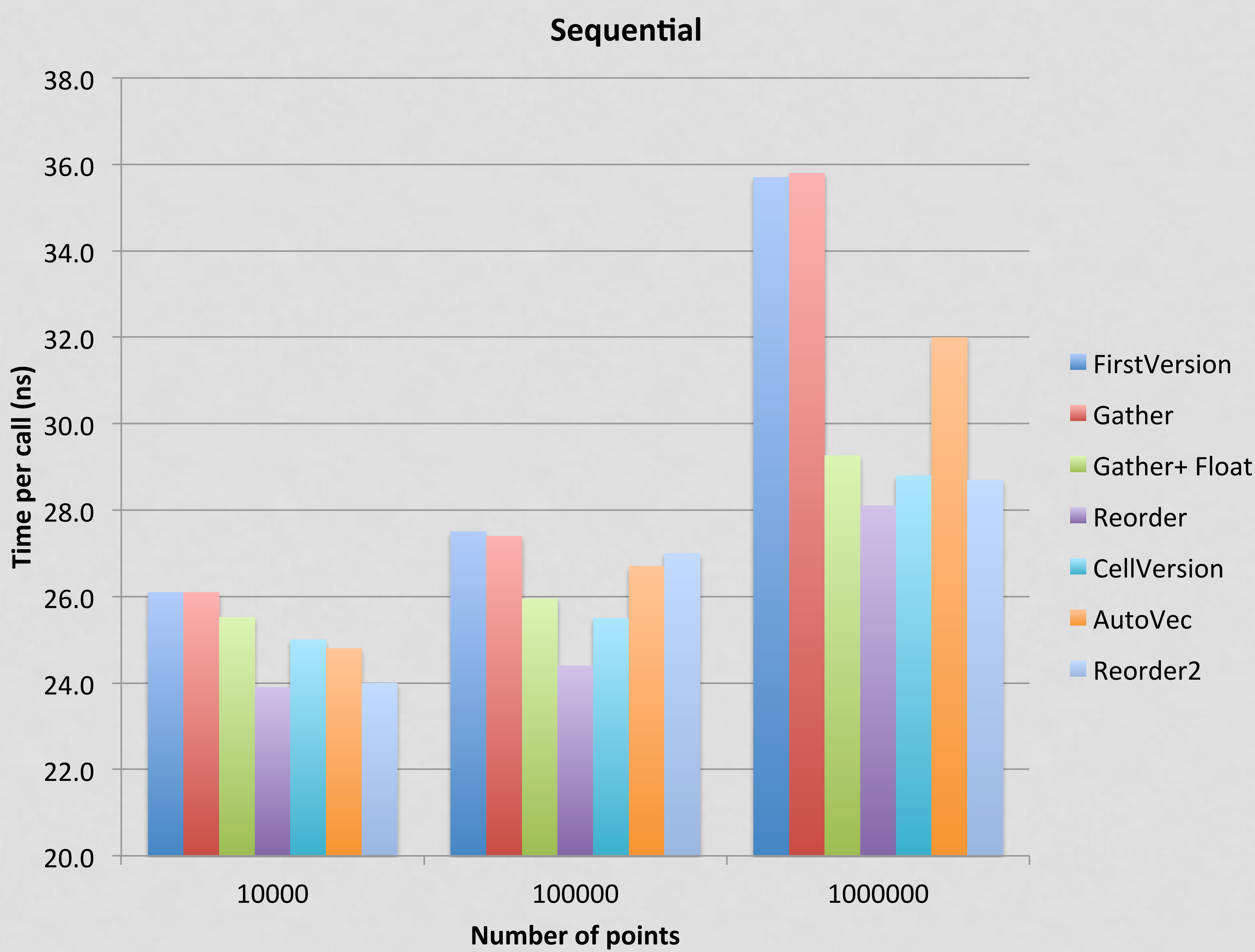
Vector					
#points	Initial	VcGather	VcGather+ Float	CellVersion	AutoVec
1.E+04	21.3	14.8	8.9	8.6	*8.6*
1.E+05	21.4	14.7	9.0	8.7	*8.2*
1.E+06	23.8	17.4	12.1	13.8	*12.4*

Speedup					
#points	Initial	VcGather	VcGather Float	CellVersion	AutoVec
1.E+04	1.23	1.76	2.85	2.91	2.88
1.E+05	1.29	1.86	2.90	2.93	3.26
1.E+06	1.50	2.06	2.43	2.09	2.58

STEPS

- Initially bruteforce approach
- VcGather:
  - Used gather method of Vc library
- VcGather + Floats :
  - Used floats instead of doubles
  - Loss of precision. Relative error > e-6 for certain test points

  (Floats used after this point)
- Reorder:
  - Reordered the gather function calls
  - Access field values in index order: (i1, then i3), then (i2, then i4).
  - In memory, contiguity in form: i1,i2,i3,i4
- Cell-version:
  - Store data in a different way. Store field values in blocks of 4.
  - Requires 4 times more memory initially since each point is stored 4 times
- Auto-vec:
  - Store data in a different way
  - Store Br[4] , Bz[4], Bphi[4] as one struct
  - Auto-vectorization by compiler for final step of adding weighted field values
- Reorder2:
  - Access components in order:
    - 1st component of 1st point
    - 1st component of 2nd point
    - 2nd component of 1st point
    - 3rd component of 1st point


