

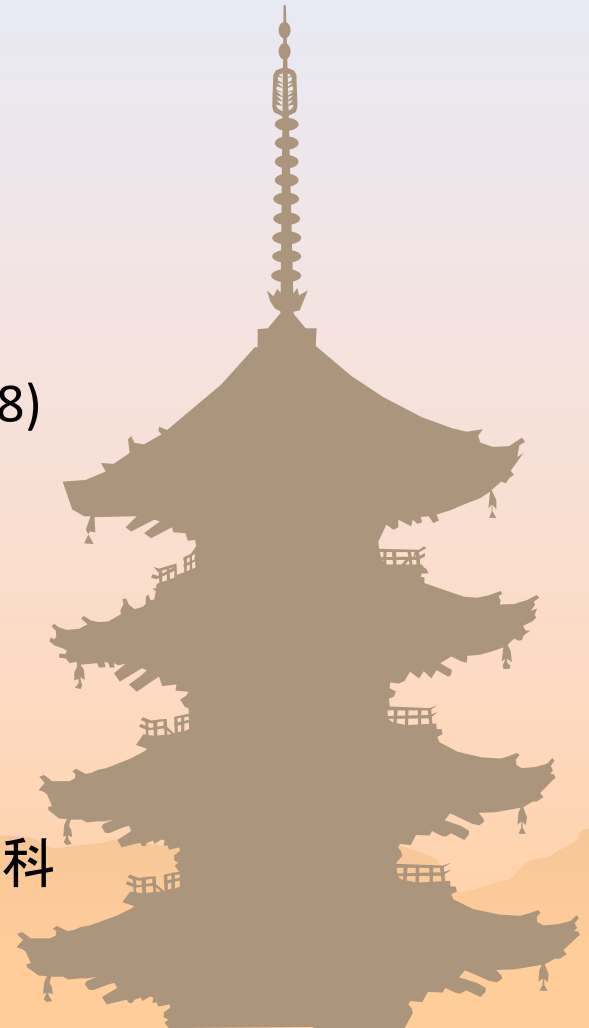
GKNETの移植・最適化報告

発表内容

1. GKNETの紹介 (5/18)
2. GKNETのGPU化 (6/18)
3. サブシステムA/Bでの性能検証 (6/18)
4. まとめ (1/18)

今寺 賢志

京都大学大学院 エネルギー科学研究科



内容

1. GKNETの紹介 (5/18)

2. GKNETのGPU化 (6/18)

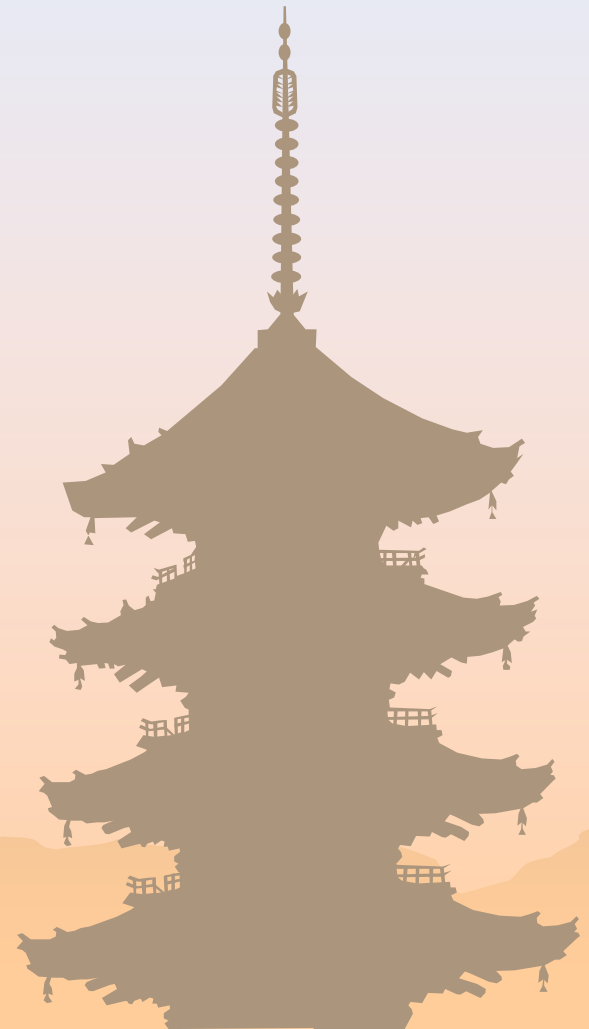
2.1. 基本的なGPUオフロード

2.2. 1D FFTのGPUオフロード

2.3. 実行用のシェルスクリプト

3. サブシステムA/Bでの性能検証 (6/18)

4. まとめ (1/18)



基本的なGPUオフロード

[Vlasov solverのメイン計算]

```
!$omp target teams distribute parallel do collapse(5) private(s_i, u_i, v_i, z_i, x_i, y_i, xy_i, fts)
DO s_i = N_spec_s, N_spec-1
  DO u_i = 3, N_u_p+2
    DO v_i = 4, N_v_p+3
      DO z_i = 3, N_z+2
        DO xy_i = 0, N_x*N_y-1
          y_i = xy_i/N_x + 3
          x_i = mod(xy_i, N_x) + 3
          fts = .....
        END DO
      END DO
    END DO
  END DO
END DO
!$omp end target teams distribute parallel do
```

✓ 原則以下を指定.

- (1) teams distribute: チーム単位の並列化を許可.
- (2) parallel do: スレッド単位の並列化を許可.
- (3) collapse: パフォーマンスのために可能な限りループ統合.
- (4) private: ネスト内のプライベート変数とループインデックス変数を指定.

従来の1D FFT

[Field solverの1D FFT計算(GPU オフロード前)]

```
DO u_i = 3, N_u_p+2
  !$OMP PARALLEL DO PRIVATE(x_i, z_i, y_i, ky_i, rtmp, ctmp)
