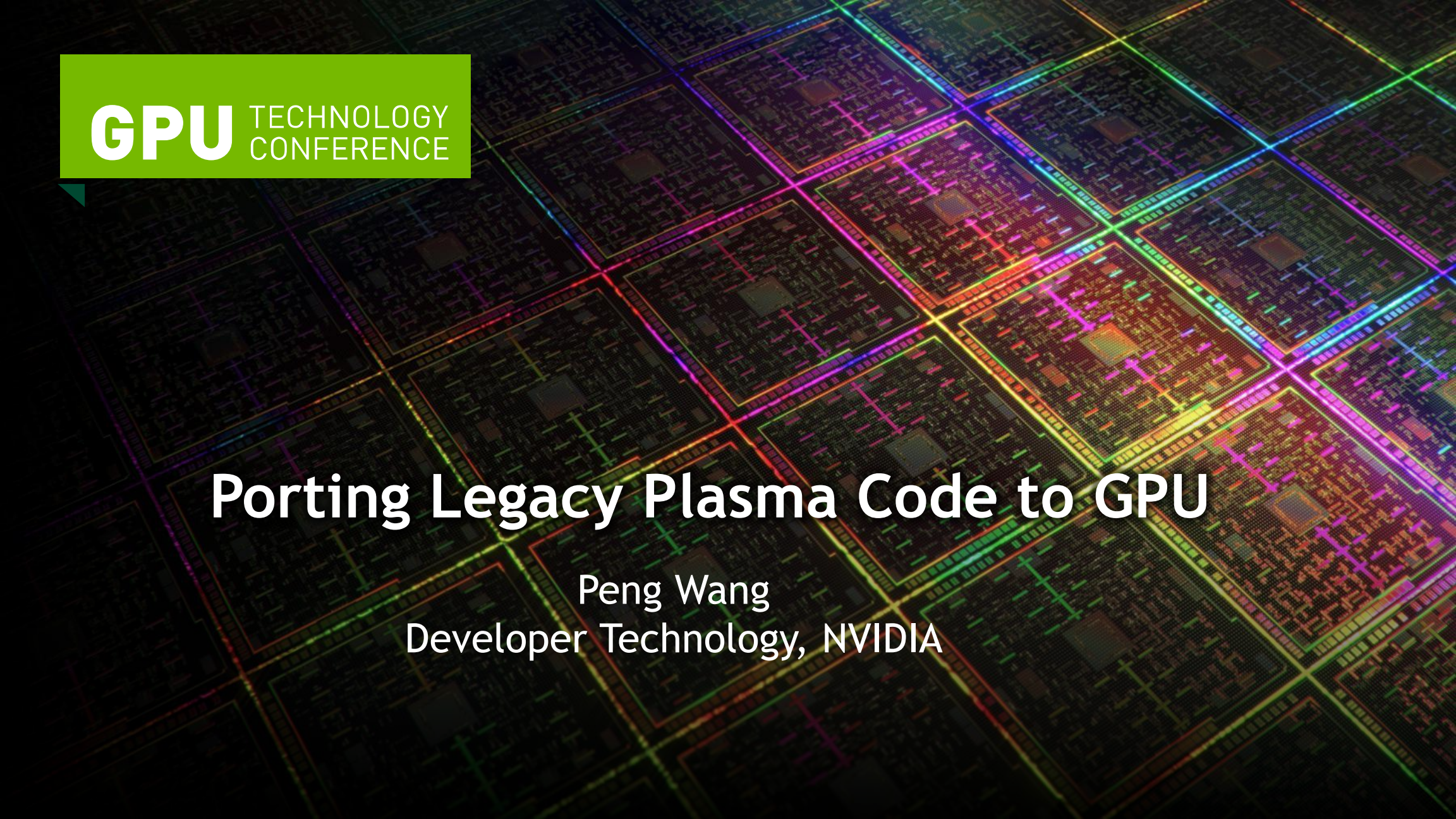
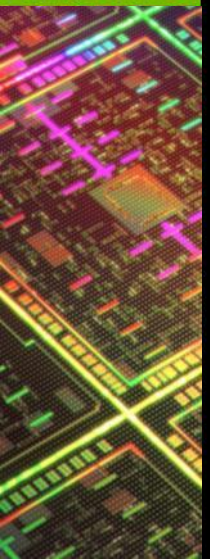


