

スーパーコンピューティング ニュース

Vol.24 No.2, 2022.3



東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

スーパーコンピュータシステム 利用負担金表

Wisteria/BDEC-01 スーパーコンピュータシステム 利用負担金表(2021 年 5 月 14 日)

コース	負担金額(税込)		ディスク容量	備考
	大学・公共機関等	企業		
一般申込	【Wisteria-O/A】 申込 1 セット当り 60,000 円*1 (8,640 トークン)		グループ 1 セット当り /work 2TB 利用者当り /home 50GB	【Wisteria-O】 最大ノード数 2,304 ノード 【Wisteria-A】 最大 GPU 数 64
公募制度による申込 (Wisteria-A における 「GPU 専有申込」,「ノード 固定」も可能)	【Wisteria-O】 申込 1 セット当り 60,000 円*1 (8,640 トークン)	【Wisteria-O】 申込 1 セット当り 72,000 円*1 (8,640 トークン)	グループ 1 セット当り /work 2TB 利用者当り /home 50GB	
	【Wisteria-A】 申込 1 セット当り 180,000 円*1 (25,920 トークン)	【Wisteria-A】 申込 1 セット当り 216,000 円*1 (25,920 トークン)	グループ 1 セット当り /work 6TB 利用者当り /home 50GB	
GPU 専有申込 (Wisteria-A のみ、公募 制度による申込可能)	【Wisteria-A】 申込 1 セット当り 270,000 円*1 (25,920 トークン)	【Wisteria-A】 申込 1 セット当り 324,000 円*1 (25,920 トークン)	グループ 1 セット当り /work 6TB 利用者当り /home 50GB	1, 2, 4GPU のみ申込可, 申込単位は下表参照
ノード固定 (Wisteria-A のみ、公募 制度による申込可能)	【Wisteria-A】 申込 1 セット当り 2,160,000 円*1 (207,360 トークン)	【Wisteria-A】 申込 1 セット当り 2,592,000 円*1 (207,360 トークン)	グループ 1 セット当り /work 48TB 利用者当り /home 50GB	1 ノード 8GPU 1 年分, 1 セ ットのみ申込可
トークン追加	5,000 円 (720 トークン)	6,000 円 (720 トークン)		
ディスク追加	6,480 円/(1TB*年)			1TB 単位で申込可 (/work のみ)

※パーソナルコースとグループコースの区分を廃止し、これまでのパーソナルコースは一般申込に統合した。

※Wisteria-O のトークン消費係数は 1.00 (1 ノード当り), Wisteria-A のトークン消費係数は 3.00 (1GPU 当り)である。

Wisteria-O にトークン消費係数 1.50 のノード群(優先利用向け)を全体の 15%程度設ける。

※ *1: 利用期間 12 ヶ月の場合の金額。利用期間は 1 ヶ月単位で設定可。

※括弧()内は付与するトークン量。実行したジョブのノード時間積または GPU 時間積と消費係数に応じてトークンが消費される。
付与したトークンは、利用期間内に全量が使用できることを保証するものではない。

トークンは利用期間内に限り有効とし、利用終了後に残量がある場合でも繰越や利用負担金の返還は行わない。

トークンの他のシステムへの移行については、「トークン移行におけるノード時間積の換算表」を参照。

※公募制度による申し込み、ノード固定の申し込みには審査を要する。

※/home のディスク容量は複数のグループに所属している場合でも利用者当り 50GB 固定。

※GPU 専有申込の申込単位

GPU 数	トークン量	大学・公共機関等	企業
1	25,920	270,000 円	324,000 円
2	51,840	540,000 円	648,000 円
4	103,680	1,080,000 円	1,296,000 円

Oakbridge-CX スーパーコンピュータシステム 利用負担金表(2022 年 4 月 1 日)

コース	負担金額(税込)		ディスク容量	備考
	大学・公共機関等	企業		
一般申込	申込 1 セット当り 100,000 円*1 (8,640 ノード時間)	申込 1 セット当り 120,000 円*1 (8,640 ノード時間)	グループ 1 セット当り /work 4TB 利用者当り /home 50GB	最大ノード数 256 ノード
ノード固定	申込 1 セット当り 150,000 円*1 (8,640 ノード時間)	申込 1 セット当り 180,000 円*1 (8,640 ノード時間)		
トークン追加	8,400 円 (720 ノード時間)	10,000 円 (720 ノード時間)		
ディスク追加	6,480 円/(1TB*年)			1TB 単位で申込可 (/work のみ)

※2022 年度よりパーソナルコースとグループコースの区分を廃止し、これまでのパーソナルコースは一般申込に統合する。

※トークン消費係数は 1.00 である。

※ *1: 利用期間 12 ヶ月の場合の金額。利用期間は 1 ヶ月単位で設定可。

※括弧内のノード時間は付与するトークン量。実行したジョブのノード時間積と消費係数に応じてトークンが消費される。

付与したトークンは、利用期間内に全量が使用できることを保証するものではない。

トークンは利用期間内に限り有効とし、利用終了後に残量がある場合でも繰越や利用負担金の返還は行わない。

トークンの他のシステムへの移行については、「トークン移行におけるノード時間積の換算表」を参照。

※ノード固定の申し込みには審査を要する。

※/home のディスク容量はパーソナルコースやグループコースに複数所属していても利用者当り 50GB 固定。

Oakforest-PACS スーパーコンピュータシステム 利用負担金表(2020 年 4 月 1 日)

コース	負担金額(税込)		ディスク容量	備考
	大学・公共機関等	企業		
パーソナルコース	申込 1 セット当り、 最大 6 セットまで 50,000 円*1 (8,640 ノード時間)		申込 1 セット当り /work 1TB 利用者当り /home 50GB	最大ノード数 2,048 ノード
グループコース	申込 1 セット当り 50,000 円*1 (8,640 ノード時間)	申込 1 セット当り 60,000 円*1 (8,640 ノード時間)	グループ 1 セット当り /work 1TB 利用者当り /home 50GB	最大ノード数 2,048 ノード
トークン追加	4,200 円 (720 ノード時間)	5,000 円 (720 ノード時間)		
ディスク追加	6,480 円/(1TB*年)			1TB 単位で申込可 (/work のみ)

※2022 年 3 月 31 日 9:00 をもってすべてのサービスを終了。

※トークン消費係数は 1.00 である。

※ *1: 利用期間 12 ヶ月の場合の金額。利用期間は 1 ヶ月単位で設定可。

※括弧内のノード時間は付与するトークン量。実行したジョブのノード時間積と消費係数に応じてトークンが消費される。

付与したトークンは、利用期間内に全量が使用できることを保証するものではない。

トークンは利用期間内に限り有効とし、利用終了後に残量がある場合でも繰越や利用負担金の返還は行わない。

トークンの他のシステムへの移行については、「トークン移行におけるノード時間積の換算表」を参照。

※/home のディスク容量はパーソナルコースやグループコースに複数所属していても利用者当り 50GB 固定。

トークン移行におけるノード時間積の換算表

移行元 \ 移行先	Oakforest-PACS システム	Oakbridge-CX システム	Wisteria/BDEC-01 システム
Oakforest-PACS システム	—	0.5	0.8
Oakbridge-CX システム	2	—	1.6
Wisteria/BDEC-01 システム	1.2	0.6	—

移行先に追加されるトークン量(ノード時間) = 移行トークン量 × 係数

注意事項(Wisteria/BDEC-01, Oakbridge-CX, Oakforest-PACS スーパーコンピュータシステム 共通)

- ・「大学・公共機関等」は大学、高等専門学校及び大学共同利用機関、文部科学省所管の独立行政法人、学術研究及び学術振興を目的とする国又は地方公共団体が所管する機関、並びに文部科学省科学研究費補助金の交付を受けて学術研究を行う者に適用する。
- ・「企業」の申し込みには、企業利用申込書添付書類の提出および審査を要する。
- ・利用期間は、利用開始月から終了月の末日またはサービス休止前までとする。利用期間内に計算機利用を中止した場合であっても利用負担金額の変更は行わない。年度の途中で利用開始または終了する場合の負担金額は月数別利用負担金表(Web ページ)を参照すること。
- ・前掲の利用負担金表は基本セットの内容であり、最小セットについては Web ページを参照すること。
- ・パーソナルコース(ただし、本センターのスーパーコンピュータシステムに初めて登録された利用者)においてのみ、利用開始月の翌月末日までに利用を中止することができる。利用負担金はパーソナルコースの利用期間1ヶ月の金額を適用し、請求する。
- ・利用負担金は、原則として利用開始月に応じ、以下の月の初旬に一括して請求する。
 - － 利用開始月が4月から7月までは10月、8月から9月までは12月、10月から12月までは3月、1月から3月までは3月末。
 - － 前年度内に事前申込をした分については、利用開始月に関わらず、7月初旬の請求となる。
- ・利用負担金額が減額となる変更はできない。
- ・コース間の変更については、利用負担金が増額になる場合のみ別途相談に応じる。(ただし、利用者番号変更の場合がある。)
- ・グループコースのディスク量は、グループ全体の上限值である。

スーパーコンピュータシステム ジョブクラス制限値

Wisteria/BDEC-01 スーパーコンピュータシステム (Wisteria-O) ジョブクラス制限値 (2021 年 5 月 14 日)

キュー名※1	ノード数※2 (最大コア数)	制限時間 (経過時間)		メモリー 容量 (GiB) ※3	一般申込	公募制度 による申込
		正式運用 期間	試験運用 期間			
debug-o	1 ~ 144 (6,912)	30 分	30 分	28	○	○
short-o	1 ~ 72 (3,456)	8 時間	4 時間	28	○	○
(regular-o)						
small-o	1 ~ 144 (6,912)	48 時間	12 時間	28	○	○
medium-o	145 ~ 576 (27,648)	"	"	"	○	○
large-o	577 ~ 1,152 (55,296)	"	"	"	○	○
x-large-o	1,153 ~ 2,304 (110,592)	24 時間	6 時間	"	○	○
priority-o	1 ~ 288 (13,824)	48 時間	12 時間	28	○	○
challenge-o	1 ~ 7,680 (368,640)	24 時間	-	28	★	★
(interactive-o) ※4						
interactive-o_n1	1 (48)	30 分	30 分	28	○	○
interactive-o_n12	2 ~ 12 (576)	10 分	10 分	"	○	○
prepost	1 (56)	6 時間	3 時間	340	○	○
prepost1_n1 ~ prepost4_n1	1 (56)	1~6 時間	1~3 時間	340	○	○
prepost1_n4	1 ~ 4 (224)	1~6 時間	1~3 時間	340	○	○
prepost1_n8	1 ~ 8 (448)	1~6 時間	1~3 時間	340	○	○

★ 審査による課題選定の上、月1回の一定期間のみ利用可能(原則として月末処理日前日の朝～翌日朝)

※1 キューの指定("#PJM -L "rscgrp=キュー名" ") は, regular-o, debug-o, short-o を小文字で指定する
regular-o キューはノード数の指定("#PJM -L "node=ノード数" ") でノード数別のキューに投入される

※2 トークンの消費係数は1ノード当り1.00。ただし priority-o は優先利用ノード群のためトークン消費係数は1.50

※3 1ノード当りの利用者が利用可能なメモリー容量

※4 インタラクティブジョブの起動は次のとおり (トークン消費なし)

pjsub --interact -g グループ名 -L "rscgrp=interactive-o,node=ノード数"

Wisteria/BDEC-01 スーパーコンピュータシステム (Wisteria-A) ジョブクラス制限値 (2021 年 5 月 14 日)

キュー名※1	ノード数・GPU 数※2 (最大 GPU 数)	制限時間 (経過時間)		メモリー 容量 (GiB) ※3	一般申込	公募制度 による申込	GPU 専有申込	ノード固定
		正式運用 期間	試験運用 期間					
debug-a	1 ノード (8)	30 分	30 分	448	○	○	○	○
short-a	1 ~ 2 ノード (16)	2 時間	1 時間	448	○	○	○	○
(regular-a)								
small-a	1 ~ 2 ノード (16)	48 時間	12 時間	448	○	○	○	○
medium-a	3 ~ 4 ノード (32)	"	"	"	○	○	○	○
large-a	5 ~ 8 ノード (64)	24 時間	6 時間	"	○	○	○	○
share-debug	1, 2, 4 GPU	30 分	30 分	56	○	○	○	○
share-short	1, 2, 4 GPU	2 時間	1 時間	56	○	○	○	○
(share)								
share-1	1 GPU	48 時間	12 時間	56	○	○	○	○
share-2	2 GPU	"	"	"	○	○	○	○
share-4	4 GPU	24 時間	6 時間	"	○	○	○	○
challenge-a	1 ~ 39 ノード (312)	24 時間	-	448	★	★	★	★
任意	1 ノード (8)	任意 ※4	-	448	×	×	○	○
interactive-a ※5	1 ノード (8)	10 分	10 分	56	○	○	○	○
share-interactive	1 GPU	"	"	"	○	○	○	○

★ 審査による課題選定の上、月1回の一定期間のみ利用可能(原則として月末処理日前日の朝～翌日朝)

※1 キューの指定("#PJM -L "rscgrp=キュー名" ") は, regular-a, debug-a, short-a を小文字で指定する
regular-a キューはノード数の指定("#PJM -L "node=ノード数" ") でノード数別のキューに投入される

※2 トークンの消費係数は1GPU 当り3.00

※3 1ノード当りの利用者が利用可能なメモリー容量

※4 申込ノード数の合計以内ならば、キュー名・制限時間(原則 48 時間以内)は相談の上、任意に設定可能

※5 インタラクティブジョブの起動は次のとおり (トークン消費なし)

pjsub --interact -g グループ名 -L "rscgrp=interactive-a,node=ノード数"

Oakbridge-CX スーパーコンピュータシステム ジョブクラス制限値(2019 年 7 月 1 日)

キュー名※1	ノード数※2 (最大コア数)	制限時間 (経過時間)	メモリー 容量 (GB) ※3	パ ー ソ ナ ル コ ー ス	グループ コース	
					申 込 一 般	固 定 ノ ー ド
debug	1 ~ 16 (896)	30 分	168	○	○	○
short	1 ~ 8 (448)	8 時間	168	○	○	○
(regular)						
small	1 ~ 16 (896)	48 時間	168	○	○	○
medium	17 ~ 64 (3584)	"	"	○	○	○
large	65 ~ 128 (7168)	"	"	○	○	○
x-large	129 ~ 256 (14336)	24 時間	"	○	○	○
challenge	1 ~ 1368 (76608)	24 時間	168	★	★	★
任意	申込数	任意 ※4	168	×	×	○
(interactive) ※5						
interactive_n1	1 (56)	2 時間	168	○	○	○
interactive_n8	2 ~ 8 (448)	10 分	"	○	○	○

★ 審査による課題選定の上、月1回の一定期間のみ利用可能(原則として月末処理日前日の朝～翌日朝)

