

# vmccal.cの並列化仕様の変更

After

~~SplitMPI(samples)~~

SplitMPI(Ntransfer)

DO sample=x;

~~Wc += w;~~

Wc += w/NSplitSize;

~~Calc\_Energy;~~

Ec=#pragma Calc\_Coulomb

~~#pragma Calc\_Kinetic~~

Ek=#pragma Calc\_Kinetic

~~Calc\_OH~~

Calc\_<OHc> & Calc\_<OHk>

END;

MPI\_AllReduce(Ek,<OH\_k>)

Energy=Ec+Ek,

<OH>=<OH\_c>+<OH\_k>

Before

SplitMPI(samples)

DO sample=x;

Wc += w;

Ene=Calc\_Energy;

#pragma Calc\_Coulomb

#pragma Calc\_Kinetic

Calc\_OH

END;

Calc\_Kinetic関数の1サンプル当たりの計算量(通常は $O(Ntransfer \times Ns)$ )を減らすような並列化。MPI並列数を十分確保でき、バックフローやLanczosといったある一つの関数で異常に時間がかかる場合やロードバランスが悪い場合はFLOPSに対して有効であると考えられる。(計算時間で言えば、並列数が少なければ、変更前の方が早い)

# 1st-Lanczosの場合(Islocgrn.c, vmccal.c)

After(NStoreO=0場合)

~~SplitMPI(samples)~~

SplitMPI(NTransfer)

LSHH\_Store=[\*,\*,\*, LSHkH\_Store]

DO sample=x;

$H(x) = H/NSplitSize$

$HcH(x) = Hc * H/NSplitSize$

$HkH(rank, x) = Calc\_HK$

Calc\_HCA

#pragma Calc\_Coulomb

#pragma Calc\_Kinetic

MPI\_Allreduce(HkH)

LSQQ=[1.0, H, H, HH=HcH+HkH]

END;

Before

SplitMPI(samples)

DO sample=x;

Calc\_HK (LSLocalQ in Islocgrn.c)

Calc\_HCA

#pragma Calc\_Coulomb

#pragma Calc\_Kinetic

END;

1<sup>st</sup>-Lanczosの場合、  
Calc\_Kinetic関数の1サンプル当たりの  
計算量は $O(Ntransfer^2 \times Ns)$ 。

ロードバランスをなるべく崩さずに、  
計算コストをSplit\*Threads分割した。

※zvo\_ls\_qqqq.datに出力される形式は以前のものと同じ

# 1st-Lanczosの場合(Islocgrn.c, vmccal.c)

After(NStoreO=1場合)

~~SplitMPI(samples)~~

SplitMPI(NTransfer)

LSHH\_Store=[\*,\*,\*, LSHkH\_Store]

DO sample=x;

H(x) = H/NSplitSize

HcH(x) = Hc\*H/NSplitSize

HkH(rank, x) = Calc\_HK

Calc\_HCA

#pragma Calc\_Coulomb

#pragma Calc\_Kinetic

LSHH\_Store(rank,x)=[1/NSplitSize,H(x),HcH(x),HkH(rank,x)]

END;

MPI\_Reduce(LSHH\_Store)

If(rank==0) LSHHHH=LSHH\_Store\*LSHH\_Store(詳細は後述)

If(rank==0) CalculateQQQQ\_Store(QQQQ,LSHHHH)

※zvo\_ls\_qqqq.datに出力される形式は以前のものと変更しています

Before

SplitMPI(samples)

DO sample=x;

Calc\_HK (LSLocalQ in Islocgrn.c)

Calc\_HCA

#pragma Calc\_Coulomb

#pragma Calc\_Kinetic

END;

1<sup>st</sup>-Lanczosの場合、

Calc\_Kinetic関数の1サンプル当たりの  
計算量は $O(Ntransfer^2 \times Ns)$ 。

# 1st-Lanczosの場合2(vmccal.c) NStoreO=1ときのみ

$$\text{LSHH\_Store} = \begin{pmatrix} 1.0 & \mathcal{H}(x_0) & \mathcal{H}_C \mathcal{H}(x_0) & \mathcal{H}_K^{\text{rank}} \mathcal{H}(x_0) \\ & \vdots & & \\ 1.0 & \mathcal{H}(x_M) & \mathcal{H}_C \mathcal{H}(x_M) & \mathcal{H}_K^{\text{rank}} \mathcal{H}(x_M) \end{pmatrix}$$

$$A(x) = \frac{\langle x | A | \psi \rangle}{\langle x | \psi \rangle}$$

$$\mathcal{H}_K(x) = [\mathcal{H}_K^0(x) \cdots \mathcal{H}_K^N(x)]$$

M=サンプルサイズ  
N=分割数  
rank=所属ランク

LSHHHH=LSHH\_Store\*LSHH\_Store (StoreOと同じでDGEMMを使う)