  DO xz_i = 0, N_x*N_z-1
    z_i = xz_i/N_x + 3
    x_i = mod(xz_i, N_x) + 3

    DO y_i = 3, N_y+2
      rtmp(y_i-3) = rho(x_i, y_i, z_i, u_i)
    END DO

    CALL dfftw_execute_dft_r2c(plan1, rtmp, ctmp)

    DO ky_i = 0, n_max
      rho_ky(x_i, z_i, ky_i, u_i) = ctmp(ky_i)/DBLE(N_y)
    END DO
  END DO
!$OMP END PARALLEL DO
END DO
```

4次元配列を
1次元配列に格納

二重ループの内側で
1D FFTを呼び出し

1D FFT後の1次元配列を
4次元配列に戻す

✓ 4次元配列データに対して1次元方向のみ, FFTWを用いて1D FFTを行っていた.

1D FFTのGPUオフロード - 1

[Field solverの1D FFT計算(GPU オフロード後)]

```
!$omp target teams distribute parallel do collapse(3) private(u_i, xz_i, x_i, z_i, y_i)
```

```
DO u_i = 3, N_u_p+2
```

```
  DO xz_i = 0, N_x*N_z-1
```

```
    DO y_i = 3, N_y+2
```

```
      z_i = xz_i/N_x + 3
```

```
      x_i = mod(xz_i, N_x) + 3
```

```
      rtmp_3d(y_i-3, x_i-3, z_i-3, u_i-3) = rho(x_i, y_i, z_i, u_i)
```

```
    END DO
```

```
  END DO
```

```
END DO
```

```
!$omp end target teams distribute parallel do
```

```
CALL wrapper_dfftw_execute_dft_r2c_many(rtmp_3d, ctmp_3d)
```

バッファ領域をカットした
4次元配列に格納

同時に複数の1D FFTを
FFTWを用いて行う

(続く)