※1 キューの指定("#PJM -L "rscgrp=キュー名" ") は, regular, debug, short を小文字で指定する
regular キューはノード数の指定("#PJM -L "node=ノード数" ") でノード数別のキューに投入される

※2 トークンの消費係数は 1.00

※3 1ノード当りの利用者が利用可能なメモリー容量

※4 申込ノード数の合計以内ならば, キュー名・制限時間(原則 48 時間以内)は相談の上, 任意に設定可能

※5 インタラクティブジョブの起動は次のとおり (トークン消費なし)

pjsub --interact -g グループ名 -L "rscgrp=interactive,node=ノード数"

Oakforest-PACS スーパーコンピュータシステム ジョブクラス制限値(2018 年 4 月 1 日)

キュー名※1	ノード数※2 (最大コア数)	制限時間 (経過時間)	メモリー 容量 (GB) ※3	パ ー ソ ナ ル コ ー ス	グ ル ー プ コ ー ス
debug-cache	1 ~ 128 (8704)	30 分	82	○	○
debug-flat	1 ~ 128 (8704)	30 分	96	○	○
(regular-cache)					
small-cache	1 ~ 128 (8704)	48 時間	82	○	○
medium-cache	129 ~ 512 (34816)	"	"	○	○
large-cache	513 ~ 1024 (69632)	"	"	○	○
x-large-cache	1025 ~ 2048 (139264)	24 時間	"	○	○
(regular-flat)					
small-flat	1 ~ 128 (8704)	48 時間	96	○	○
medium-flat	129 ~ 512 (34816)	"	"	○	○
large-flat	513 ~ 1024 (69632)	"	"	○	○
x-large-flat	1025 ~ 2048 (139264)	24 時間	"	○	○
challenge	1 ~ 8208 (558144)	24 時間	82 / 96	★	★
(interactive-cache) ※4					
interactive_n1-cache	1 (68)	2 時間	82	○	○
interactive_n16-cache	2 ~ 16 (1088)	10 分	"	○	○
(interactive-flat) ※4					
interactive_n1-flat	1 (68)	2 時間	96	○	○
interactive_n16-flat	2 ~ 16 (1088)	10 分	"	○	○
prepost	1 (28)	6 時間	222	○	○

★ 審査による課題選定の上、月1回の一定期間のみ利用可能(原則として月末処理日前日の朝～翌日朝)

※1 キューの指定("#PJM -L "rscgrp=キュー名" ") は, regular-cache/flat, debug-cache/flat を小文字で指定する
regular-cache/flat キューはノード数の指定("#PJM -L "node=ノード数" ") でノード数別のキューに投入される

※2 トークンの消費係数は 1.00

※3 1ノード当りの利用者が利用可能なメモリー容量

※4 インタラクティブジョブの起動は, pjsub --interact -g グループ名 -L "rscgrp=キュー名,node=ノード数"
(キュー名は interactive-cache/flat, トークン消費なし)

センターから

サービス休止等のお知らせ

2022 年 3 月下旬からの計算機サービス予定は以下のとおりです。

Wisteria/BDEC-01 スーパーコンピュータシステム

○ Wisteria/BDEC-01 スーパーコンピュータシステム サービス休止のお知らせ

日 付	利用者サービス	センター内作業
3 月 31 日 (木) ~ 4 月 4 日 (月)	3/31 9:00 ~ 4/4 17:00 までサービス休止	年度末処理
4 月 22 日 (金)	9:00 ~ 22:00 までサービス休止	月末処理
5 月 27 日 (金)	9:00 ~ 22:00 までサービス休止	月末処理

- ・ Wisteria/BDEC-01 システムは、原則 24 時間サービスを行っています。
ただし、月末処理日（原則として毎月最終金曜日）はサービスを停止します。

○ Wisteria/BDEC-01 スーパーコンピュータシステム 大規模 HPC チャレンジのお知らせ (*)

大規模 HPC チャレンジ 実施期間
3 月 30 日 (水) 9:00 ~ 31 日 (木) 9:00 まで 4 月 21 日 (木) 9:00 ~ 22 日 (金) 9:00 まで 5 月 26 日 (木) 9:00 ~ 27 日 (金) 9:00 まで

- ・ 上記期間中、Wisteria/BDEC-01 の debug-o/a, short-o/a, regular-o/a, priority-o, interactive-o/a, prepost, share, share-debug, share-short, share-interactive, ノード固定及び講義用キューのサービスを休止します。
ログインノードは通常どおり利用できます。

Oakbridge-CX スーパーコンピュータシステム

○ Oakbridge-CX スーパーコンピュータシステム サービス休止のお知らせ

日 付	利用者サービス	センター内作業
3 月 31 日 (木) ~ 4 月 4 日 (月)	3/31 9:00 ~ 4/4 17:00 までサービス休止	年度末処理
4 月 20 日 (水)	9:00 ~ 20:00 までサービス休止	月末処理
5 月 25 日 (水)	9:00 ~ 20:00 までサービス休止	月末処理

- ・ Oakbridge-CX システムは、原則 24 時間サービスを行っています。
ただし、月末処理日（原則として毎月最終金曜日）はサービスを停止します。

○ Oakbridge-CX スーパーコンピュータシステム 大規模 HPC チャレンジのお知らせ (*)

大規模 HPC チャレンジ 実施期間
3 月 30 日 (水) 9:00 ~ 31 日 (木) 9:00 まで 4 月 19 日 (火) 9:00 ~ 20 日 (水) 9:00 まで 5 月 24 日 (火) 9:00 ~ 25 日 (水) 9:00 まで

- ・ 上記期間中、Oakbridge-CX の debug, short, regular, interactive, prepost, ノード固定 及び 講義用キューのサービスを休止します。ログインノードは通常どおり利用できます。

Oakforest-PACS スーパーコンピュータシステム【3月31日 9:00 サービス終了】

○ Oakforest-PACS スーパーコンピュータシステム サービス休止のお知らせ

日 付	利用者サービス	センター内作業
3 月 31 日 (木)	9:00 すべてのサービスを終了	サービス終了

- ・ Oakforest-PACS スーパーコンピュータシステムは、原則 24 時間サービスを行っています。
ただし、月末処理日（原則として毎月最終水曜日）はサービスを停止します。

○ Oakforest-PACS スーパーコンピュータシステム 大規模 HPC チャレンジ のお知らせ (*)

大規模 HPC チャレンジ 実施期間
3 月 30 日 (水) 9:00 ～ 31 日 (木) 9:00 まで

- ・ 上記期間中、Oakforest-PACS の regular-flat, regular-cache, debug-flat, debug-cache, interactive-flat, interactive-cache 及び 講義用キューのサービスを休止します。ログインノード、prepost キューは通常どおり利用できます。

【注意事項】

- ・ サービス休止等の計画は原稿作成時の予定です。やむを得ずサービスを変更したり、休止したりする場合がありますので、最新の情報は login 時のメッセージ及びスーパーコンピューティング部門の Web ページの運用スケジュール (<https://www.cc.u-tokyo.ac.jp/supercomputer/schedule.php>) をご確認ください。
- ・ 平日の9:00～17:00以外、休日（土・日・祝日等）は、システム障害等でサービスが停止した場合、運転を継続できない場合があります。その場合は、その時間をもってサービスを中止しますのでご了承ください。
- * Wisteria/BDEC-01, Oakbridge-CX 及びOakforest-PACS における大規模HPC チャレンジについて、新型コロナウイルス感染症の状況次第で実施時間・実施条件の変更や、中止となる可能性があります。詳細はWeb ページ(<https://www.cc.u-tokyo.ac.jp/guide/hpc/>)をご覧ください。

システム変更等のお知らせ

(2022.1.1 - 2022.2.28 変更)

1. ハードウェア

- 1.1 Wisteria/BDEC-01 スーパーコンピュータシステム … なし
- 1.2 Oakbridge-CX スーパーコンピュータシステム … なし
- 1.3 Oakforest-PACS スーパーコンピュータシステム … なし

2. ソフトウェア

2.1 Red Hat Enterprise Linux 8 (Wisteria/BDEC-01)

➤ Odyssey

富士通社製コンパイラ (Fortran77/90/95/2003/2008、C、C++)	1.2.34	(2022.1.28)
cmake	3.22.2	(2022.2.25)

➤ Aquarius

CP2K (GPU 版)	9.1	(2022.1.28)
NVIDIA HPC-X	2.10	(2022.1.28)
CUDA	11.3.1	(2022.2.25)
cmake	3.22.2	(2022.2.25)
MATLAB	R2021b	(2022.2.25)
NVIDIA HPC SDK	v22.2	(2022.2.25)

➤ ログインノード

富士通社製コンパイラ (Fortran77/90/95/2003/2008、C、C++)	1.2.34	(2022.1.28)
cmake	3.22.2	(2022.2.25)
MATLAB	R2021b	(2022.2.25)

➤ プリポストノード

富士通社製コンパイラ (Fortran77/90/95/2003/2008、C、C++)	1.2.34	(2022.1.28)
LXQt	0.17.0-1	(2022.1.28)
cmake	3.22.2	(2022.2.25)

インストールを実施しました。利用方法については、利用支援ポータルのお知らせ、またはドキュメント閲覧より利用手引書をご覧ください。

2.2 Red Hat Enterprise Linux 7, CentOS 7 (Oakbridge-CX)

oneAPI HPC Toolkit HDF5、NetCDF、FFTW	2021.4	(2022.1.28)
HyperWorks	2021.2	(2022.2.25)
MATLAB	R2021b	(2022.2.25)

インストールを実施しました。利用方法については、利用支援ポータルのお知らせ、またはドキュメント閲覧より利用手引書をご覧ください。

3. その他

3.1 Oakforest-PACS サービス終了について

Oakforest-PACS スーパーコンピュータシステムは 2022 年 3 月末をもってすべてのサービスを終了致します。詳細については Web ページ、メール、スーパーコンピューティングニュースをご参照ください。

Oakforest-PACS サービス終了にあたっては以下の点にご注意ください。

- 2022 年 3 月 31 日 9:00 で Oakforest-PACS システムにはアクセスできなくなります。
- Oakforest-PACS において、3 月 31 日 9:00 の時点にて、残トークンがあったとしても 2022 年 3 月 31 日 9:00 以降、残トークンを消費することはできません。
- 3 月末は「大規模共通ストレージシステム(Ipomoea-01)」への移行作業が優先されるため、ログインノードでのファイルアクセス速度の低下、ネットワークの混雑の可能性があります。また、3 月末は混み合う傾向がありますので、ファイルのバックアップ等は早めに実施ください。
- 2 月中旬から 3 月末にかけて、順次各機関の SINET6 への移行が予定されており、Oakforest-PACS との間のネットワーク接続が不安定になる可能性があります。
- Oakforest-PACS の利用者様におきましては、2022 年に稼働する東京大学情報基盤センター「大規模共通ストレージシステム(Ipomoea-01)」へのファイル移行サービスについてのご案内を、最先端共同 HPC 基盤施設(JCAHPC) より別途メールで送らせていただいております。

(注) Ipomoea-01 への移行リクエスト受付は既に終了しております。

(注) 「大規模共通ストレージシステム(Ipomoea-01)」へ移行リクエストを行っていないディレクトリのデータは移行されません。

(注) システム側では利用者データ等のバックアップは取得しておりません。サービス終了後にデータの回復等是对応いたしかねますので、移行リクエストを行っていないディレクトリのデータで必要なデータは各自で 2022 年 3 月 31 日 9:00 までにバックアップを取得ください。

(注) ファイル移行サービス希望者の方が大規模共通ストレージシステム(Ipomoea-01)に置かれたファイルを利用可能となるのは 2022 年 6 月以後の予定です。

- サービス終了後のスーパーコンピュータのご利用につきましては Wisteria/BDEC-01、Oakbridge-CX をご検討ください。

利用負担金の改正について

スーパーコンピューティング部門

2022 年 4 月 1 日より、利用規程別表を改正します。本改正では、Reedbush 及び Oakforest-PACS の削除、Oakbridge-CX スーパーコンピュータシステムにおけるパーソナルコースの廃止などを行います。主な改正点は以下のとおりです。

(1) Reedbush 及び Oakforest-PACS の削除

2021 年 11 月 30 日に運用終了した Reedbush 及び、2022 年 3 月 31 日に運用終了予定の Oakforest-PACS に関する利用負担金別表を削除いたします。

(2) Oakbridge-CX スーパーコンピュータシステムにおけるパーソナルコースの廃止

これまで Oakbridge-CX スーパーコンピュータシステムにおいては、パーソナルコース及びグループコースを設置していましたが、パーソナルコースを廃止し現在のグループコースと同じ扱いといたします。なお、通常利用における名称は「一般申込」といたします。(利用規程別表 1、2)

(3) トライアル利用負担金表の改正

(1)(2)についてトライアル利用負担金表に適用いたします。

最新情報は、本センターWeb ページに掲載の「利用申込書・記入要領」より、各システムおよびコースの利用負担金表をご覧ください。なお、利用規程につきましても、本センターWeb ページに掲載しておりますので併せてご覧ください。

- スーパーコンピュータシステム利用規程 (<https://www.cc.u-tokyo.ac.jp/guide/application/rules.php>)
- 利用申込書・記入要領 (<https://www.cc.u-tokyo.ac.jp/guide/application/guideline.php>)
- トライアル (<https://www.cc.u-tokyo.ac.jp/guide/trial/>)

別表 1 (第 13 条関係)

大規模超並列スーパーコンピュータシステム利用負担金表 (Oakbridge-CX システム)

区 分		利 用 負 担 金 額	
大規模超並列スーパーコンピュータシステム	一般申込	基本セット (年額)	【大学・公共機関等 100,000 円, 企業 120,000 円】 (複数セット申込可, 利用期間は 1 ヶ月単位で設定可)
		トークン量	8,640 トークン (1 ノード, 24 時間×360 日相当, 1 セット当たり)
		トークン消費係数	1.00 ※ ※ 消費係数 1.50 のノード群(優先利用向け)を全体の 15%程度設ける
		ディスク容量	並列ファイルシステム グループにつき 4 TB (1 セット当たり)
		利用者番号登録数	制限なし
	ノード固定 (要審査)	基本セット (年額)	【大学・公共機関等 150,000 円, 企業 180,000 円】 (1 ノード 1 年分, 複数セット申込可, 利用期間は 1 ヶ月単位で設定可)
		トークン量	8,640 トークン (1 ノード, 24 時間×360 日相当, 1 セット当たり)
		トークン消費係数	1.00 ※ ※ 消費係数 1.50 のノード群(優先利用向け)を全体の 15%程度設ける
		ディスク容量	並列ファイルシステム グループにつき 4 TB (いずれも 1 セット当たり)
	ディスク容量追加	並列ファイルシステム	1 TB につき 【6,480 円/年】
	トークン量追加	【大学・公共機関等	8,400 円, 企業 10,000 円】 720 トークン (1 ノード, 24 時間×30 日相当)