Porting Legacy Plasma Code to GPU

Peng Wang
Developer Technology, NVIDIA

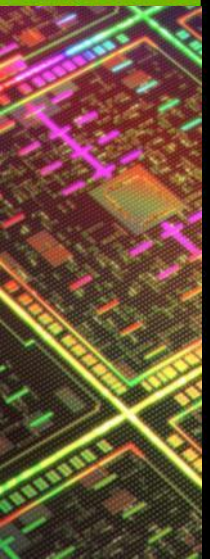


Collaborators

- Xiangfei Meng, Bao Zhang, Yang Zhao (NSCC-TJ/Tianhe-1A)
- Zhihong Lin (UC-Irvine), Yong Xiao (ZJU), Wenlu Zhang (USTC)

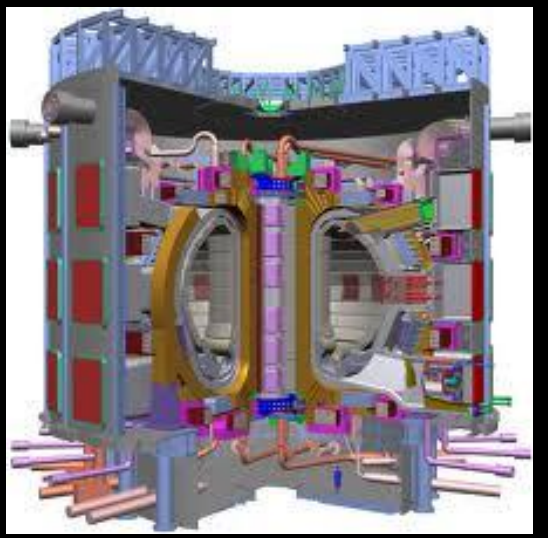
Overview

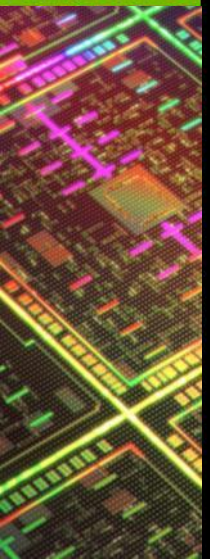
- The code: GTC
- GPU acceleration results and scaling analysis
- Optimization techniques



Gyrokinetic Toroidal Code (GTC)

- Massively parallel particle-in-cell code for first-principles, integrated simulations of the burning plasma experiments such as the *International Thermonuclear Experimental Reactor* (ITER)
  - UC-Irvine, Princeton, ZJU, USTC, PKU, etc.
- Key production code for DOE-SciDAC and China ITER Special Research Program
- First fusion code to reach teraflop in 2001 on Seaborg and petaflop in 2008 on jaguar





Benchmark Profile

Benchmark:

- Parameters from production fusion simulations studying electron turbulence
- 2B ions, 0.83 B electrons
- 128 MPI processes on 128 nodes, with 6 OpenMP threads

Platform: Tianhe-1A

- Compute node:
  - 2 Intel Xeon 5670 (6c, 2.93 GHz)
  - 1 NVIDIA Tesla M2050
  - 24 GB

	128 CPU	%
loop	521.43	
field	0.63	0.12%
ion	54.4	10.43%
shifte	84.6	16.22%
pushe	365.4	70.08%
poisson	4.4	0.84%
electron other	12	2.30%

Benchmark Profile

Benchmark:

- Parameters from production fusion simulations
- 2B ions, 0.83 B electrons
- 128 MPI processes on 128 nodes, with 6 OpenMP threads

Platform: Tianhe-1A

- Compute node:
  - 2 Intel Xeon 5670 (6c, 2.93 GHz)
  - 1 NVIDIA Tesla M2050
  - 24 GB

Focus on pushe+shifte first!