- ✓ Fortran側からCUDA側のラッパー関数をコール.
- ✓ 従来の多重ループ内のFFTWのコールを, ループ分割して外に出し, **同時に複数の1D FFTをFFTWを用いて行う.**

1D FFTのGPUオフロード - 2

(続き)

```
!$omp target teams distribute parallel do collapse(4) private(u_i, xz_i, x_i, z_i, ky_i, tmpr, tmpi)
DO u_i = 3, N_u_p+2
  DO z_i = 3, N_z+2
    DO x_i = 3, N_x+2
      DO ky_i = 0, n_max
        tmpr = dreal(ctmp_3d(ky_i, x_i-3, z_i-3, u_i-3))/DBLE(N_y)
        tmpi = dimag(ctmp_3d(ky_i, x_i-3, z_i-3, u_i-3))/DBLE(N_y)
        rho_ky(x_i, z_i, ky_i, u_i) = dcmplx(tmpr, tmpi)
      END DO
    END DO
  END DO
END DO
!$omp end target teams distribute parallel do
```

1D FFT後の4次元配列をバッファ領域
を持った4次元配列に戻す

[CUDAファイル内のラッパー関数の定義]

```
extern "C" void wrapper_dfftw_execute_dft_r2c_many(double* rtmp, hipDoubleComplex* ctmp){
  hipfftExecD2Z(planfm0, rtmp, ctmp);
  hipDeviceSynchronize();
}
```

✓ コンパイラの都合で、複素数演算は実部と虚部を分けて処理.

実行用のシェルスクリプト - 1

```
[pbs.sh]
#!/bin/bash
#PBS -q B_M    # 使用するジョブキューの指定
#PBS -l select=4:ncpus=96:mpiprocs=4:ngpus=4:mem=124gb
# ノードリソースの指定(select:物理ノード数 ncpus:総CPU数/ノード
  mpiprocs:MPI数/ノード ngpus:GPU数/ノード mem:メモリ量/ノード)

...

module load openmpi/5.0.7/rocm6.3.3_amdflang_afar export
# 必要なモジュールの読み込み
export HSA_XNACK=1
# ROCm向けのメモリ管理を有効化

NP=16    # 使用するMPIプロセス数
MPISEP=./mpisep.sh    # MPI実行用のシェルスクリプト名

BIND="--display-map --report-bindings --cpus-per-rank 24"
# MPIのバインディングオプション

mpirun -n ${NP} $BIND $MPISEP ./run.exe    # 実行コマンド
```

✓ 1APUあたりの全リソースを指定.

実行用のシェルスクリプト - 2

```
[mpisep.sh]
```

```
#!/bin/sh
```

```
touch log.${OMPI_COMM_WORLD_RANK}    # ログファイルの作成
```

```
export ROCR_VISIBLE_DEVICES=$((OMPI_COMM_WORLD_RANK % 4))
```

```
# GPUを4グループに分けて割り当て
```

```
echo "Proc $OMPI_COMM_WORLD_RANK : bind GPU $ROCR_VISIBLE_DEVICES" &>>
```

```
log.${OMPI_COMM_WORLD_RANK}
```

```
# GPUバインド情報をログ出力
```

```
exec $*
```

```
# スクリプトに渡されたコマンドをそのまま実行
```

✓ 使用GPUとプロセスをバインド.