(注)上記の利用負担金額は総額表示である。尚, 月数別利用負担金については別表 2 に定める。

最小セット利用負担金表 (Oakbridge-CX システム)

区 分		利 用 負 担 金 額	
小規模超並列システム	一般申込	最小セット	【大学・公共機関等 8,400 円】
		トークン量	720 トークン (1 ノード, 24 時間×30 日相当)
		トークン消費係数	1.00 ※ ※ 消費係数 1.50 のノード群(優先利用向け)を全体の 15%程度設ける
		ディスク容量	並列ファイルシステム グループにつき 4 TB
		利用期間	当該年度末まで
	ディスク容量追加	並列ファイルシステム	1 TB につき 【6,480 円/年】
	トークン量追加	【大学・公共機関等	8,400 円】 720 トークン (1 ノード, 24 時間×30 日相当)

備考

- 「大学・公共機関等」は第 3 条第 1 号, 第 2 号, 第 3 号, 第 4 号, 第 6 号および第 8 号に該当する者に適用する
- 「企業」は第 3 条第 7 号に該当する者に適用する
- 利用期間については利用開始月から当該年度のサービス終了月までとし, 年度を超えないものとする。利用期間の指定がある場合は利用終了月までとする
- トークンはジョブ実行ごとにノード時間積 (経過時間×ノード数) にトークン消費係数を乗じた値を消費するものとし, トークン残量がなくなった場合には, ジョブ実行は抑止される。但し, 計算リソースの状況によって非優先ジョブの実行を許可するものとする

- 5 計算資源を予約する申込においては、予約した期間およびノード数に相当するジョブ実行を行ったものとみなして、トークンを消費するものとする
- 6 トークンは利用期間内に限り有効とし、利用終了後にトークンの残量がある場合でも引継ぎおよび利用負担金の返還は行わない
- 7 付与したトークンは、利用期間内に全量が使用できることを保証するものではない
- 8 トークンの一部を他のスーパーコンピュータシステムへ移行することができる。移行元と移行先でのトークン量の対応については、別表5に定める
- 9 ノード固定は審査によって認められた利用期間およびノード数にて占有利用できるものとする。但し、トークン残量が無くなった時点で占有は解除される
- 10 ノード固定の申込に提供するリソースは、申込状況やシステムの稼働状況により変更する場合がある。変更となった場合の負担金額の適用については申し合わせによるものとする。
- 11 追加オプションは利用期間内に限り有効とする
- 12 追加オプションの負担金額は追加単位額に追加する資源量および利用期間を乗じたものとする
- 13 申込全体のトークン量が提供可能なトークン量の1.2倍を超えない場合に限り受け付けるものとする。但し、企業、若手・女性、大規模HPCチャレンジ等の公募制度による利用、講義・講習会等の教育利用およびトライアルユース等の申込においてはこの限りでない

別表2（第13条関係）

大規模超並列スーパーコンピュータシステム月数別利用負担金（Oakbridge-CX システム）

一般申込 利用負担金表（単位 円）

利用期間	一般申込（1セットにつき）			
	大学・公共機関等	（トークン量）	企業	（トークン量）
12ヶ月	100,000	（8,640）	120,000	（8,640）
11ヶ月	91,700	（7,920）	110,000	（7,920）
10ヶ月	83,400	（7,200）	100,000	（7,200）
9ヶ月	75,000	（6,480）	90,000	（6,480）
8ヶ月	66,700	（5,760）	80,000	（5,760）
7ヶ月	58,400	（5,040）	70,000	（5,040）
6ヶ月	50,000	（4,320）	60,000	（4,320）
5ヶ月	41,700	（3,600）	50,000	（3,600）
4ヶ月	33,400	（2,880）	40,000	（2,880）
3ヶ月	25,000	（2,160）	30,000	（2,160）
2ヶ月	16,700	（1,440）	20,000	（1,440）
1ヶ月	8,400	（720）	10,000	（720）

括弧内は含まれるトークン量(単位はトークン)

ノード固定 利用負担金表(単位 円)

利用期間	ノード固定（1セットにつき）			
	大学・公共機関等	（トークン量）	企業	（トークン量）
12ヶ月	150,000	（8,640）	180,000	（8,640）
11ヶ月	137,500	（7,920）	165,000	（7,920）
10ヶ月	125,000	（7,200）	150,000	（7,200）
9ヶ月	112,500	（6,480）	135,000	（6,480）
8ヶ月	100,000	（5,760）	120,000	（5,760）
7ヶ月	87,500	（5,040）	105,000	（5,040）
6ヶ月	75,000	（4,320）	90,000	（4,320）
5ヶ月	62,500	（3,600）	75,000	（3,600）
4ヶ月	50,000	（2,880）	60,000	（2,880）
3ヶ月	37,500	（2,160）	45,000	（2,160）

2ヶ月	25,000	(1,440)	30,000	(1,440)
1ヶ月	12,500	(720)	15,000	(720)

括弧内は含まれるトークン量(単位はトークン)

追加オプション

トークン量 追加利用負担金表(単位 円)

追加トークン量	大学・公共機関等	(トークン量)	企業	(トークン量)
1 ノード月相当	8,400	(720)	10,000	(720)

括弧内は含まれるトークン量(単位はトークン)

ディスク利用負担金表(単位 円)

並列ファイルシステム (1TBにつき)	
1ヶ月	540

別表 (トライアル実施要領関係)

表1 トライアル利用負担金表 (Oakbridge-CX システム)

通常利用(トライアル)	基本セット 【大学・公共機関等 30,000 円/年】 (最大1セットまで) トークン量 8,640 トークン (1 ノード, 24 時間×360 日相当, 1 セット当たり) トークン消費係数 1.00 ※ ※ 消費係数 1.50 のノード群(優先利用向け)を全体の 15%程度設ける ディスク容量 並列ファイルシステム グループにつき 4TB (1 セット当たり) 利用者番号登録数 制限なし
企業利用(トライアル)	基本セット 【企業 27,000 円/年】 (最大4セットまで) トークン量 8,640 トークン (1 ノード, 24 時間×360 日相当, 1 セット当たり) トークン消費係数 1.00 ※ ※ 消費係数 1.50 のノード群(優先利用向け)を全体の 15%程度設ける ディスク容量 並列ファイルシステム グループにつき 4TB (1 セット当たり) 利用者番号登録数 制限なし
トークン量追加 (新規申込時に付与されたトークン を含め1セットにつき最大12ヶ月 分まで)	【大学・公共機関等 2,500 円, 企業 3,000 円】 720 トークン (1 ノード, 24 時間×30 日相当)

表2 トライアル付与トークン別利用負担金表 (Oakbridge-CX システム)

利用負担金表 (単位 円)

付与 トークン	1 セットにつき		
	大学・公共機関等	企業	(トークン量)
12ヶ月分	30,000	27,000	(8,640)
11ヶ月分	27,500	24,000	(7,920)
10ヶ月分	25,000	21,000	(7,200)

9ヶ月分	22,500	18,000	(6,480)
8ヶ月分	20,000	15,000	(5,760)
7ヶ月分	17,500	12,000	(5,040)
6ヶ月分	15,000	9,000	(4,320)
5ヶ月分	12,500	6,000	(3,600)
4ヶ月分	10,000	3,000	(2,880)
3ヶ月分	7,500	0	(2,160)
2ヶ月分	5,000	0	(1,440)
1ヶ月分	2,500	0	(720)

括弧内は含まれるトークン量, 単位はトークン

大学・公共機関等は1セットまで申込可, 企業は4セットまで申込可

トークン量追加利用負担金表 (単位 円)

追加トークン量	大学・公共機関等	(トークン量)	企業	(トークン量)
1ヶ月分	2,500	(720)	3,000	(720)

括弧内は追加されるトークン量, 単位はトークン

新規申込時に付与されたトークンを含め1セットにつき最大12ヶ月分まで追加可

例：申込時に5ヶ月分申し込んだ場合は7ヶ月分まで追加可

備考

- 1 「大学・公共機関等」は第3条第1号, 第2号, 第3号, 第4号, 第6号および第8号に該当する者に適用する。
- 2 「企業」は第3条第7号に該当する者に適用する。
- 3 利用期間については当該年度のサービス終了日までとする。
- 4 トークンはジョブ実行ごとにノート時間積(経過時間×ノート数)にトークン消費係数を乗じた値を消費するものとする。
- 5 トークンは利用期間内に限り有効とし, 利用終了後にトークンの残量がある場合でも引継ぎおよび利用負担金の返還は行わない。
- 6 付与したトークンは, 利用期間内に全量が使用できることを保証するものではない。
- 7 トークンの他のシステムへの移行はできないものとする。
- 8 トークンの追加は新規申込時点で付与されたトークンも含め, 1セットにつき最大12ヶ月分までとする。

共通ストレージシステム利用規程及び別表の制定について

スーパーコンピューティング部門

大規模共通ストレージシステム(第 1 世代)運用にあたり、東京大学情報基盤センター共通ストレージシステム利用規程及び別表を制定しました。内容については以下の通りとなります。

(1) 共通ストレージシステム利用規程

(趣旨)

第1条 この規程は、東京大学情報基盤センター(以下「センター」という。)が管理・運営する共通ストレージシステム(以下「共通ストレージ」という。)の利用について、必要な事項を定めるものとする。

(利用目的)

第2条 共通ストレージの利用は、学術研究、教育及び社会貢献に供することを目的とする。

(利用資格者)

第3条 共通ストレージの利用資格を有する者は、次の各号に掲げるものとする。

- (1) 大学、高等専門学校及び大学共同利用機関の教職員および学生
- (2) 文部科学省所管の独立行政法人に所属し、専ら研究に従事する者
- (3) 学術研究及び学術振興を目的とする国又は地方公共団体が所管する機関に所属し、専ら研究に従事する者(前号に該当する者を除く。)
- (4) 前号を除く、学術研究を目的とする機関で東京大学情報基盤センター長(以下「センター長」という。)が認める機関に所属し、専ら研究に従事する者
- (5) 第1号から第4号までに該当する者が所属する機関との契約により共同研究に研究分担者として参加し、専ら研究に従事する者
- (6) 科学研究費補助金等の交付を受けて学術研究を行う者(前各号に該当する者を除く。)
- (7) 民間企業その他の法人に所属する者で、別に定める委員会において審査規程に基づく審査の上、センター長が認めた者(第6号に該当する者を除く。)
- (8) 高度情報科学技術研究機構又は学際大規模情報基盤共同利用・共同研究拠点が審査し、選定された課題を実施する者
- (9) 前各号に掲げる者のほか、特にセンター長が認めた者

(利用申込)

第4条 共通ストレージを利用しようとする者は、センター長に所定の利用申込みを行うものとする。

(利用承認)

第5条 前条の利用申込みを受け付け、センター長が利用を認めたときは、これを承認し、承認された者(以下「利用者」という。))に利用者番号を付して「利用登録のお知らせ」を発行するものとする。

(利用者番号の有効期間)

第6条 前条の利用者番号の有効期間は、1年以内とし、当該事業年度を超えることができない。

(利用者番号の転用等禁止)

第7条 利用者は、利用者番号及びパスワードを適切に管理し、不正利用の防止に努めなければならない。

2 利用者は、利用者番号を第2条に規定する利用目的以外のために利用し、又は第三者に利用させてはならない。

(利用者の義務)

第8条 利用者は、共通ストレージの利用に当たっては、本規程を遵守しなければならない。

(施設・設備等の利用)

第9条 利用者は、共通ストレージの利用に当たっては、他に定めがある場合を除き、第2条に規定する利用目的の範囲内において、センターの施設、設備及び物品を使用することができる。

(届出)

第 10 条 利用者は、利用者番号の有効期間内において第4条の申込内容に変更が生じたときは、速やかにセンター長に届け出なければならない。

(利用承認の取消等)

第 11 条 センター長は、共通ストレージの運用に支障をきたすおそれがあると判断した場合、利用者に利用方法の改善を指示することができる。

2 利用者又は第 14 条に規定する支払責任者が、前項の指示に従わない場合、又は次の各号の一に該当したときは、センター長は共通ストレージの利用承認を取り消し、又は利用を停止させることができる。

- (1) 第2条に規定する利用目的以外に共通ストレージを利用したとき
- (2) 第3条の利用資格を喪失したとき
- (3) 第 13 条第1項及び第2項に規定する利用負担金を支払わないとき

(報告書の提出等)

第 12 条 センター長は、利用者に対し、共通ストレージを利用した結果又は経過の報告を求めることができる。利用者は、報告を求められた場合は、センター長に報告するものとする。

2 利用者は、共通ストレージの利用による研究等の成果を論文等により公表するときは、当該論文等にセンターを利用した旨を明記しなければならない。

3 報告書は原則として公開とし、センターの広報等の用に供することができるものとする。ただし、利用者の申出により最大2年間公開を延長することができるものとする。

4 センターは、予め書面による承諾を得ない限り、共通ストレージの利用の報告に際して知り得た利用者の研究上、技術上その他の秘密とすべき情報を厳格に取扱い、センター運用上の目的以外には利用してはならない。

(利用負担金)

第 13 条 第5条により共通ストレージの利用申込みが承認された場合は、利用負担金として別表に定める利用負担金額を次条に規定する支払責任者が支払わなければならない。ただし、第2条に規定する利用目的で、特にセンター長が認めた場合は、利用負担金の全部又は一部を免除することができる。

2 前項に規定する利用負担金の支払は、東京大学内における内部取引に基づく振替又は東京大学の発行する請求書により定められた期日までに指定口座への振込によるものとする。

3 前項により支払われた利用負担金は、原則として返還しない。

(支払責任者)

第 14 条 利用者は、利用負担金の支払の責任を有する者(以下「支払責任者」という。)を利用申込み時にセンター長に届け出なければならない。

2 機関に複数の支払責任者がある場合は、各支払責任者に係る利用負担金を取りまとめて支払う統括支払責任者を別に届け出ることができる。

(免責事由)

第 15 条 センターは、利用者への共通ストレージの提供については最善を尽くすが、通信回線等のネットワーク機器や計算機、記憶装置等の障害による中断・遅延・中止・停止やデータ等の滅失又は毀損が生じないことを保証できない。

2 第 11 条に基づく利用方法の改善指示違反又は利用者自身の操作ミス等の過失等によるデータ等の滅失又は毀損に関わる損害、その他共通ストレージに関連して被った損害について、一切の責任及び負担を負わない。

3 センターは、データ等の漏洩、改ざん防止に努めるが、第三者による不正アクセスやクラッキング行為等によるデータ等の滅失又は毀損・漏洩・改ざん等に関わる損害について、一切の責任及び負担を負わない。

4 他の利用者によるデータ等の保存により共通ストレージ全体の空き容量が不足し、利用者のデータ等保存利用容量が申請時の容量を下回って追加保存ができない場合において、センターは一切の責任を負わない。