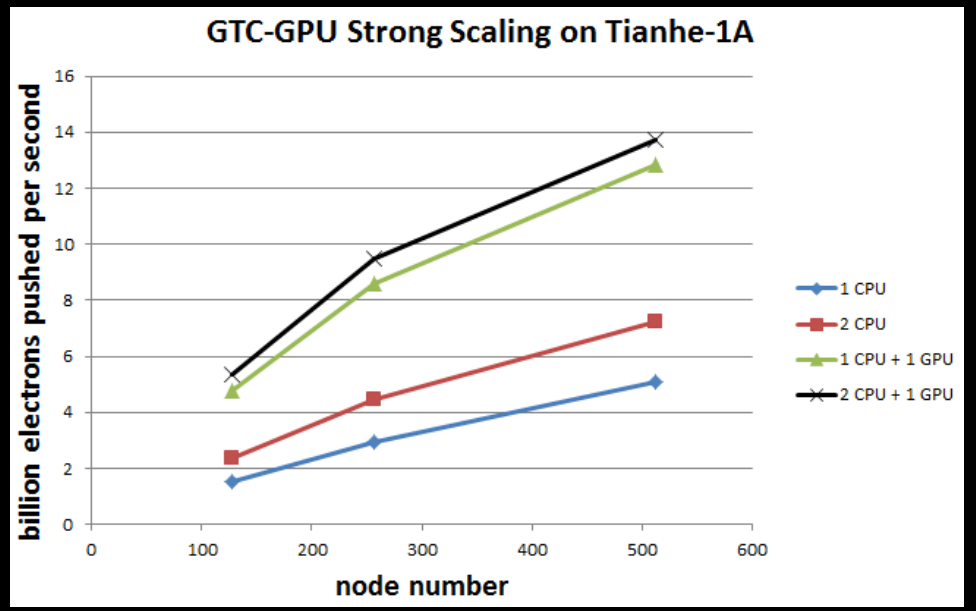
	128 CPU	%
loop	521.43	
field	0.63	0.12%
ion	54.4	10.43%
shifte	84.6	16.22%
pushe	365.4	70.08%
poisson	4.4	0.84%
electron other	12	2.30%