内容

1. GKNETの紹介 (5/18)

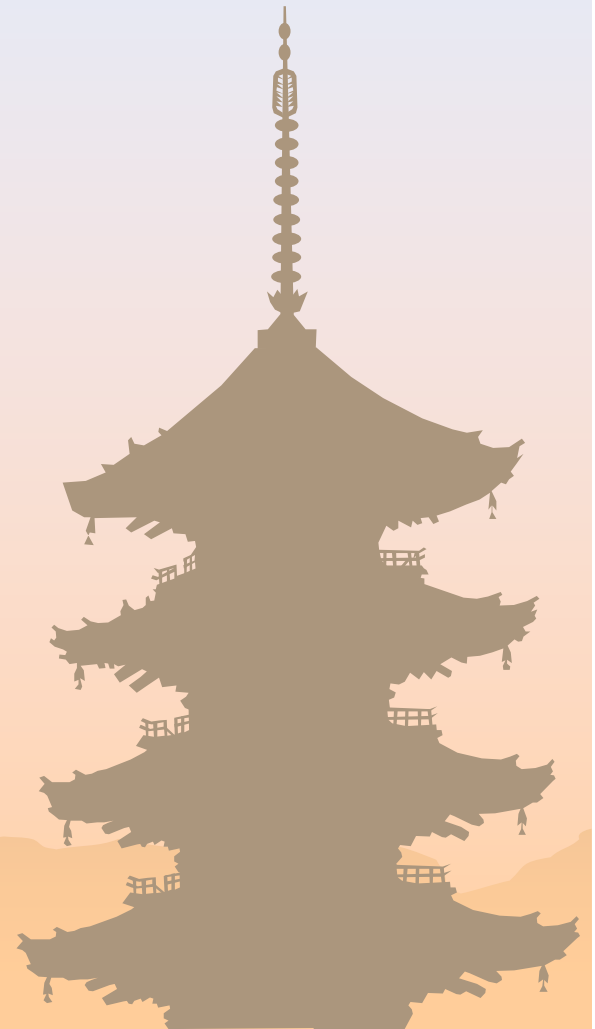
2. GKNETのGPU化 (6/18)

3. サブシステムA/Bでの性能検証 (6/18)

3.1. 静電コードの性能検証

3.2. 電磁コードの性能検証

4. まとめ (1/18)



静電コードの性能計測 - 1

N_x	N_y	N_z	N_v	N_μ
128	64	32	96	16

✓ 2次元(v, μ)領域分割

CPU数とAPU数を揃えた性能比較

システム	ノード数 (CPU or APU)	MPI	SMP	TOTAL	VLASOV	SENDRECV	NEUTRAL
A	2 (4)	128	4	5443.5	3097	1966	380.5
B	1 (4)	4		1774.6	1017	589.7	167.9

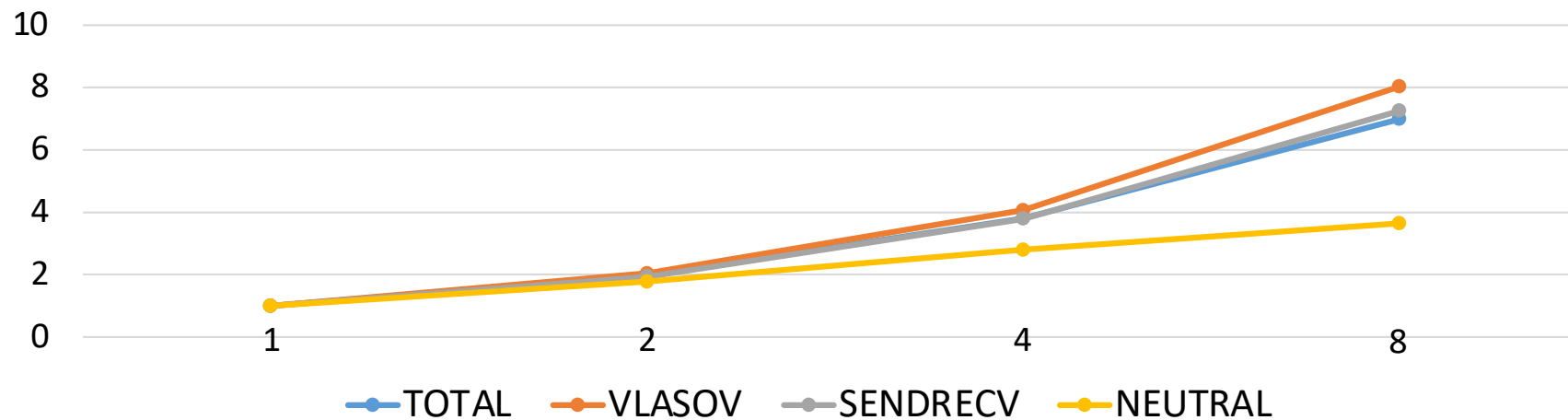
✓ サブシステムA(Xeon GNR)に対してCPU数とAPU数を4で揃えた比較では, 全体で3.06倍.