(データ等の取扱い)

第 16 条 センターは、利用者が共通ストレージに保存したプログラム、計算に必要なデータ、および計算結果(以下「データ等」という。)及びそれらを格納したファイルの情報に対し、共通ストレージの利用期間中及び利用期間終了後の取扱いを、次の各号に定める。

- (1) センターは、データ等の漏洩、滅失又は棄損を防止するため、必要な予防措置を講じ、適正に管理する。
- (2) センターは、共通ストレージの運用上必要がある場合以外は、利用者のデータ等にアクセスしない。
- (3) センターは、法令等による要請がある場合を除いて、第三者に利用者のデータ等を開示しない。

(4) センターは、共通ストレージの円滑な運用を図るため、ファイル情報を参照することがある。

(5) 第6条による有効期限の終了又は第 11 条による承認の取消の場合においては、利用者のデータ等をセンターが削除できるものとする。

(データ等のバックアップ)

第 17 条 センターは、利用者のデータ等のバックアップを行わないものとする。利用者のデータ等のバックアップに関しては利用者の責任において行うものとする。

(利用の制限)

第 18 条 センターは、利用者への予告無しに共通ストレージを停止することができる。

(補則)

第 19 条 この規程に定めるもののほか、共通ストレージの利用に関し必要な事項は、センター長が別に定める。

附 則

この規程は、令和 4 年 6 月 1 日から適用する。

(2) 大規模共通ストレージシステム(第 1 世代)利用負担金表(別表 1, 2)

別表 1 (第 1 3 条関係)

大規模共通ストレージシステム(第 1 世代)利用負担金表 (Ipomoea-01 システム)

区 分		利 用 負 担 金 額		
大規模共通ストレージシステム(第 1 世代)	一般申込	【大学・公共機関等 7,200 円、企業 8,640 円】(1 TB の場合、年額) (ディスク容量ごとの負担金額は下表参照、利用期間は 1 ヶ月単位で設定可) 利用者番号登録数 制限なし		
		並列ファイルシステム ディスク容量	大学・公共機関等	企業
		1 TB	7,200 円/年	8,640 円/年
		[10 TB まで 1 TB 追加当たり]	[4,200 円/年]	[5,040 円/年]
		10 TB	45,000 円/年	54,000 円/年
		[100 TB まで 1 TB 追加当たり]	[3,000 円/年]	[3,600 円/年]
		100 TB	315,000 円/年	378,000 円/年
		[1,000 TB まで 1 TB 追加当たり]	[2,400 円/年]	[2,880 円/年]
		1,000 TB	2,475,000 円/年	2,970,000 円/年
		[以降 1 TB 追加当たり]	[2,100 円/年]	[2,520 円/年]
	ディスク容量追加	※ 東京大学情報基盤センターのスーパーコンピュータシステムのいずれかに利用者番号(教育利用、講習会を除く)を有する場合、利用者ごとにディスク容量 5 TB を無償で付与し、グループごとに登録されているシステム(トークン移行先のシステムを除く)で付与されているディスク容量の 15%を無償で付与する。いずれも申込不要。		
		申込時点のディスク容量に応じて、1 TB 追加当たりの負担金額は下表参照 (無償で付与されたディスク容量は「申込時点のディスク容量」に含まない)		
		申込時点のディスク容量	大学・公共機関等	企業
		1 TB 未満	7,200 円/年	8,640 円/年
		1 TB 以上 10 TB 未満	4,200 円/年	5,040 円/年
		10 TB 以上 100 TB 未満	3,000 円/年	3,600 円/年
		100 TB 以上 1,000 TB 未満	2,400 円/年	2,880 円/年
		1,000 TB 以上	2,100 円/年	2,520 円/年

(注) 上記の利用負担金額は総額表示である。尚、月数別利用負担金については別表 2 に定める。

備考

- 1 「大学・公共機関等」は第3条第1号、第2号、第3号、第4号、第6号および第8号に該当する者に適用する
- 2 「企業」は第3条第7号に該当する者に適用する
- 3 利用期間については利用開始月から当該年度のサービス終了月までとし、年度を超えないものとする。利用期間の指定がある場合は利用終了月までとする
- 4 ディスク容量は利用期間内に限り有効とし、利用終了後に残存しているデータは削除するものとする
- 5 ディスク容量追加の負担金額は追加単位数に追加する資源量および利用期間を乗じたものとする
- 6 ファイル、ディレクトリの総数制限についてはディスク容量に比例した値を別途定めるものとする

別表2（第13条関係）

大規模共通ストレージシステム(第1世代)月数別利用負担金（lpomoea-01 システム）

一般申込1ヶ月当たり利用負担金表

並列ファイルシステム ディスク容量	一般申込	
	大学・公共機関等	企業
1 TB	600 円/月	720 円/月
[10 TB まで 1 TB 追加当たり]	[350 円/月]	[420 円/月]
10 TB	3,750 円/月	4,500 円/月
[100 TB まで 1 TB 追加当たり]	[250 円/月]	[300 円/月]
100 TB	26,250 円/月	31,500 円/月
[1,000 TB まで 1 TB 追加当たり]	[200 円/月]	[240 円/月]
1,000 TB	206,250 円/月	247,500 円/月
[以降 1 TB 追加当たり]	[175 円/月]	[210 円/月]

追加オプション

ディスク容量追加1ヶ月当たり利用負担金表

申込時点のディスク容量	並列ファイルシステム(1 TBにつき)	
	大学・公共機関等	企業
1 TB 未満	600 円/月	720 円/月
1 TB 以上 10 TB 未満	350 円/月	420 円/月
10 TB 以上 100 TB 未満	250 円/月	300 円/月
100 TB 以上 1,000 TB 未満	200 円/月	240 円/月
1,000 TB 以上	175 円/月	210 円/月

無償で付与されたディスク容量は「申込時点のディスク容量」に含まない

スーパーコンピュータシステム「大規模 HPC チャレンジ」採択課題のお知らせ

1. はじめに

Wisteria/BDEC-01、Oakbridge-CX、Oakforest-PACS スーパーコンピュータシステムでは、「大規模 HPC チャレンジ」を実施しています。「大規模 HPC チャレンジ」は、スーパーコンピュータシステムがもつ最大規模のノード数を、最大 24 時間・1 研究グループで計算資源の専有利用ができる公募型プロジェクトです¹。

課題審査委員会による厳正な審査の結果、以下の課題を採択しましたのでお知らせいたします。

※ 新型コロナウイルス感染症拡大防止に配慮し、通常から一部条件、実施時間等を変更し実施いたしました。

- ・ 実施時間は 8 時間（実施日当日 9:00～17:00）
- ・ Oakforest-PACS スーパーコンピュータシステムにおいては、flat モード（4,200 ノード）、cache モード（3,200 ノード）の各設定の上限までの利用とします（各月 flat モード 1 件、cache モード 1 件の最大 2 件まで受入可能、ただし 1 グループで flat モード、cache モード両方利用することも可能）

2. 採択課題

システム：Oakforest-PACS

募集期間：2021 年度 第 3 回再募集 2021 年 12 月 2 日～12 月 27 日

1 件の応募があり、以下の課題を採択しました。

採択課題一覧

課題名	並列前処理付き反復法ソルバーの最適化および性能評価
代表者名(所属)	中島 研吾（東京大学情報基盤センター）
有限要素法、差分法等の偏微分方程式数値解法は、大規模な疎行列を係数行列とする連立一次方程式を解くことに帰着される。昨今は前処理付きクリロフ部分空間法が幅広いアプリケーションに使用されている。本研究では、代表者の開発した、ICCG 法ソルバーの最適化、性能評価を実施し、Oakforest-PACS (OFP) システムに代表されるメニーコアクラスタにおける前処理付きクリロフ部分空間法の挙動に関する知見を得ることを目的とする。有限要素法による三次元固体力学コード GeoFEM の ICCG ソルバーを対象として、Intel® MPI Library 2019 でサポートされる Asynchronous Progress Threads の機能を使用して、内積における集団通信を計算とオーバーラップさせるパイプライン型共役勾配法的高速化を実施する。更に、メニーコア向け OS 軽量カーネルである IHK/McKernel を適用した評価も併せて実施する。	

¹ 「大規模 HPC チャレンジ」
<https://www.cc.u-tokyo.ac.jp/guide/hpc/>

スーパーコンピュータシステム「大規模 HPC チャレンジ」課題募集のお知らせ

Wisteria/BDEC-01、Oakbridge-CX スーパーコンピュータシステムでは、「大規模 HPC チャレンジ」を実施しています。「大規模 HPC チャレンジ」は、スーパーコンピュータシステムがもつ最大規模のノード数を、最大 24 時間・1 研究グループで計算資源の専有利用ができる公募型プロジェクトです。採択条件等については、以下をご覧ください。皆様からの課題応募をお待ちしております。

※ 新型コロナウイルス感染症の状況次第で実施時間や実施条件の変更、中止の可能性もあります。

1. 提供資源

以下のスーパーコンピュータシステムのノードを最大 24 時間専有利用することができます。

- Wisteria/BDEC-01 スーパーコンピュータシステムのシミュレーションノード群 (Odyssey) 6,144 ノード (294,912 コア)、データ・学習ノード群 (Aquarius) 36 ノード (288 基)
- Oakbridge-CX スーパーコンピュータシステムの計算ノード 1,280 ノード (内 SSD 搭載 112 ノード)

2. 利用案内

- 1 ヶ月に 1 回、原則として月末処理前日の 9:00～翌 9:00 までの最大 24 時間、提供資源を専有利用することが可能です。

(大規模 HPC チャレンジの当日に、チャレンジ用の設定に変更いたします。この変更は 1 時間程度の作業時間が見込まれるため、大規模 HPC チャレンジのご利用時間は 10 時頃開始、翌日 9 時終了となります。)

- 課題は公募制とし、現ユーザーに限定せず、広く課題を募集します。個人、及びグループによる応募が可能です。各月に 1 グループの採用 (*) を原則とします。
- **本制度により得られた成果については公開して頂きます。**成果公開には東京大学情報基盤センターのスーパーコンピュータシステムを利用し、「大規模 HPC チャレンジ」制度によって実施した旨を明記していただきます。また、「スーパーコンピューティングニュース」や広報誌等への成果報告記事の執筆などを行っていただきます。
- センターの主催、共催するセミナー、ワークショップ等でご発表いただく場合があります。
- 利用料金は無料です。

*Wisteria/BDEC-01 においては各月 Odyssey で 1 件、Aquarius で 1 件、最大 2 件まで受入可能、ただし 1 グループで Odyssey、Aquarius 両方利用することも可能

3. 実施日程

2022 年度の今後の「大規模 HPC チャレンジ」実施日程は表 1～2 のとおりです。

※ 新型コロナウイルス感染症の状況次第で、実施時間や実施条件の変更、中止となる可能性もあります。

表 1. 2022 年度 Wisteria/BDEC-01 大規模 HPC チャレンジ実施日程

	実施日時	募集締切	審査	採択通知
第 2 回	2022 年 8 月 18 日(木) 10:00 ～ 19 日(金) 9:00 2022 年 9 月 15 日(木) 10:00 ～ 16 日(金) 9:00 2022 年 10 月 27 日(木) 10:00 ～ 28 日(金) 9:00 2022 年 11 月 24 日(木) 10:00 ～ 25 日(金) 9:00	2022 年 6 月 27 日 (月) 17:00 【締切】	7 月上旬	7 月中旬
第 3 回	2022 年 12 月 22 日(木) 10:00 ～ 23 日(金) 9:00 2023 年 1 月 26 日(木) 10:00 ～ 27 日(金) 9:00 2023 年 2 月 21 日(火) 10:00 ～ 22 日(水) 9:00 2023 年 3 月 30 日(木) 10:00 ～ 31 日(金) 9:00	2022 年 10 月 31 日 (月) 17:00 【締切】	11 月上旬	11 月中旬

表 2. 2022 年度 Oakbridge-CX 大規模 HPC チャレンジ実施日程

	実施日時	募集締切	審査	採択通知
第 2 回	2022 年 8 月 25 日(木) 10:00 ~ 26 日(金) 9:00 2022 年 9 月 21 日(水) 10:00 ~ 22 日(木) 9:00 2022 年 10 月 25 日(火) 10:00 ~ 26 日(水) 9:00 2022 年 11 月 21 日(月) 10:00 ~ 22 日(火) 9:00	2022 年 6 月 27 日 (月) 17:00 【締切】	7 月上旬	7 月中旬
第 3 回	2022 年 12 月 20 日(火) 10:00 ~ 21 日(水) 9:00 2023 年 1 月 24 日(火) 10:00 ~ 25 日(水) 9:00 2023 年 2 月 16 日(木) 10:00 ~ 17 日(金) 9:00 2023 年 3 月 30 日(木) 10:00 ~ 31 日(金) 9:00	2022 年 10 月 31 日 (月) 17:00 【締切】	11 月上旬	11 月中旬

- メンテナンス等の都合により募集スケジュールが変更となることがあります。最新情報は Web Page¹をご覧ください。
- 年複数回を申し込むことも可能ですが、申込状況によりご希望に添えない場合もありますのであらかじめご了承ください。また、一回の申し込みで利用可能なのは一回のみです。
- 表に掲載されている以外の日程でも募集を行うことがあります。最新情報は Web Page¹ をご覧ください。

4. 研究対象

「大規模 HPC チャレンジ」では、提供する最大ノードを使用する大規模計算を実施する研究に限定します。申込者及び研究グループのメンバーは、国内外の並列計算機を利用した大規模計算の実績があることを前提とし、以下のような「High-Performance Computing」に関連した幅広い分野の研究を対象としています。

- 大規模シミュレーション
- 大規模データ処理
- 大規模ベンチマーク、演算・通信システム性能評価
- その他、大規模計算に関係するソフトウェア実行

5. 利用資格

利用資格は、申込書を基に、東京大学情報基盤センタースーパーコンピューティング研究部門の教職員、及び、外部委員により構成される審査委員会において審査されます。現ユーザーである必要はありません。

応募課題は、審査委員会により採択課題を選考し、できるだけ速やかに公表を行う予定です。

なお、申込者は「国内の大学、公共機関に所属する研究者、及び民間企業に所属する者」とします。また、研究グループのメンバー又は申込者が企業の方の場合は、以下の書類のいずれかを提出していただく必要があります。

- 「共同研究契約書の写し」
(申込者の所属機関と共同研究契約を結んでいる研究組織に所属する者)
- 「適切に監督を行うことを記した誓約書及び請負契約書の写し」
(申込者の所属機関と請負契約を結んでいる企業の従業員)
- 「利用規定にある利用目的を遵守することを記した誓約書」
(民間企業に所属している者)