Sequential
Time per call (ns)
38.0
36.0
34.0
32.0
30.0
28.0
26.0
24.0
22.0
20.0
10000
100000
1000000
Number of points
FirstVersion
Gather
Gather+ Float
Reorder
CellVersion
AutoVec
Reorder2

Sequential							
#points	FirstVersion	VcGather	VcGather+ Float	Reorder	CellVersion	AutoVec	Reorder2
1.E+04	26.1	26.1	25.5	23.9	25.0	24.8	*24.0*
1.E+05	27.5	27.4	26.0	24.4	25.5	26.7	*27.0*
1.E+06	35.7	35.8	29.3	28.1	28.8	32.0	*28.7*

Vector							
#points	FirstVersion	VcGather	VcGather+ Float	Reorder	CellVersion	AutoVec	Reorder2
1.E+04	21.3	14.8	8.9	10.2	8.6	*8.6*	8.4
1.E+05	21.4	14.7	9.0	10.2	8.7	*8.2*	8.5
1.E+06	23.8	17.4	12.1	13.4	13.8	*12.4*	9.6

Speedup							
#points	FirstVersion	VcGather	VcGather +Float	Reorder	CellVersion	AutoVec	Reorder2
1.E+04	1.12	1.61	2.67	2.34	2.78	2.78	2.85
1.E+05	1.14	1.66	2.73	2.38	2.80	2.98	2.87
1.E+06	1.18	1.61	2.33	2.10	2.04	2.27	2.93

VTUNE PARAMETERS

- Instructions Retired
  - Instructions actually needed by program flow
  - No. of instructions completely executed between 2 clocktick event samples
- CPI Rate:
  - Clockticks per Instructions Retired/Cycles per instruction
  - Higher CPI, more latency (cache misses, I/O etc)
- Back-end Bound:
  - Identify slots where no microps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. E.g. stalls due to data-cache misses or overloaded divider unit
- Memory Bound:
  - Fraction of slots where pipeline could be stalled due to demand load or store instructions
  - Incomplete in-flight memory demand loads
- Core Bound
  - Shouldn't be so high. Ideally 20%
  - Represents how much core non-memory issues were of a bottleneck.
  - Shortage in hardware compute resources, dependencies software's instructions
  - Indicates dependencies in program's data or instruction flow, overload of execution units
- Port Utilization:
  - Represents fractions of cycles during which an application was stalled due to core non-divider related issues. E.g. heavy data-dependency between nearby instructions

VTUNE ANALYSIS

		Elapsed Time	Instructions Retired	CPI Rate	Back-end Bound	Memory Bound	Core Bound	Port Utilization
Scalar (nRep = 200)	Auto_vec	8.978	28.2B	1.211	0.837	0.456	0.381	0.245
	Cell_version	8.276	32.7B	0.967	0.788	0.305	0.483	0.314
	VcGather	8.224	33.0B	0.949	0.787	0.295	0.492	0.322
	Reorder	**8.006**	**31.7B**	**0.965**	**0.797**	**0.320**	**0.476**	**0.296**
	Reorder2	*8.168*	*31.7B*	*0.984*	*0.794*	*0.307*	*0.487*	*0.321*
	Reorder2(Haswell)	8.166	31.7B	0.947	0.786	0.282	0.505	0.327
Vector (nRep = 500)	**Auto_vec**	**6.495**	**26.6B**	**0.849**	**0.725**	**0.335**	**0.390**	**0.214**
	Cell_version	7.359	26.5B	0.968	0.770	0.438	0.332	0.184
	VcGather	6.828	35.4B	0.669	0.655	0.151	0.505	0.371
	Reorder	7.841	35.7B	0.763	0.715	0.212	0.503	0.407
	Reorder2	*6.806*	*38.9B*	*0.610*	*0.657*	*0.153*	*0.504*	*0.343*
	Reorder2(Haswell)	7.665	33.8B	0.776	0.697	0.143	0.555	0.463

INTEGRATION DRIVER

- In order to propagate a track inside a field, the equation of motion of the particle in the field is integrated. In general, this is done using a Runge-Kutta method for the integration of ordinary differential equations.
- *Runge-Kutta* methods propagate a solution over an interval by combining the information from several Euler-style steps (each involving one evaluation of the right-hand f's), and then using the information obtained to match a Taylor series expansion up to some higher order.
- TMagFieldEquation calculates derivatives (dy/dx)
- TUniformMagField makes a constant magnetic field
- Stepper CashKarp takes a step and computes final position and error estimates which are used in adaptive size control to decide whether it is a good step or not

DIFFERENT ROUTINES

- Algorithm Routine
  - This implements the basic formulas of the method(CashKarp here), starts with dependent variables y_i at x, and calculates new values of the dependent variables at the value x + h.
- Stepper Routine
  - Makes quality control decision : acceptable solution or not
  - Takes the largest stepsize consistent with specified performance.
  - Calls the algorithm routine. It may reject the result, set a smaller stepsize, and call the algorithm routine again, until compatibility with a predetermined accuracy criterion has been achieved.
- Driver Routine
  - Starts and stops the integration, stores intermediate results, and generally acts as an interface with the user.