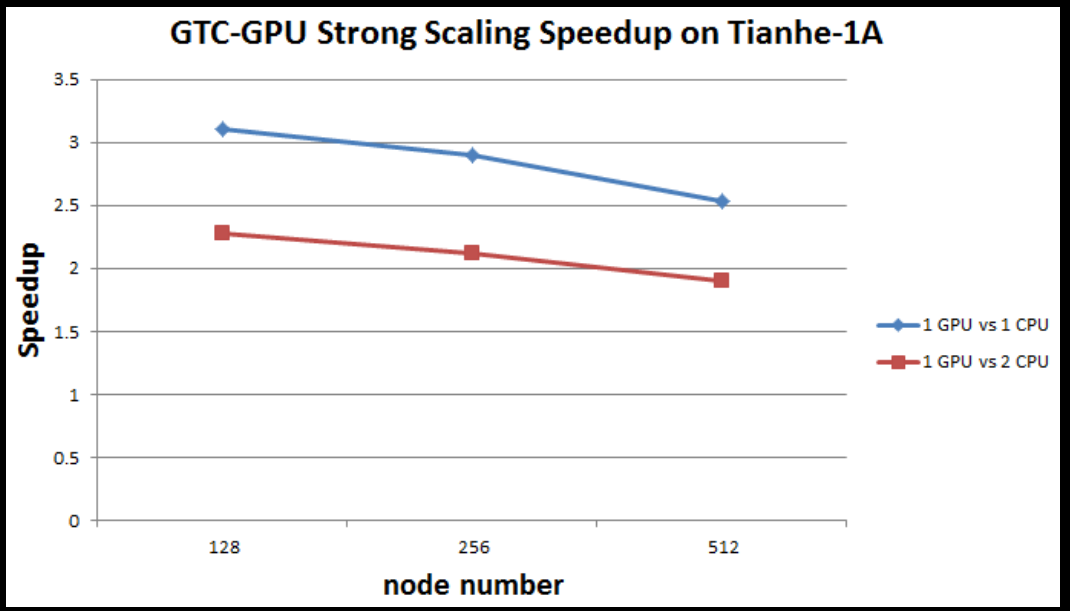
GPU Acceleration Result

	128 CPU	%	128 GPU	%	speedup
loop	521.43		167.46		3.1
field	0.63	0.12%	0.66	0.39%	
ion	54.4	10.43%	54.6	32.60%	
shifte	84.6	16.22%	51.8	30.93%	1.6
pushe	365.4	70.08%	44	26.27%	8.3
poisson	4.4	0.84%	4.4	2.63%	
electron other	12	2.30%	12	7.17%	

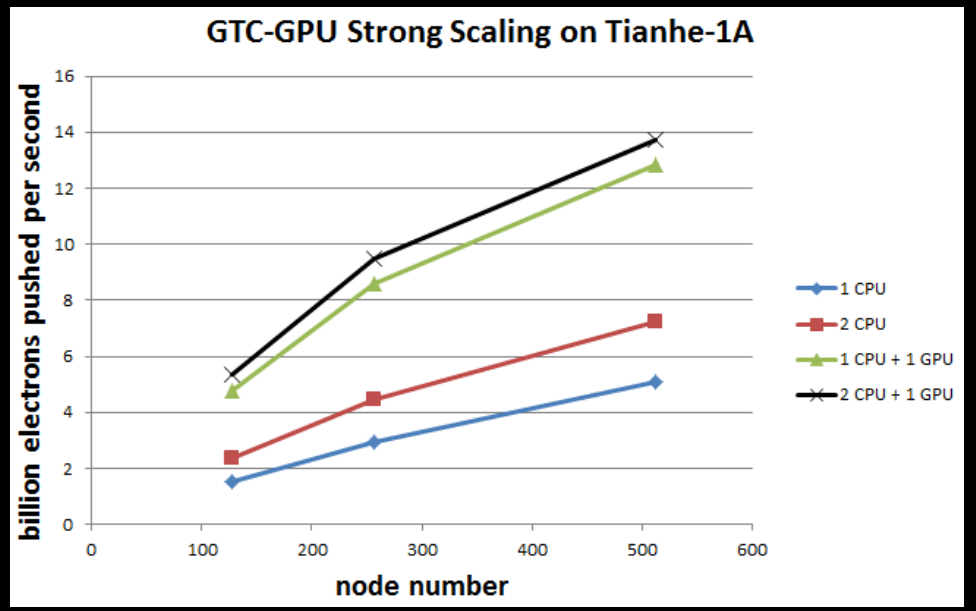
- Computationally heavy pushe got good speedup.
- Porting shifte to GPU is also crucial for CPU-GPU data transfer to not become a bottleneck