¹ 「大規模 HPC チャレンジ」

<https://www.cc.u-tokyo.ac.jp/guide/hpc/>

6. 採択基準、審査方法

応募課題は、以下の基準により、東京大学情報基盤センタースーパーコンピューティング研究部門の教職員、及び、外部委員より構成される審査委員会により採択課題を選考し、できるだけ速やかに公表を行う予定です。

主な採択基準

- A) 計算・結果の詳細を論文等も含めて公表できること。
- B) 計算結果が科学的に有用、あるいは社会的なインパクトがあると考えられること。
- C) 提供する最大ノード数の利用を目標としていること。
- D) 計画に実現性があり、短期間で効果を示すことが可能であること（一回の使用時間は最大 24 時間です）。
- E) 本システムの運用、ユーザーにとって有用な情報を提供すること。

※ 項目 A) ～ D)は、必須となります。項目 E)は必須ではありませんが、申込書に該当する記述がある場合、加点点評価される場合があります。

7. 利用申込

募集要項、スーパーコンピューターシステム利用規程等をよくお読みの上、利用申込書に必要事項をご記入ください。ご記入頂いた利用申込書は以下の提出先まで、電子メールに添付してお送りください。

申込書類に必要な項目は以下の通りです。

1. 申込年月日
2. 利用希望時期
3. 申込者情報（氏名、所属、職名、連絡先住所、E-mail、電話）
4. 研究課題名（和文、英文）、概要
5. 研究課題の内容、目標
6. 申込者、研究グループメンバーの当該分野における研究業績のうち、大規模計算機利用の実績として代表的な論文 1 編の別刷り
7. プログラム情報、利用スケジュール等
8. 要望事項、特記事項
9. 研究グループメンバーの情報（氏名、所属、職名、研究課題における役割）

8. 問い合わせ先

利用申込等ご不明な点は、電子メールでお問い合わせください。
（電話でのお問い合わせはご遠慮ください。なお、詳細は本センターWeb Page¹でもご案内しておりますので、あわせてご覧ください。）

【 申込書提出先 】

E-Mail : koubo@cc.u-tokyo.ac.jp
東京大学 情報システム部 情報戦略課 研究支援チーム

【 問い合わせ先 】

E-Mail : uketsuke@cc.u-tokyo.ac.jp
東京大学 情報システム部 情報戦略課 研究支援チーム

研究成果の登録のお願い

情報戦略課研究支援チーム
情報基盤センタースーパーコンピューティング部門

研究成果の登録は、本センターのスーパーコンピューターシステムを利用して得られた研究成果のうち、論文、口頭発表、著書、受賞情報についてご報告いただくものです。研究成果の登録は、本センタースーパーコンピューティング部門のWebサイト（<https://www.cc.u-tokyo.ac.jp/>）から「研究成果登録」に進んでください。なお、ご報告いただいた内容は、研究成果データベースへの登録、本センター発行の広報誌及びWebページに掲載させていただきますので、ご了解ください。

研究成果は、東京大学におけるスーパーコンピューターシステムの整備・拡充につながるものとなりますので、利用者の皆様には何卒ご協力くださいますようお願い申し上げます。

① 「研究成果」をクリック
② 研究成果ページの「研究成果登録」をクリック

研究成果登録

研究成果の登録をお願いいたします。

研究成果の登録は、本センターのスーパーコンピューターシステムを利用して得られた研究成果のうち、論文、口頭発表、著書、受賞情報についてご報告いただくものです。ご報告いただいた内容は、研究成果データベースへの登録、センター発行の広報誌及びWebページへの掲載に使用させていただきますことをご承諾ください。研究成果は、東京大学におけるスーパーコンピューターシステムの整備・拡充につながるものとなりますので、利用の皆様には何卒ご協力の程宜しくお願い申し上げます。

お使いのアカウント名と登録メールアドレスを入力し、登録しようとする業績を選択してください。

JavaScriptを有効にしてお使いください。

成果登録内容の削除などの依頼、登録メールアドレスが不明といった際のお問い合わせは

Email: kenkyu_shien.adm@gs.mail.u-tokyo.ac.jp までお願い致します。

東京大学/情報基盤センター/スーパーコンピューティング部門
Supercomputing Division, Information Technology Center, The University of Tokyo

12・1月のジョブ統計

1. Wisteria/BDEC-01スーパーコンピュータシステム(Odyssey) ジョブ処理状況 (RedHat Enterprise Linux 8)

年月	登録者数	実利用者数	処理件数				ファイル使用量 [GiB]		接続時間 [時間]	演算時間 [ノード時間] (経過時間)			平均ノード 利用数 (ノード)	ノード 利用率 (%)
			ログイン	ブリポスト	インタラクティブ ジョブ	バッチジョブ	/home	/lustre		ログイン (実CPU)	ブリポスト	インタラクティブ ジョブ	バッチジョブ	
2021年7月	860	206	3,348	54	188	6,718	148	66,121	13,623	697.32	40	35	282,715	822.0
6月	974	270	5,845	97	186	12,367	361	189,018	23,491	1,398.60	122	36	473,446	842.5
7月	1,103	229	4,638	66	64	10,730	438	211,696	24,723	1,670.10	74	10	698,405	1,492.2
8月	939	147	2,212	82	28	4,551	383	128,337	9,866	2,218.48	255	6	123,322	219.4
9月	1,058	196	3,835	76	150	20,431	434	246,063	19,769	1,944.79	281	40	204,279	289.0
10月	1,193	311	5,221	256	226	12,759	580	396,240	27,589	836.51	284	85	351,435	480.9
11月	1,279	256	5,337	76	311	7,340	774	515,631	26,676	3,092.91	86	91	147,549	208.8
12月	1,286	298	6,189	76	366	9,752	837	833,732	31,952	1,879.83	123	64	672,591	920.1
2022年1月	1,323	294	7,329	92	220	27,108	961	979,806	33,550	1,928.47	256	60	801,198	1,096.0
合計			43,954	875	1,739	111,756			211,239	15,665	1,521	427	3,754,940	

・試運転は、2021年5月14日より開始。正式サービスは、2021年8月2日より開始

・接続時間： ログイン時間の累計

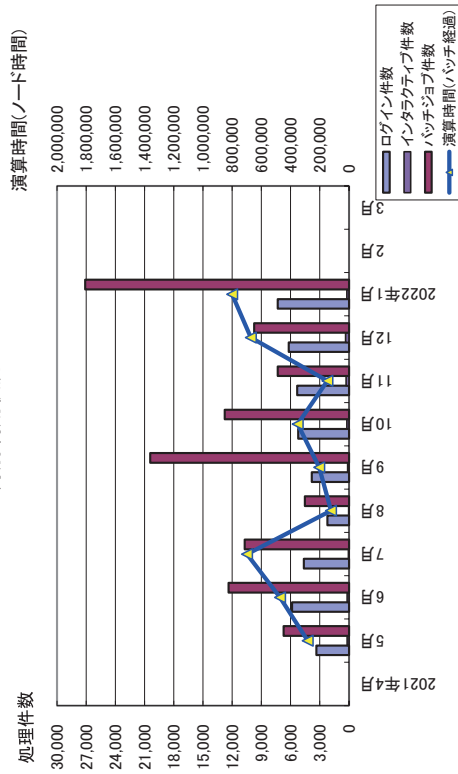
・ログイン(実CPU)： コア時間単位

・ノード利用数： インタラクティブおよびバッチジョブの経過時間を1ノードが100%動作したと仮定した場合の利用ノード数。

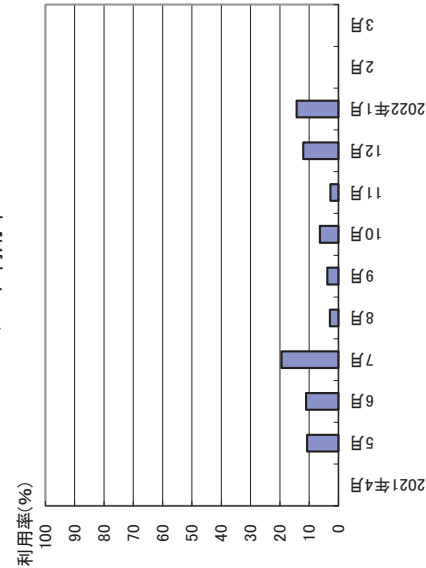
計算式＝1ヶ月のインタラクティブおよびバッチジョブ経過時間合計÷1ヶ月の稼働時間

・ノード利用率： サービスノードに対する利用比率。 計算式＝ノード利用数÷サービスノード数×100

月別利用状況



ノード利用率



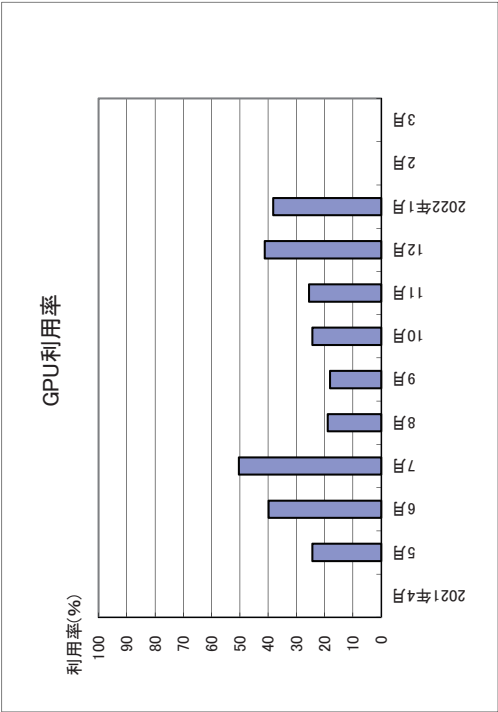
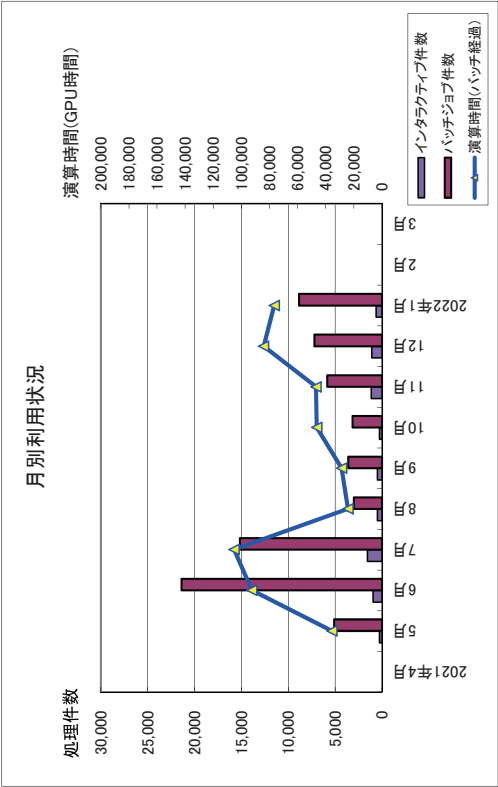
2. Wisteria/BDEC-01スーパーコンピュータシステム (Aquarius) ジョブ処理状況 (RedHat Enterprise Linux 8)

年月	処理件数		演算時間 (GPU時間)(経過時間)		平均GPU 利用数 (GPU)	GPU 利用率 (%)
	インタラクティブ ジョブ	バッチジョブ	インタラクティブ ジョブ	バッチジョブ		
2021年1月	252	5,095	158	35,645	87.5	24.3
6月	953	21,394	1,245	92,643	143.3	39.8
7月	1,540	15,155	1,531	105,239	180.9	50.3
8月	466	3,004	470	24,188	67.9	18.9
9月	480	3,599	358	28,702	65.1	18.1
10月	272	3,139	91	46,327	87.5	24.3
11月	1,122	5,824	1,107	46,895	91.9	25.5
12月	1,077	7,209	957	84,187	148.5	41.2
2022年1月	615	8,846	623	76,630	137.7	38.2
合計	6,777	73,265	6,540	540,456		

・試運転は、2021年5月14日より開始。正式サービスは、2021年8月2日より開始

・GPU利用数： インタラクティブおよびバッチジョブの経過時間を(GPUが100%動作したと仮定した場合)の利用GPU数。
計算式＝1ヶ月のインタラクティブおよびバッチジョブ経過時間合計÷1ヶ月の稼働時間

・GPU利用率： サービスGPUに対する利用率。計算式＝GPU利用数÷サービスGPU数×100



3. Oakbridge-CX スーパーコンピュータシステムジョブ処理状況 (Red Hat Enterprise Linux 7, CentOS 7)

年月	登録者数	実利用者数	処理件数				接続時間 [時間]		ファイル使用量 [GiB]		ログイン (実CPU)	演算時間 [ノード時間] (経過時間)		平均ノード 利用率 (ノード)	ノード 利用率 (%)
			ログイン	プリポスト	インタラクティブ ジョブ	バッチジョブ			/home	/work		プリポスト	インタラクティブ ジョブ		
2021年4月	1,070	323	6,405	2	927	24,498	27,351	913	1,339,482	1,242,42	0	385	474,088	769.1	56.2
5月	1,196	299	6,395	0	982	60,233	38,699	874	1,292,853	8,996,00	0	992	472,558	704.1	51.5
6月	1,087	358	8,649	43	1,356	79,376	48,740	993	1,357,480	2,853,51	1	1,093	547,424	832.8	60.9
7月	1,132	366	8,545	42	1,860	62,733	57,857	1,017	1,430,169	16,807,20	2	674	622,202	909.5	66.5
8月	1,141	278	5,494	59	1,814	31,559	22,299	1,060	1,499,355	4,136,73	5	592	446,153	945.0	69.1
9月	1,067	284	6,700	17	1,585	31,851	32,068	1,095	1,592,894	2,314,47	0	592	586,073	1,011.9	74.0
10月	1,120	305	8,969	29	3,898	72,268	48,565	1,183	1,697,798	3,616,60	3	1,253	715,570	1,057.7	77.3
11月	1,179	348	8,683	0	7,370	63,788	49,289	1,227	1,860,386	3,446,55	0	881	686,231	1,047.8	76.6
12月	1,187	362	9,052	2	763	73,500	43,544	1,377	1,883,279	2,295,56	0	258	686,828	1,016.8	74.3
2022年1月	1,200	375	10,915	0	497	84,721	50,652	1,522	1,941,483	7,899,93	0	262	670,263	994.1	72.7
2021年1月	1,105	215	10,238	49	1,601	64,441	53,307	1,021	1,891,719	3,322,22	253	982	754,704	1,110.5	81.7
2月	1,110	161	7,326	117	2,215	40,336	28,469	1,049	1,963,710	2,864,16	398	893	591,306	941.2	69.2
3月	1,080	170	7,803	67	1,373	63,001	35,810	1,115	1,502,680	2,660,82	387	671	591,775	854.0	62.8
合計			94,936	378	24,640	687,864	483,343		59,134		796	8,546	7,090,471		