$\langle 1 \rangle$	$\langle \mathcal{H} \rangle$	$\langle \mathcal{H}_C \mathcal{H} \rangle$	$\langle \mathcal{H}_K^0 \mathcal{H} \rangle$	$\cdots$	$\langle \mathcal{H}_K^N \mathcal{H} \rangle$	$\langle \mathcal{H}^2 \rangle$
	$\langle \mathcal{H} \cdot \mathcal{H} \rangle$	$\langle \mathcal{H} \cdot \mathcal{H}_C \mathcal{H} \rangle$	$\langle \mathcal{H} \cdot \mathcal{H}_K^0 \mathcal{H} \rangle$	$\cdots$	$\langle \mathcal{H} \cdot \mathcal{H}_K^N \mathcal{H} \rangle$	$\langle \mathcal{H}^3 \rangle$
		$\langle \mathcal{H} \mathcal{H}_C \cdot \mathcal{H}_C \mathcal{H} \rangle$	$\langle \mathcal{H} \mathcal{H}_C \cdot \mathcal{H}_K^0 \mathcal{H} \rangle$	$\cdots$	$\langle \mathcal{H} \mathcal{H}_C \cdot \mathcal{H}_K^N \mathcal{H} \rangle$	
				$\ddots$	$\vdots$	$\langle \mathcal{H}^4 \rangle$
					$\langle \mathcal{H} \mathcal{H}_K^N \cdot \mathcal{H}_K^N \mathcal{H} \rangle$	

# 実行例 : Ns=10x10, 2D Hubbard model (エネルギーのみ)

・実行条件(量子数射影無し)

OpenMP=1

MPI = 192

NSplitSize = 48

NVMCSample = 240

実行システム : maki (System C)

ノード形状: Tofu有, 2x3x2

	Time[s]	MFLOPS[%]	MIPS[%]
Before	14.75	0.0521	15.1762
NStoreO無	17.97	0.2367	11.7398
NStoreO有	10.86	0.3762	14.2272

※プロファイルの詳細は、test/no/profile\_\*.datに記載

・実行条件(量子数射影有、計24)

OpenMP=4

MPI = 48

NSplitSize = 24 (指定がない場合)

NVMCSample = 480

実行システム : maki (System C)

ノード形状: Tofu有, 2x3x2

	Time[s]	MFLOPS[%]	MIPS[%]
Before	19.55	0.5398	7.5812
NStoreO無	24.64	2.2225	10.8576
NStoreO有	24.77	2.2142	10.8480
有, Split80	20.43	2.3126	9.3011
有, Split160	17.26	2.6057	9.6543

※プロファイルの詳細は、test/projection/profile\_\*.datに記載

# バックフローの場合(in vmccal.c) 未実装

Before

```
SplitMPI(samples)
```

```
DO sample=x;
```

```
Mat(x) = Call_Pairing_BF;
```

```
Ene=Calc_Energy;
```

```
#pragma Calc_Coulomb
```

```
#pragma Calc_Kinetic
```

```
x->x'=ci^dagger cj x
```

```
Mat(x')=Update_Mat(x)
```

```
Calc_Inner_Product
```

```
Calc_OH
```

```
END;
```

バックフローの場合、  
Calc\_Kinetic関数の1サンプル当たりの計算量は  
 $O(NTransfer(Nbf \times M \times Ns + M \times Ns^2))$ 。  
Mは更新されたMat(x)の行数。  
Mは、だいたい近接サイト数。  
初めの項はUpdate\_Mat(x)、  
次の項はCalc\_Inner\_Productの計算コスト。

After

```
SplitMPI(samples)
```

```
SplitMPI(NTransfer)
```

```
DO sample=x;
```

```
Calc_inner_product;
```

```
Calc_Energy;
```

```
Ec=#pragma Calc_Coulomb
```

```
Ek=#pragma Calc_Kinetic
```

```
|x>->|x'>=ci^dagger cj |x>
```

```
#pragma Update_Mat(x)
```

```
Calc_Inner_Product
```

```
Calc_OH
```

```
Calc_<OHc> & Calc_<OHk>
```

```
END;
```

```
MPI_AllReduce(Ek,<OH_k>)
```

```
Energy=Ec+Ek
```

```
<OH>=<OH_c>+<OH_k>
```

OpenMPは、どちらか  
のみにかける(普通は上)