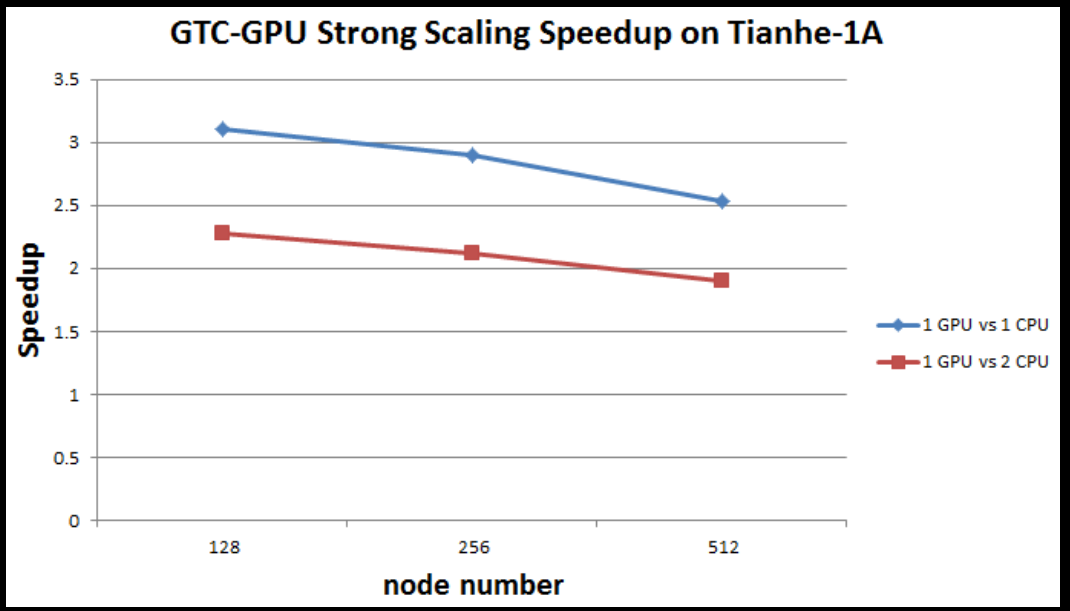
Strong Scaling Results





Strong Scaling Results





Decreasing speedup: two possibilities

- Decreased GPU speedup due to smaller problem
- CPU scaling

Strong Scaling Analysis

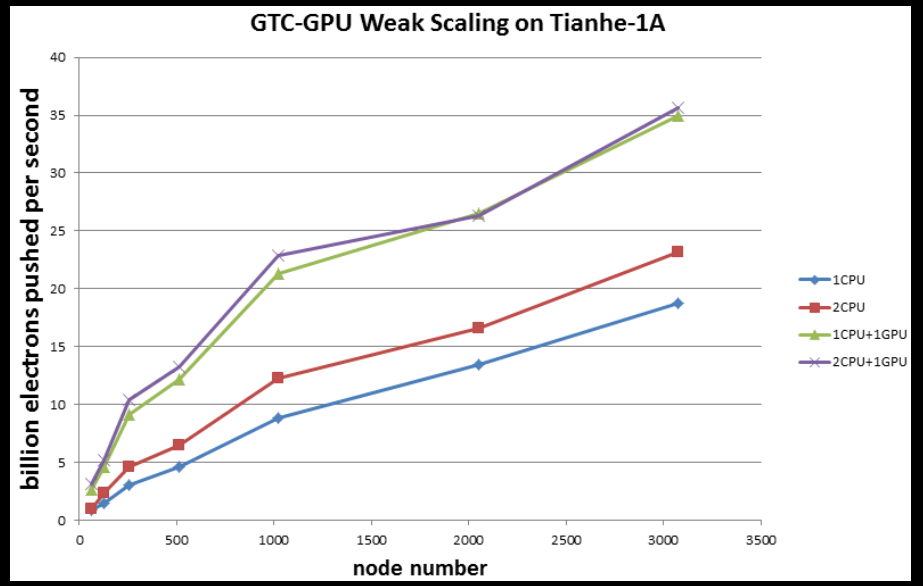
	128 CPU	%	128 GPU	%	speedup
loop	521.43		167.46		3.1
field	0.63	0.12%	0.66	0.39%	
ion	54.4	10.43%	54.6	32.60%	
shifte	84.6	16.22%	51.8	30.93%	1.6
pushe	365.4	70.08%	44	26.27%	8.3
poisson	4.4	0.84%	4.4	2.63%	
electron other	12	2.30%	12	7.17%	