・接続時間: ログイン時間の累計

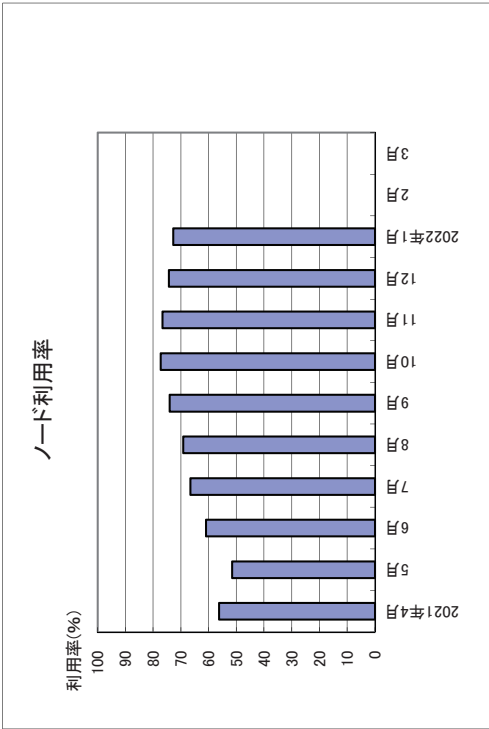
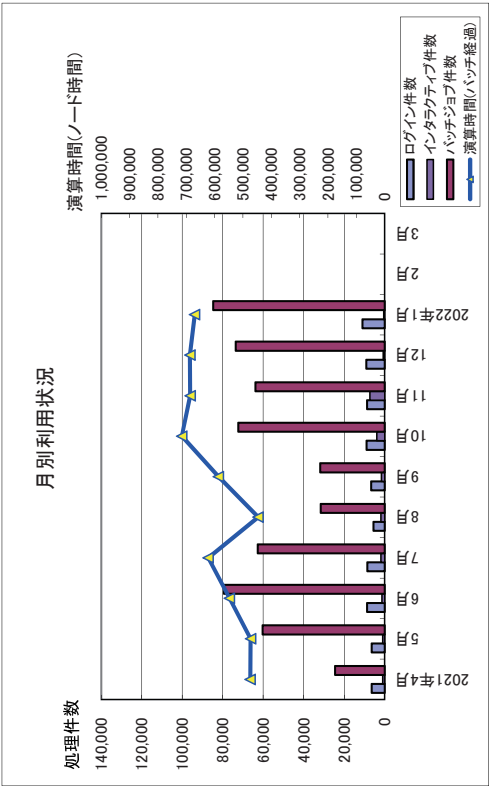
・ログイン(実CPU): コア時間単位

・2021年1月分は合計に含まない

・ノード利用率: インタラクティブおよびバッチジョブの経過時間を1ノードが100%動作したと仮定した場合の利用ノード数。

計算式=1ヶ月のインタラクティブおよびバッチジョブ経過時間合計÷1ヶ月の稼動時間

・ノード利用率: サービスノードに対する利用率。 計算式=ノード利用率÷サービスノード数×100



4. Oakforest-PACスーパーコンピュータシステムジョブ処理状況 (RedHat Enterprise Linux 7, CentOS 7)

年 月	登録者数	実利用者数	処理件数				接続時間 [時間]	ファイル使用量 [GiB]		ログイン (実CPU)	演算時間 [ノード時間] (経過時間)			平均ノード 利用数 (ノード)	ノード 利用率 (%)
			ログイン	ブリポスト	インタラクティブ ジョブ	バッチジョブ		/home	/work		ブリポスト	インタラクティブ ジョブ	バッチジョブ		
2021年4月	1,690	450	8,322	128	181	68,373	47,950	2,618	6,233,823	3,528.98	310	65	2,731,129	4,116.8	50.2
5月	1,732	347	8,252	192	288	63,829	63,987	2,216	4,729,782	2,036.24	411	78	2,977,248	4,089.8	49.8
6月	1,397	378	9,348	274	511	59,097	66,215	2,236	4,830,259	2,088.30	470	92	3,179,605	4,557.0	55.5
7月	1,329	380	8,428	193	415	69,240	74,087	2,199	4,947,221	2,050.97	420	203	3,574,862	4,895.8	59.6
8月	1,336	308	5,994	189	237	33,416	28,905	2,177	5,049,162	5,565.78	523	160	2,981,494	5,366.4	65.4
9月	1,373	319	5,891	184	171	31,112	38,671	2,246	5,093,072	1,940.11	430	108	2,547,260	4,011.9	48.9
10月	1,414	329	6,956	170	113	28,226	59,362	2,212	5,137,733	1,148.12	363	45	2,165,785	2,978.2	36.3
11月	1,429	328	6,928	249	73	79,672	46,522	2,226	5,254,149	9,648.32	372	43	2,509,969	3,571.9	43.5
12月	1,429	359	7,502	207	37	41,964	40,731	2,265	5,113,607	26,901.21	345	17	3,071,995	4,207.9	51.3
2022年1月	1,433	314	6,926	206	123	28,488	55,906	2,266	5,034,724	15,948.13	206	120	2,786,947	3,823.3	46.6
2021年1月	1,804	475	12,636	495	556	210,638	84,101	2,687	6,512,605	3,174.92	787	233	5,112,137	6,993.7	85.2
2月	1,786	431	11,384	314	282	79,481	78,257	2,787	6,581,603	5,791.25	546	197	4,733,131	7,166.3	87.3
3月	1,727	472	12,986	384	302	271,840	102,642	2,760	6,629,043	6,329.17	706	174	5,484,594	7,523.7	91.7
合計			98,917	2,690	2,733	854,738	703,245			82,977	5,102	1,302	38,744,019		

・接続時間：ログイン時間の累計

・ログイン(実CPU)：コア時間単位

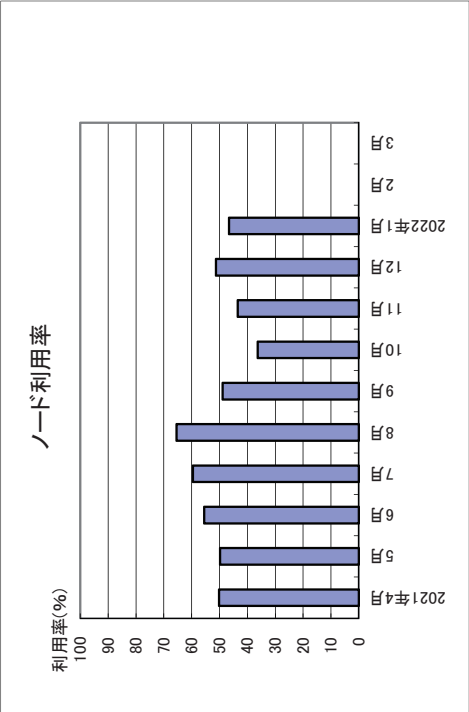
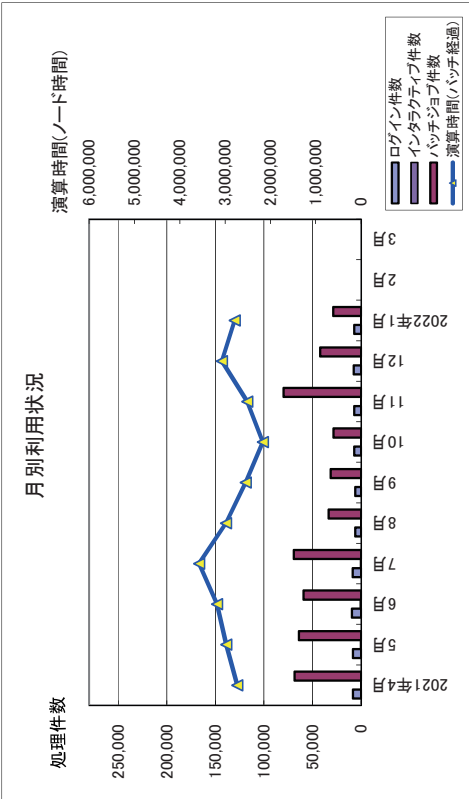
・2021年1月分は合計に含まない

・ノード利用数：インタラクティブおよびバッチジョブの経過時間をノードが100%動作したと仮定した場合の利用ノード数。

計算式＝ノード利用率×総ノード数(8208)

・ノード利用率：サービスノードに対する利用率。計算式＝利用ノード時間÷サービスノード時間×100

・下線部分の値に誤りがございました。お詫びして訂正いたします。



第 41 回 ASE 研究会実施報告

Ode to Numerical Linear Algebra

中 島 研 吾

東京大学情報基盤センター

2022 年 2 月 4 日（金）に開催された第 41 回 ASE 研究会¹について報告する．東京大学情報基盤センターASE 研究会（Advanced Supercomputing Environment）は内外からの講演者をお招きして不定期に開催している．前回の ASE 研究会は第 40 回（2020 年 2 月 4 日）、まさに COVID19 の音を聞きながら開催された．今回は、2 年ぶりに奇しくも同じ 2 月 4 日に第 41 回研究会を開催した．第 41 回 ASE 研究会では、2021 年 12 月から名古屋大学に客員教授として滞在されている Osni Marques 博士（Lawrence Berkeley National Laboratory）が 2 月上旬に帰国される送別の宴をかねてオンラインで開催し、日本滞在中に Marques 博士が交流を持たれた皆様を中心に、数値線形代数（Numerical Linear Algebra）にフォーカスしたやや渋めのプログラム構成とした．登録者合計 59 名、出席者は学内外から合計 54 名であった．2 年ぶりの ASE 研究会は活発な議論で大いに盛り上がった．遠くない将来に対面での研究会が復活することを切に願うものである．

表 1 プログラム

時間帯	講演者	題 目
13:00-13:05	Kengo Nakajima (ITC/The University of Tokyo)	Opening
13:05-13:50	Osni Marques (Lawrence Berkeley National Laboratory)	Solving Eigenvalue Problems in Electronic Structure Calculations (Keynote)
13:50-14:15	Takeshi Iwashita (Hokkaido University)	Algebraic Auxiliary Matrix Construction for Subspace Correction and Deflation in Solving a Series of Linear Systems
14:15-14:40	Takeshi Fukaya (Hokkaido University)	Development of the Mixed Precision GMRES(m) Method using Low Precision Computing
14:55-15:20	Satoshi Ohshima (Nagoya University)	QR Factorization of Block Low-rank Matrices on GPU
15:20-15:45	Tomohiro Suzuki (University of Yamanashi)	Empirical Parameter Tuning of Task-based Tiled Matrix Factorization
15:45-16:10	Takahiro Katagiri (Nagoya University)	Accuracy and Power Optimization by a Directive-based Auto-tuning Language
16:25-16:50	Yen-Chen Chen (The University of Tokyo)	An Efficient Parallel-in-Time Solver for Compressible Fluid Flow Simulation
16:50-17:15	Masatoshi Kawai (ITC/The University of Tokyo)	Low/Adaptive Precision Computation in Preconditioned Iterative Solvers for Ill-Conditioned Problems
17:15-17:40	Kengo Nakajima (ITC/The University of Tokyo)	Communication-Computation Overlapping for Preconditioned Iterative Solvers by Dynamic Loop Scheduling
17:40-17:50	Kengo Nakajima (ITC/The University of Tokyo)	Closing

¹ <https://www.cc.u-tokyo.ac.jp/events/ase/41/41.php>

Wisteria/BDEC-01 利用事例 (3)

データ受け渡しライブラリ h3-Open-SYS/WaitIO (1/2)

住元真司・坂口吉生 (富士通株式会社)

松葉浩也・中島研吾 (東京大学情報基盤センター)

1 はじめに

今回と次回 (2022 年 5 月号)、次々回 (2022 年 7 月号) の 3 回に分けて、Wisteria/BDEC-01 [1,2,3] のようなヘテロジニアスな環境で「計算・データ・学習」融合を実現するためのツールである、h3-Open-SYS/WaitIO [4] と h3-Open-UTIL/MP [5] について紹介する。今回と次回については h3-Open-SYS/WaitIO と h3-Open-UTIL/MP の基本知識と協調動作の事例を中心に紹介し、次々回では応用例として h3-Open-UTIL/MP を活用した GP-GPU 連携を含めた多機能カップラーとして「計算・データ・学習」融合の事例も紹介する。

2 Wisteria/BDEC-01

東京大学情報基盤センターで 2021 年 5 月 14 日に運用を開始した「Wisteria/BDEC-01」[1,2,3] は、「シミュレーション (Simulation)・データ (Data)・学習 (Learning) (S+D+L)」融合を目指す BDEC システム構想 (Big Data & Extreme Computing) に基づくシステムの第 1 号機であり、シミュレーションノード群 (Odyssey (オデッセイ)) とデータ・学習ノード群 (Aquarius (アクエリウス)) の 2 つの計算ノード群を有したシステムである (図 1)。

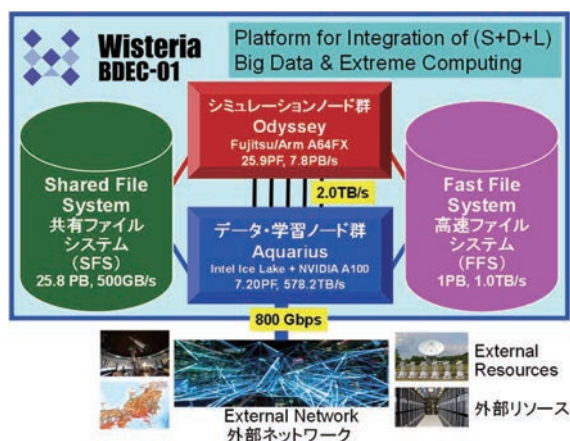


図1 Wisteria/BDEC-01の概要 [1,2,3]

シミュレーションノード群 (Odyssey) は「FUJITSU Supercomputer PRIMEHPC FX1000」20 ラックから構成され、「A64FX」を 7,680 ノード (368,640 コア) 搭載する。各ノードは倍精度浮動小数点演算で 3.3792 TFLOPS の理論ピーク性能を実現する。合計ピーク性能は 25.9 PFLOPS であり、各ノードは 32 GiB の HBM2 メモリを搭載し、シミュレーションノード群 (Odyssey) の総メモリ容量は 240 TiB、総メモリバンド幅は 7.8 PB/秒である。各ノードはバイセクションバンド幅が 13.0 TB/秒のノード間相互結合ネットワーク (Tofu インターコネクト D) で結合されている。

データ・学習ノード群 (Aquarius) 各ノードは汎用 CPU 2 基 (Intel Xeon Platinum 8360Y (Ice Lake), 36core, 2.4GHz), 演算加速装置 (GPU) 8 基 (NVIDIA A100 Tensor コア (SXM4, 40GB)) から構成されており, ノード間インターコネクには NVIDIA Mellanox HDR InfiniBand ネットワークが採用されている。データ・学習ノード群 (Aquarius) の合計ピーク性能は 7.2 PFLOPS, 総メモリ容量は 36.5 TiB, 総メモリバンド幅は 578.2 TB/秒である。各ノードは, データ転送速度が 200 Gbps の帯域を有する InfiniBand HDR を 4 リンク用いて, フルバイセクションバンド幅を持つノード間相互結合ネットワークで結合されている。さらに, 外部接続のために 25 Gbps Ethernet インタフェースも有している。