✓ どのパートという訳ではなく, 全体的に性能は向上している.

静電コードの性能計測 - 2

サブシステムBを想定した実行環境でのスケーリング

ノード数 (APU)	TOTAL	VLASOV	SENDRECV	NEUTRAL
1 (4)	1774.6	1017	589.7	167.9
2 (8)	897.76	499.3	304.1	94.36
4 (16)	465.47	250	155.5	59.97
8 (32)	253.95	126.6	81.3	46.05



- ✓ 全体では1ノード性能を1としたとき, 8ノードに対して7倍程度のスケール.
- ✓ VLASOV区間は理想的なスケール. 通信を含む区間, 特にMPI_Allreduceを含む NEUTRALは若干スケーラビリティが低い.

静電コードの性能計測 - 3

N_x	N_y	N_z	N_v	N_μ
256	64	32	96	16

CPU数とAPU数を揃えた性能比較

システム	ノード数 (CPU or APU)	MPI	SMP	TOTAL	VLASOV	SENDRECV	NEUTRAL
A	4 (8)	256	4	6730.3	3605	2417	708.3
B	2 (8)	8		1895.4	1062	609.4	224

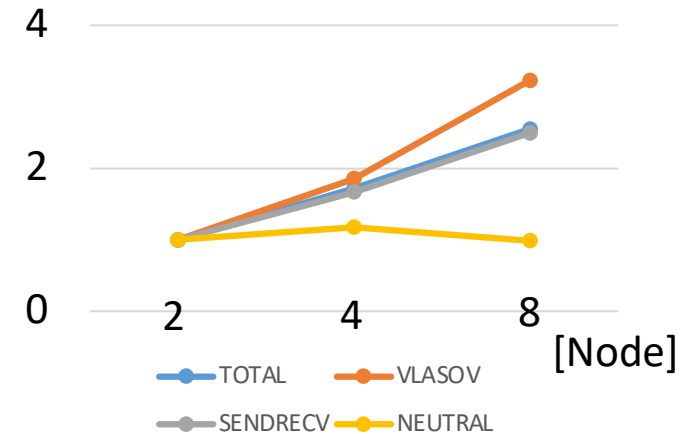
- ✓ サブシステムA(Xeon GNR)に対してCPU数とAPU数を8で揃えた比較では、**全体で3.55倍**.
- ✓ どのパートという訳ではなく、全体的に性能は向上している.

静電コードの性能計測 - 4

サブシステムAを想定した実行環境でのスケーリング

ノード数 (CPU)	TOTAL	VLASOV	SENDRECV	NEUTRAL
2 (4)	11540.7	6682	4026.7	832
4 (8)	6730.3	3605	2417	708.3
8 (16)	4524.2	2067	1612.7	844.5

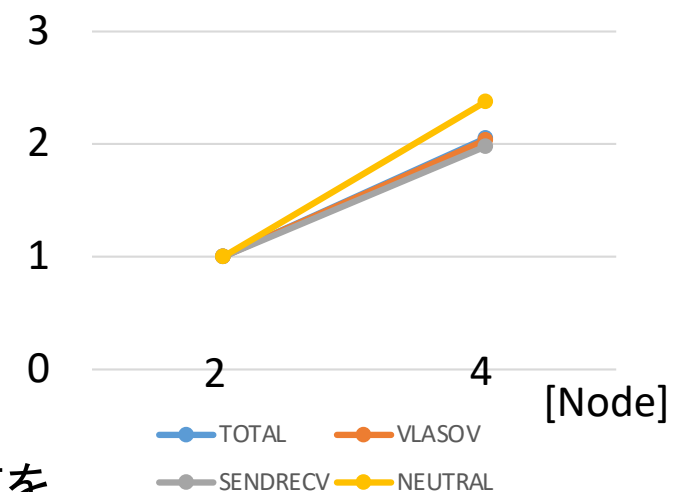
- ✓ NEUTRALのスケーリングが悪い原因は、並列化手法を (x, y, μ) から (v, μ) に変えたことが影響か。