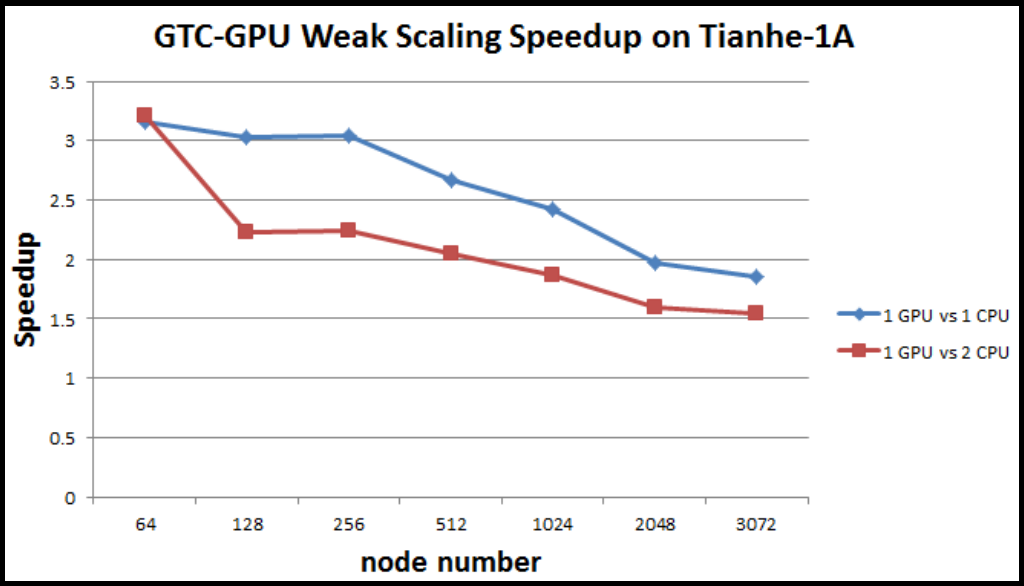
	512 CPU	%	512 GPU	%	speedup
loop	157.83		61.79		2.6
field	0.63	0.40%	0.69	1.12%	
ion	16.3	10.33%	16.2	26.22%	
shifte	31.7	20.08%	17.5	28.32%	1.8
pushe	94.3	59.75%	12.5	20.23%	7.5
poisson	4.9	3.10%	4.9	7.93%	
electron other	10	6.34%	10	16.18%	

Analysis:

- PUSHE+SHIFTE speedup
  - 4.7x vs 4.2x
- PUSHE+SHIFTE profile
  - 86.3% vs 79.8
- Both plays a role

Weak Scaling Results



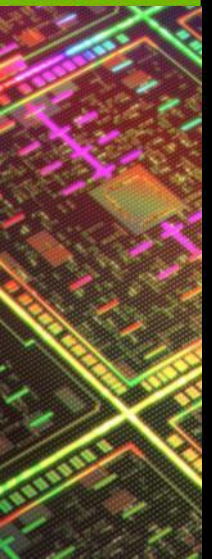


Weak Scaling Analysis

	128 CPU	%	128 GPU	%	speedup
loop	521.43		167.46		3.1
field	0.63	0.12%	0.66	0.39%	
ion	54.4	10.43%	54.6	32.60%	
shifte	84.6	16.22%	51.8	30.93%	1.6
pushe	365.4	70.08%	44	26.27%	8.3
poisson	4.4	0.84%	4.4	2.63%	
electron other	12	2.30%	12	7.17%	

	3072 CPU	%	3072 GPU	%	speedup
loop	699.5	100%	375.4	100%	1.9
field	9.3	1.33%	8.9	2.37%	
ion	79.5	11.37%	79.3	21.12%	
shifte	67.3	9.62%	55	14.65%	1.2
pushe	359.5	51.39%	44.2	11.77%	8.1
poisson	53	7.58%	53	14.12%	
electron other	98.3	14.05%	102.8	27.38%	
diagnosis	32.6	4.66%	32.2	8.58%	