FEFS (Fujitsu Exabyte File System) による, 共有ファイルシステム (容量: 25.8 PB, データ転送速度: 0.504 TB/秒) および SSD を搭載した高速ファイルシステム (容量: 1.0 PB, データ転送速度: 1.00 TB/秒) を有し, それぞれシミュレーションノード群 (Odyssey), データ・学習ノード群 (Aquarius) からアクセスし, 大規模なデータを高速に処理することが可能である。

シミュレーションノード群 (Odyssey) とデータ・学習ノード群 (Aquarius) は, 合計 160 本の InfiniBand EDR (100Gbps) を用いて 2.0 TB/秒のネットワークバンド幅で結合されている。また, データ・学習ノード群 (Aquarius) は合計 800 Gbps のネットワーク転送速度で外部との通信が可能である。Aquarius の一部のノードは SINET 等の外部ネットワークを介して, サーバー, ストレージ, センサーネットワークを含む様々な外部リソースに直接アクセス可能であり, 観測データをリアルタイムに取り込んで解析, シミュレーションに利用することも可能である。現在は, Odyssey と Aquarius は別々に運用されている。

3 h3-Open-BDEC

3.1 概要

「計算・データ・学習 (S+D+L)」融合のためには, Wisteria/BDEC-01 のようなこれまでにない革新的なハードウェアが必要であるが, 様々なアプリケーション, ワークロードを Wisteria/BDEC-01 上で開発, 実行していくためのソフトウェア群も重要である。

当センターで開発した「ppOpen-HPC (自動チューニング機構を有するアプリケーション開発・実行環境)」[6], 「h3-Open-BDEC (「計算・データ・学習」融合のための革新的ソフトウェア基盤)」[7,8] を利用し, 高性能なアプリケーションを容易に開発することが可能である。

当センターでは, センター内外の計算科学, 計算機科学, 数値アルゴリズム, データ科学, 機械学習の専門家と協力して, エクサスケール時代のスパコンの能力を最大限活用し, 科学的発見を持続的に促進するために, (計算・データ・学習) 融合による革新的シミュレーション手法を提案し, 最小限の計算量・消費電力で融合シミュレーションを実現する研究開発, ソフトウェア基盤実装を実施している [7,8]。本研究では, Wistria/BDEC-01 を (計算・データ・学習) 融合のためのプラットフォームと位置付け, ①変動精度演算・精度保証・自動チューニング (Automatic Tuning, AT) による新計算原理に基づく革新的高性能・高信頼性・省電力数値解法, ②機械学習に基づく革新的手法である階層型データ駆動アプローチ (hDDA) の 2 項目を中心に研究開発を実施し, 革新的ソフトウェア基盤「h3-Open-BDEC」を開発する。(図 2) [7,8]

革新的ソフトウェア基盤「h3-Open-BDEC」(図 2) を構成する h3-Open-MATH (数値アルゴリズム), h3-Open-VER (精度保証), h3-Open-AT (自動チューニング), h3-Open-APP (アプリ開発), h3-Open-DATA (データ科学), h3-Open-DDA (データ駆動アプローチ), h3-Open-UTIL (並列ユ

ーティリティ), h3-Open-SYS (統合・制御) は複数の構成要素を含み, 緊密に関連し, 以下の 3 層を構成している:

- ①「変動精度演算に基づく新計算原理」層 (h3-Open-MATH, h3-Open-VER, h3-Open-AT)
- ②「(計算・データ・学習) 融合」層 (h3-Open-APP, h3-Open-DATA, h3-Open-DDA)
- ③「統合・通信・ユーティリティ」層 (h3-Open-SYS, h3-Open-UTIL)

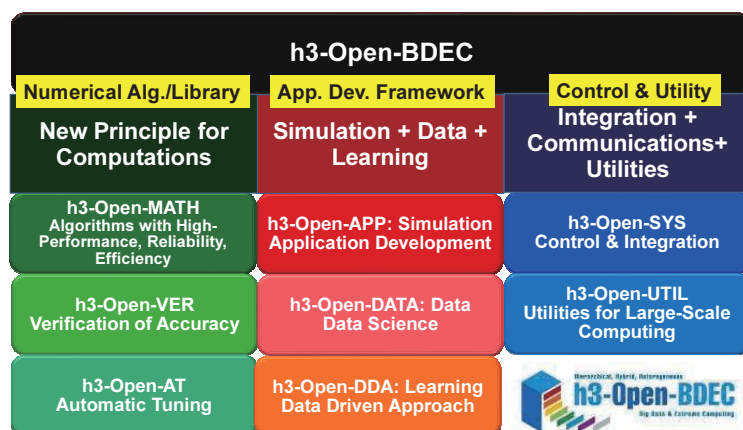


図 2 h3-Open-BDEC の概要 [7,8]

エクサスケールシステムにおける高性能数値アルゴリズム実現には, メモリ・ネットワークの階層の深化に対応した通信最適化 (Serial, Parallel), 省電力・省エネルギーに向けた検討が必要である。数値計算による近似解 (数値解) は様々な計算誤差を含み, 計算結果の信頼性の観点から, 数値解の正しさを数学的に保証する必要がある, 低精度・混合/変動精度使用時, 悪条件問題には重要であるが, 実問題で現れる大規模疎行列・H 行列への応用例はほとんどない。本研究では, 「変動精度演算に基づく新計算原理」確立を目指して, 高性能アルゴリズム, 精度保証, 最適精度選択のための自動チューニング手法の研究開発を実施する。

シミュレーションに機械学習を適用して異なるパラメータでの解を予測するデータ駆動アプローチ (Data Driven Approach, DDA) では, 計算を繰り返して教師データを生成する必要がある。本研究で提案する階層型 DDA (hDDA) は, 特徴検知, MOR (Model Order Reduction), UQ (Uncertainty Quantification), スパースモデリング, 適応格子等の諸機能を駆使して, 計算量 (メッシュ数, 粒子数) を削減した簡易モデルを, 機械学習により自動生成し, 教師データ生成用モデルとして利用する。「統合・通信・ユーティリティ」層は Wisteria/BDEC-01 のようなヘテロジニアスなシステム上で, 「計算・データ・学習」融合を容易に実現するための環境を提供する。

h3-Open-BDEC はエクサスケール時代のスパコンで (計算・データ・学習) 融合を実現する世界初の革新的ソフトウェア基盤であり, 計算科学の専門家のみで (計算・データ・学習) 融合を容易に実現できる。ソースコード, マニュアル類も含めて一般に公開し, 様々な環境で利用できるよう, 普及に努める。h3-Open-BDEC 利用による (計算・データ・学習) 融合シミュレーションにより, 従来手法と同等の正確さを保ちつつ, 大幅な計算量・消費電力削減を目指す。

計算科学シミュレーションは多くの場合, 非線形な問題を扱うため, 多数のパラメータスタディが必要である。Wisteria/BDEC-01 では, 機械学習による最適パラメータ推定を, 外部から取り

込んだ実験・観測データによる同化と組み合わせて、正確な解をより短時間で求めることを目指している。

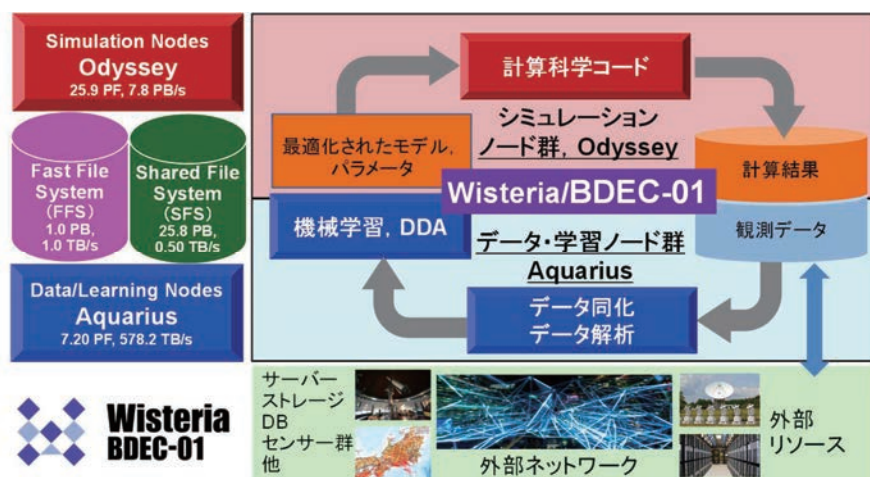


図3 Wisteria/BDEC-01利用による「計算・データ・学習」融合のイメージ [7,8]

図3は、Wisteria/BDEC-01における「計算・データ・学習 (S+D+L)」融合のイメージである。h3-Open-BDECを使用することによって、シミュレーションノード群で計算科学シミュレーションコードを実行し、データ・学習ノード群では外部から取り込んだ観測データや、機械学習による推論等に基づきパラメータを最適化し、更に計算を実施するというサイクルを容易に実現することができ、またパラメータ最適化によって計算時間を全体として短縮できることが期待される。

3.2 h3-Open-SYS

h3-Open-SYSはWisteria/BDEC-01のようなヘテロジニアスなシステムにおいて、シミュレーションとデータ処理実行を統合するソフトウェア群である。h3-Open-SYS/WaitIO [4,7,8]はその中核機能として、ファイルシステムを通じて複数の並列プログラムがデータの受け渡しを行うライブラリである。多くのスパコンで提供されている共有ファイルシステムをプログラム間のデータ連携手段として使用することで高い汎用性を確保し、ファイルを通信手段として用いる際に一般的に問題となる同期の問題をWaitIOライブラリで解決する。WaitIOライブラリは元々ファイルシステム経由で通信を前提としていたが、複数の並列プログラム間での通信を実施するために、OdysseyとAquarius間の通信ライブラリとしても開発が進められている[4]。h3-Open-SYS/WaitIOは、h3-Open-BDECの多機能カプラ (Coupler) であるh3-Open-UTIL/MP [5]と組み合わせることによって、Wisteria/BDEC-01のようなヘテロジニアスなシステム上で「S+D+L」融合を実現できる。h3-Open-SYS/WaitIOとh3-Open-UTIL/MPの使用によって、OdysseyとAquariusを連携し、「計算・データ・学習」融合の実現が始まっており、観測データ同化による長周期地震動リアルタイム予測シミュレーション等へ適用されている [9,10,11]。

4 h3-Open-SYS/WaitIO

h3-Open-SYS/WaitIO(以下 WaitIO)はOdysseyとAquariusを連携し「S+D+L」融合を実現するため

に開発された 2 つ以上の並列プログラム間で通信を行う通信ライブラリである。各並列プログラムは典型的には MPI プログラムであるが、MPI 以外の形の並列プログラムからも利用できるような汎用的に設計されている。

WaitIO は多様な並列計算機環境を想定している。特に、通信を行う並列プログラムが同一の並列計算機で動作する場合はもちろん、前述の *Odyssey* と *Aquarius* のように各々の並列プログラム異なる並列計算機で動作する場合にも対応する。そのために通信経路としてファイルと TCP/IP を想定する。

4.1 WaitIO-File

ファイル経由での通信を行う WaitIO を特に WaitIO-File（以下、簡単に WaitIO-File）と呼ぶ。WaitIO-File は同時実行される並列プログラムの間の通信手段が共有ファイルシステムに限られる状況を想定し、共有ファイルを通信路として使用するものである。一般にスーパーコンピュータは、並列プロセスに対して割り当てたノード群から外への通信を許さない運用が多いため、共有ファイルを用いたデータの受け渡しは非効率であるものの現実的な手段である。WaitIO-File は通信を行うプロセスのペアそれぞれに対して通信方向別に 1 個ずつのファイルを準備する。送信側は送信データをファイル末尾に追記し続け、受信側はそのファイルを読み込む動作が基本であるが、特に受信側はファイルの末尾以降を読み込もうとした場合には新たなデータの追記を待つ点が通常のファイル I/O と異なる（WaitIO はこの特徴を表した名前である）。このファイルの追記を効率的に待つ手法、また、送受信が完了したデータを削除する方法について、ファイルシステムに依存した細かな最適化が必要であり現在検討を進めている。

4.2 WaitIO-Socket

TCP/IP を用いて通信を行う WaitIO を特に WaitIO-Socket と呼ぶ。WaitIO-Socket は並列プログラムを構成する各プロセスが、別の並列プログラムの各プロセスに直接通信できる環境を想定している。前述のように現在のスーパーコンピュータの通常の運用ではこのような通信は許されていないことが多いが、並列計算機を構成するインターコネクトは TCP/IP での通信をサポートしていることが多いため、セキュリティ上の問題を考慮の上、運用方針を明確にすれば技術的には容易に実現可能な環境と言える。WaitIO-Socket の基本アイデアは、並列プログラムを構成する各プロセスが通信相手の並列プログラムの全プロセスに対して TCP/IP のコネクションを張り通信を行うものである。この基本アイデアはシンプルであるが、各並列プロセスが 1 万を超える数のプロセスから構成されるような大規模環境を想定する際に、通信ソケットだけでメモリ資源を使い尽くすようなことがないよう、設計、実装上の工夫が必須である。

本稿では、WaitIO-Socket についてその動作と使い方を含めて詳細に説明する。

5 WaitIO の動作仕様

WaitIO は複数の MPI アプリケーション間で通信を行うライブラリであり既存プログラムに WaitIO によるデータ通信を追加することにより複数アプリケーション間のデータ交換を実現するものである。図 4 に WaitIO の動作イメージを示す。

各アプリケーションは Parallel Block (以降、PB) と呼ばれる複数のプロセスグループで管理される。1 PB は MPI の場合一回の `mpiexec` コマンド実行に対応する。

複数の PB から構成される WaitIO 全体は WaitIO Instance と呼ばれ、各 PB は PBID と呼ばれる識別子で管理されている。PBID=0 の PB を MASTER PB と呼ばれ、プログラムの起動時に初期

化され、構成される PB 間で同期して各プロセス情報を交換するサーバーとして機能する。初期化に必要な情報はプログラム実行時のシェルの環境変数で定義され、WAITIO_MASTER_HOST, WAITIO_MASTER_PORT, WAITIO_PBBID, WAITIO_NPB (構成される PB 数)を各プログラム実行時に設定する。

WaitIO を用いて通信を行うには WaitIO Group(WG)を作成する必要がある。WG を構成するメンバープロセスは開発者が任意に選択することが可能である。WG は MPI の Communicator と同じ概念である。WG が作成されると各メンバープロセスには MPI と同様な概念のランク番号が与えられ、このランク番号を宛先に指定した通信が実行可能になる。