サブシステムBを想定した実行環境でのスケーリング

ノード数 (APU)	TOTAL	VLASOV	SENDRECV	NEUTRAL
2 (8)	1895.4	1062	609.4	224
4 (16)	924.21	521.8	308.08	94.33

- ✓ スケーリングもサブシステムBの方が優れており、理想値を上回っている。



電磁コードの性能計測[実機] - 1

N_x	N_y	N_z	N_v	N_μ
256	64	32	96	16

CPU数とAPU数を揃えた性能比較

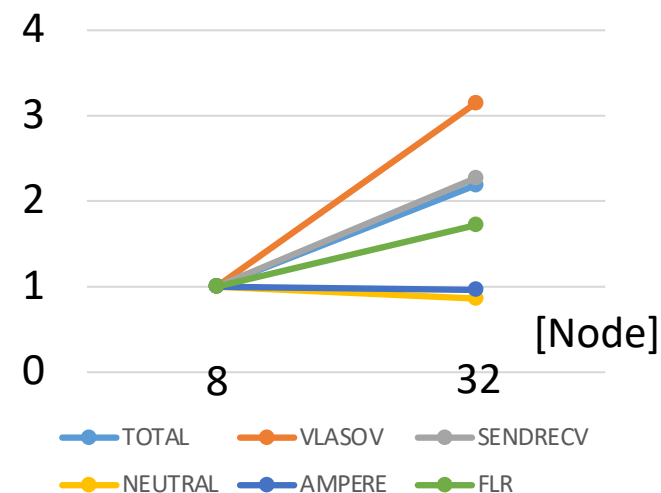
システム	ノード数 (CPU or APU)	MPI	SMP	TOTAL	VLASOV	SEND RECV	NEUTRAL	AMPERE	FLR
A	8 (16)	512	4	5.914	2.691	2.418	0.261	0.360	0.184
B	4 (16)	16		2.741	1.146	0.446	0.241	0.347	0.561

- ✓ サブシステムAに対してCPU数とAPU数を揃えた比較では, 全体で2.15倍.
- ✓ 性能が下がった原因としては, テスト段階では入っていなかったFLRとAMPREEREの最適化が不十分であること, (2)メモリ利用の問題, などが挙げられる.

電磁コードの性能計測[実機] - 2

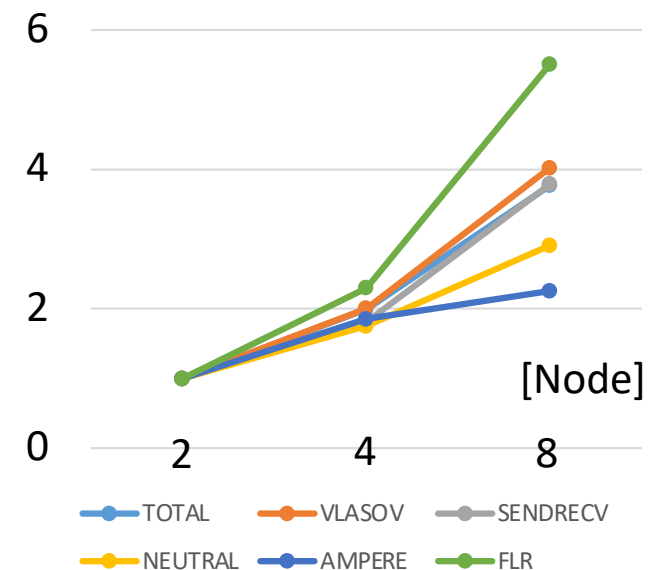
サブシステムAを想定した実行環境でのスケーリング

ノード数 (CPU)	TOTAL	VLASOV	SENDRE CV	NEUTRA L	AMPE RE	FLR
8 (16)	5.914	2.691	2.418	0.261	0.360	0.184
32 (64)	2.704	0.855	1.065	0.303	0.374	0.107