- PUSHE+SHIFTE speedup
  - 4.7x vs 4.3x
- PUSHE+SHIFTE profile
  - 86.3% vs 60.9%
- CPU scaling is the reason



Pushe

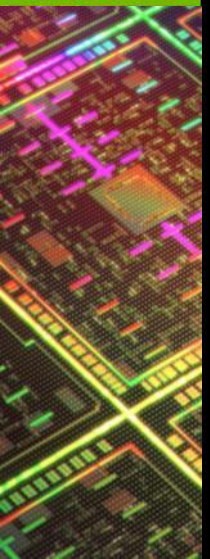
- 2 major kernels
  - Gather fields (gather_fields)
  - Update guiding center position (update_gcpos)
- 1 thread per particle to replace the particle loop “do m=1,me”
- Key optimization technique:
  - Texture prefetch
  - Minimizing data transfer

Bottleneck of pushe

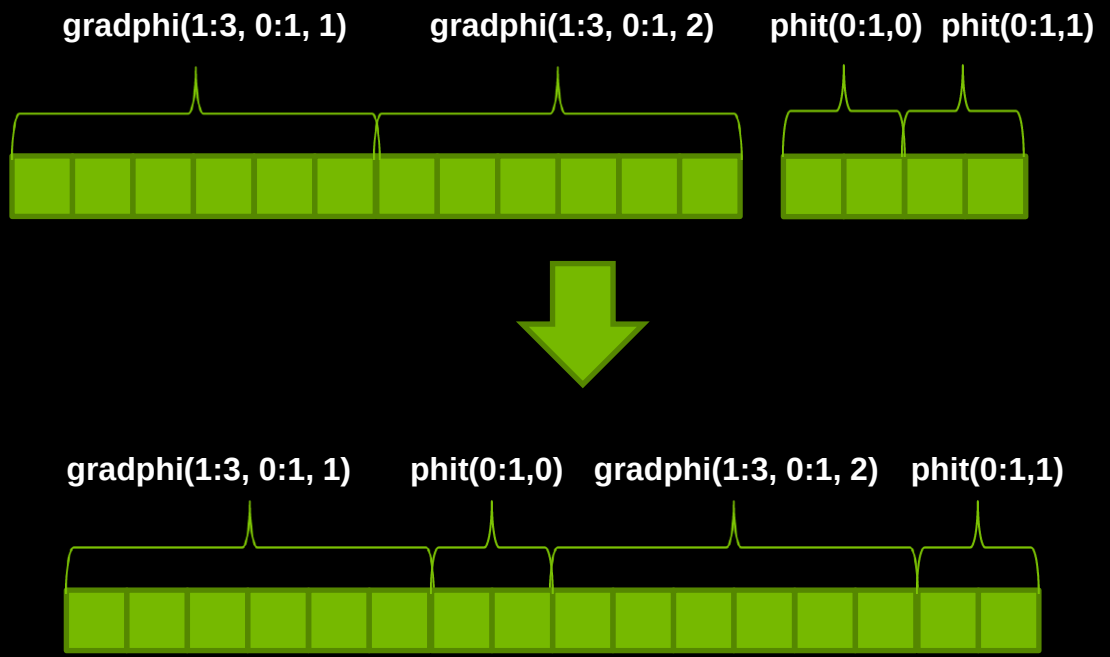
Gather: each particle gather from neighboring cells

```
e1=e1+wt00*(wz0*gradphi(1,0,ij)+wz1*gradphi(1,1,ij))
e2=e2+wt00*(wz0*gradphi(2,0,ij)+wz1*gradphi(2,1,ij))
e3=e3+wt00*(wz0*gradphi(3,0,ij)+wz1*gradphi(3,1,ij))
e4=e4+wt00*(wz0*phit(0,ij)+wz1*phit(1,ij))
```

Uncoalesced access. Binding to texture not working very well: low hit rate.



Improve Texture Hit Rate 1: Data Reorganization



Improve Texture Hit Rate 2: Texture Prefetch

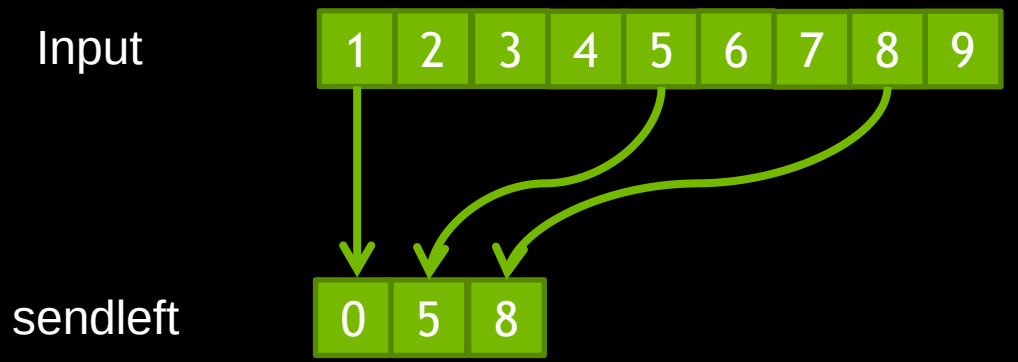
```
e1=e1+wt00*(wz0*gradphi(1,0,ij)+wz1*gradphi(1,1,ij))
e2=e2+wt00*(wz0*gradphi(2,0,ij)+wz1*gradphi(2,1,ij))
e3=e3+wt00*(wz0*gradphi(3,0,ij)+wz1*gradphi(3,1,ij))
e4=e4+wt00*(wz0*phit(0,ij)+wz1*phit(1,ij))
```

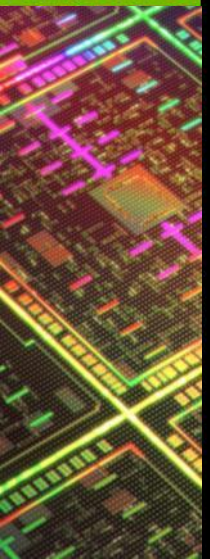
```
float4 tmp1 = tex1Dfetch(texGradphiPhit, ij*2);
float4 tmp2 = tex1Dfetch(texGradphiPhit, ij*2+1);
e1 += wt00*(wz0*tmp1.x + wz1*tmp1.w);
e2 += wt00*(wz0*tmp1.y + wz1*tmp2.x);
e3 += wt00*(wz0*tmp1.z + wz1*tmp2.y);
e4 += wt00*(wz0*tmp2.z + wz1*tmp2.w);
```

Texture cache hit rate goes from 8% to 35%, which gives 3X kernel time speedup

Shifte Problem

- Input:
  - pos[N]: particle position
- Output:
  - sendleft[mleft]: mleft particles that needs to be send to left
  - sendright[mright]: mright particles that needs to be send to right





Shifte GPU Algorithm

- Using Thrust, not very hard to implement a first scan-based version
  - ~10 lines of Thrust code
- Performance not good: ~2X slower than CPU version
  - CPU version only needs to iterate the particle position array once. GPU version needs to iterate the position array many more times.
- Solution: hierarchical scan shifte algorithm
  - Count number of outgoing particles in each warp
  - Scan only the np/32 count arrays
    - Scans a 32 times smaller array, key for higher performance

Conclusion

- GPU is a good fit for the parallelism & data locality in GTC algorithms
  - Texture prefetch for gather operation
  - Hierarchical scan for shift operation

- Demonstrated the advantage of GPU for large scale production fusion simulations