サブシステムBを想定した実行環境でのスケーリング

ノード数 (APU)	TOTAL	VLASO V	SENDR ECV	NEUTR AL	AMPE RE	FLR
2 (8)	5.460	2.298	0.807	0.422	0.643	1.290
4 (16)	2.741	1.146	0.446	0.241	0.347	0.561
8 (32)	1.448	0.571	0.213	0.145	0.285	0.234



✓ スケーリングもサブシステムBの方が優れているが、
静電コードのケースと比較すると若干劣化している。

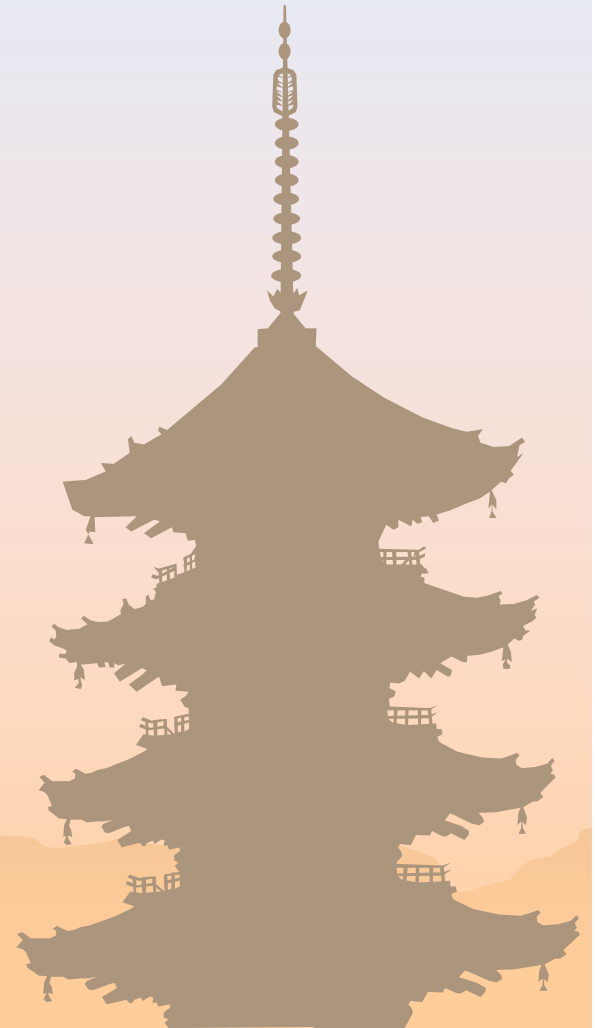
内容

1. GKNETの紹介 (5/18)

2. GKNETのGPU化 (6/18)

3. サブシステムA/Bでの性能検証 (6/18)

4. まとめ (1/18)



まとめ

まとめ

- ✓ GKNETの沿磁力線座標-静電版のGPU化を, NECの皆様に行って頂いた.
- ✓ (1)ループ統合, (2)HIPFFTによる1D FFT計算, (3)FFT計算の一括化, が主たる変更点.
- ✓ 静電コードについては, XeonGNR(サブシステムA)に対してAPU/CPU数を8で揃えたプロダクトランで**3.55倍程度**. スケーリングは極めて良好.
- ✓ 電磁コードについては, サブシステムAに対してAPU/CPU数を16で揃えたプロダクトランで**2.15倍程度**. スケーリングは良好.

今後の改善点

- ✓ 3次元(x, y, μ)領域分割の検討.
- ✓ 新しく導入したサブルーチン(AMPERE, FLR)の最適化.
- ✓ メモリアクセスを意識したコードの改変.
- ✓ 48Vnode以上の計算.