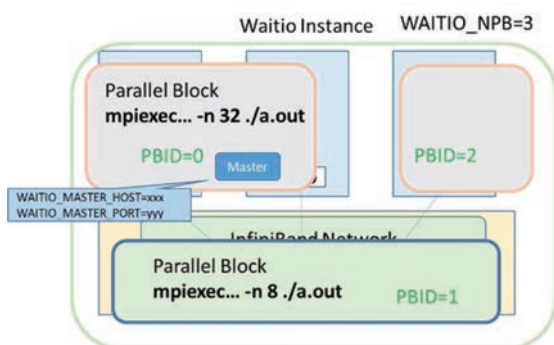


図 4 WaitIO の動作イメージ

5.1 実装 API の概要

表 1 に WaitIO の API を示す。初期化とサポート関数以外は 基本通信関数 Non-Blocking Send/Recv と Wait 関数だけである。

表 1. WaitIO の API

WaitIO API	概要
waitio_isend	Non-Blocking送信
waitio_irecv	Non-Blocking受信
waitio_wait	送受信完了待ち合わせ
waitio_init	WaitIO初期化
waitio_get_nprocs	PB毎の参加プロセス数獲得
waitio_create_group	PB間通信グループ生成 (メンバ配列指定、関数指定)
waitio_create_group_wranks	
waitio_group_rank	グループ内Rankの獲得
waitio_group_size	グループサイズの獲得
waitio_pb_size	全PBのサイズ獲得
waitio_pb_rank	全PB内Rankの獲得

各関数のうち特徴的な関数について説明する。

(1) 初期化終了 API: int waitio_init(int timeout);

- 初期化を実施する。MPI における MPI_Init()と同様に全プロセスが必ず最初に呼ぶ必要がある。本関数の実行によりマスタープロセスが立ち上がり、全員が参加するまで timeout 秒待つ。

(2) 各 PB を構成するプロセス数獲得 API : int waitio_get_nprocs(int ary[]);

- 返り値は WAITIO Instance に含まれる並列プロセス数(環境変数 WAITIO_NPB で設定した数)
 - ary に各 PB に含まれるプロセス数が返る
- (3) 初期化終了 API : `int waitio_finalize();`
- 終了処理を実施する。実行完了後、再び `waitio_init()` を呼び出すことはできない。
- (4) グループ作成 API : `waitio_group_t waitio_create_group(int gid, waitio_filter_func_t[], int order[]);`
- 本関数は構成される各 PB に対して PB 内プロセスを選別する関数(`waitio_filter_func_t`)を与え選別した結果を `order` の順に並べて WG を作成する関数である。
- (5) 通信 API : `int waitio_isend(waitio_group_t group, int dst, char *buf, size_t len, unsigned long tag, waitio_req_t *req);`
- group 内のランク `dst` に送信する。
- (6) 通信 API : `int waitio_irecv(waitio_group_t group, int src, char *buf, size_t len, unsigned long tag, waitio_req_t *req);`
- group 内のランク `src` から受信する。ただし MPI の持つ `MPI_ANY_SOURCE`, `MPI_ANY_TAG` の概念はない。
- (7) 通信 API : `int waitio_wait(waitio_req_t *req);`
- req で指定した `isend`, `irecv` 処理の完了を待つ

6 WaitIO プログラミング

図 5 に WaitIO の API を用いた PingPong プログラムを示す。このプログラムは 0 から始まる WaitIO のランク番号が偶数の場合は送信後に受信、奇数の場合は受信後に送信する。

図 5 において左半分は初期化部分である。13 行目の MPI の初期化の後 17 行目で WaitIO の初期化関数を呼んでいる。17 行目において引数の -1 は無限大の待ち時間を示す。

22 行目に WaitIO の複数 MPI プログラムを結合するプロセスグループを生成する関数 `waitio_create_group()` の引数は 8 行目と 9 行目で定義されている。8 行目は 2 つまでの PB 結合をする際のプロセスを選択する関数を指定しており、`ture()` 関数が指定されている。これは PB を構成するすべてのプロセスが生成されるグループのメンバになることを示している。9 行目は PB 間を結合した場合のランク番号の順序を指定する配列で、PB 番号 0 番の後 PB 番号 1 番の PB の順にランク番号が割り当てられる。

22 行目で生成されたプロセスグループの WaitIO Group(WG)におけるランク番号は 28 行目の `waitio_group_rank()` 関数により取得することができる。

29 行目以降は通信を行うコードであるが、28 行目で取得したランク番号を元に偶数奇数で送受信の順序を変更する。ランク番号が奇数の場合は自己のランク番号より 1 つ小さなランク番号からの受信処理後送信、ランク番号が偶数の場合は自己のランク番号より 1 つ大きなランク番号への送信処理後に受信するコードとなっている。

WaitIO API が提供する通信は Non-Blocking 通信のみであるので、送受信処理の完了待ちのために `waitio_wait()` 関数を発行している。

```

001 /* waitio-test.c */
002 #include <mpi.h>
003 #include "waitio.h"

004 int truef(int pbid, int n) { return 1; }

006 int main (int argc, char *argv[]) {
007     int ret, wrank;
008     waitio_filter_func_t func[4] = {truef, truef, NULL, NULL};
009     int array[4] = {1, 2, 0, 0};
010     int data[2], *buf = data;
011     waitio_req_t req;
012     waitio_group_t grp1;

013     if((ret = MPI_Init(&argc, &argv)) != MPI_SUCCESS) {
014         fprintf(stderr, "MPI_Init failed code %d\n", ret);
015         exit(ret);
016     }
017     if((ret = waitio_init(-1)) != 0) {
018         fprintf(stderr, "waitio_init failed code %d\n", ret);
019         MPI_Abort(MPI_COMM_WORLD, ret);
020         exit(ret);
021     }

022     grp1 = waitio_create_group(0, func, array);
023     if(grp1 == NULL) {
024         fprintf(stderr, "waitio_create_group failed code %d\n", ret);
025         MPI_Finalize();
026         exit(ret);
027     }
028     waitio_group_rank(grp1, &wrank);

029     if((wrank%2) == 1) {
030         waitio_irecv(grp1, wrank-1, (char *)buf, 4, 0, &req);
031         if((ret = waitio_wait(&req)) != 0) {
032             fprintf(stderr, "%d waitio_irecv error %d\n", wrank, ret);
033         }
034     } else {
035         fprintf(stderr, "rank%d: waitio_irecv from rank%d\n", wrank, wrank-1);
036     }
037     waitio_isend(grp1, wrank-1, (char *)buf, 4, 0, &req);
038     if((ret = waitio_wait(&req)) != 0) {
039         fprintf(stderr, "%d waitio_isend error %d\n", wrank, ret);
040     }
041     else {
042         fprintf(stderr, "rank%d: waitio_isend to rank%d\n", wrank, wrank-1);
043     }
044 }
045 else {
046     waitio_isend(grp1, wrank+1, (char *)buf, 4, 0, &req);
047     if((ret = waitio_wait(&req)) != 0) {
048         fprintf(stderr, "%d waitio_isend error %d\n", wrank, ret);
049     }
050     else {
051         fprintf(stderr, "rank%d: waitio_isend to rank%d\n", wrank, wrank+1);
052     }
053     waitio_irecv(grp1, wrank+1, (char *)buf, 4, 0, &req);
054     if((ret = waitio_wait(&req)) != 0) {
055         fprintf(stderr, "%d waitio_irecv error %d\n", wrank, ret);
056     }
057     else {
058         fprintf(stderr, "rank%d: waitio_irecv from rank%d\n", wrank, wrank+1);
059     }
060 }
061 waitio_finalize();
062 MPI_Finalize();
063 }

```

図 5 WaitIO-Socket の API を用いた PingPong プログラム例

図 6 に図 5 のプログラムの実行例を示す。

この例では Intel MPI を用いた wisteria01 でのインタラクティブな実行例と Odyssey と Fujitsu MPI を用いたバッチシステムでの実行例を示す。

図 6 左は Intel MPI を用いた実行例である。WaitIO で利用するライブラリは利用する MPI 毎に提供されており、Intel MPI の場合は `_intel`、Fujitsu MPI の場合は `_a64fx` の post fix をライブラリに追加する。Aquarius の Open MPI の場合は Post Fix は不要である。また、実行時に図にある環境変数 `WAITIO_MASTER_HOST`, `WAITIO_MASTER_PORT`, `WAITIO_PPID`, `WAITIO_NPB` の設定が必要である。この他、WaitIO 関連のライブラリが環境変数 `LD_LIBRARY_PATH` の指定にある `$(WAITIOLIB)`にある想定としている。ここで、環境変数 `WAITIO_MASTER_PORT` については、同一ホストを MASTER とするプログラムが他に存在し PORT 番号が同じ場合誤動作するため `WAITIO_MASTER_PORT` の数値は他のプログラムと同じにならないよう注意する必要がある。図 6 左の例では続いて、これらの環境変数を実行環境に設定し `mpirun` コマンドで 2 プロセスの場合、8 プロセスの場合を実行している。この実行においては、`WAITIO_NPB` の値が 1 であるために他の PB を待つことなく実行が開始される。

図 6 右上は Fujitsu MPI でコンパイルして Odyssey システムにてバッチ実行した実行例である。この例では図 6 左と同様にバッチスクリプトに必要な環境変数を設定し、`pjsub` でバッチ処理を開始する。実行結果は図 6 右下のようにファイルに出力される。

```
[z30xxx@wisteria01 waitio]$ mpicc test.c -o test -L$(WAITIOLIB)/ -lwaitio_intel

[z30xxx@wisteria01 waitio]$ cat env.sh
export LD_LIBRARY_PATH=$(WAITIOLIB)/:$LD_LIBRARY_PATH
export WAITIO_MASTER_HOST=wisteria01
export WAITIO_MASTER_PORT=7100
export WAITIO_PPID=0
export WAITIO_NPB=1
[z30xxx@wisteria01 waitio]$ . env.sh

[z30xxx@wisteria01 waitio]$ mpiexec -np 2 ./test
wisteria01:0:0:sock_create master_port 7100
wisteria01:0:0: WAITIO_MASTER socket bind port 7100
rank0: waitio_isend to rank1
rank1: waitio_irecv from rank0
rank0: waitio_isend to rank1
rank1: waitio_irecv from rank0

[z30xxx@wisteria01 waitio]$ mpiexec -np 8 ./test
wisteria01:0:0:sock_create master_port 7100
wisteria01:0:0: WAITIO_MASTER socket bind port 7100
rank0: waitio_isend to rank1
rank4: waitio_isend to rank5
rank1: waitio_irecv from rank0
rank2: waitio_isend to rank3
rank5: waitio_irecv from rank4
rank5: waitio_isend to rank4
rank0: waitio_irecv from rank1
rank1: waitio_isend to rank0
rank3: waitio_irecv from rank2
rank4: waitio_irecv from rank5
rank6: waitio_isend to rank7
rank6: waitio_irecv from rank7
rank7: waitio_irecv from rank6
rank7: waitio_isend to rank6
rank2: waitio_irecv from rank3
rank3: waitio_isend to rank2
[z30zzz@wisteria01 waitio]$
```

Intel MPIを用いたインタラクティブ実行例

```
[z30xxx@wisteria01 waitio]$ mpifccpx test.c -o test-a64fx -L$(WAITIOLIB)/ -lwaitio_a64fx
[z30xxx@wisteria01 waitio]$ cat test-a64fx.sh
#!/bin/sh
#----- pjs sub option -----#
#PJM -L rscgrp-debug-o
#PJM -L node=1
#PJM --mpi proc=2
#PJM -L elapse=00:02:00
#PJM -g jhxxxxxxo
#PJM -j
#----- Program execution -----#

export LD_LIBRARY_PATH=$(WAITIOLIB)/:$LD_LIBRARY_PATH
module load fj
module load fjmpi
export WAITIO_MASTER_HOST="hostname"
export WAITIO_MASTER_PORT=7100
export WAITIO_PPID=0
export WAITIO_NPB=1

mpiexec ./test-a64fx
exit
[z30xxx@wisteria01 waitio]$ pjs sub test-a64fx.sh
[INFO] PJM 0000 pjs sub Job 244072 submitted.
[z30xxx@wisteria01 waitio]$
```

```
wo0001:0:0:sock_create master_port 7100
wo0001:0:0: WAITIO_MASTER socket bind port 7100
rank0: waitio_isend to rank1
rank1: waitio_irecv from rank0
rank1: waitio_isend to rank0
rank0: waitio_irecv from rank1
```

Fujitsu MPIを用いたバッチシステム実行例

図 6 WaitIO-Socket の API を用いた PingPong プログラムの実行例(単一プログラム実行)

```
[z30xxx@wisteria01 waitio]$ cat test-a64fx-1.sh
#!/bin/sh
#PJM -L rscgrp-debug-o
#PJM -L node=1
#PJM --mpi proc=2
#PJM -L elapse=00:02:00
#PJM -g jhxxxxxxo
#PJM -j
hostname
export LD_LIBRARY_PATH=$(WAITIOLIB)/:$LD_LIBRARY_PATH
module load fj
module load fjmpi
export WAITIO_MASTER_HOST="hostname"
export WAITIO_MASTER_PORT=7100
export WAITIO_PPID=0
export WAITIO_NPB=2
mpiexec ./test-a64fx
exit

[z30xxx@wisteria01 waitio]$ cat test-intel-2.sh
#!/bin/sh
#PJM -L rscgrp-debug-a
#PJM -L node=1
#PJM --mpi proc=2
#PJM -L elapse=00:05:00
#PJM -g jhxxxxxxa
#PJM -j
module unload aquarius
module unload gcc
module load intel
module load impi
export LD_LIBRARY_PATH=$(WAITIOLIB)/:$LD_LIBRARY_PATH
export WAITIO_MASTER_HOST=wo0001
export WAITIO_MASTER_PORT=7100
export WAITIO_PPID=1
export WAITIO_NPB=2
mpiexec -n 2 ./test
exit

[z30xxx@wisteria01 waitio]$ pjs sub test-a64fx-1.sh
[INFO] PJM 0000 pjs sub Job 244102 submitted.
[z30xxx@wisteria01 waitio]$ pjs sub test-intel-2.sh
[INFO] PJM 0000 pjs sub Job 244104 submitted.
[z30zzz@wisteria01 waitio]$
```

```
wo0001
wo0001:0:0:sock_create master_port 7100
wo0001:0:0: WAITIO_MASTER socket bind port 7100
wo0001:0:0:waitio_init Procs:
PB No.0 rank=0 IP-addr:port#=10.11.101.1:8000 nprocs=2, nGIO=0
PB No.1 rank=0 IP-addr:port#=10.0.6.1:8000 nprocs=2, nGIO=0
wo0001:0/2(85): Multiple WaitIO Connector now ready NPB=2!
wo0001:0/2(85): PBserver: waitio_fetch_PBdata accepted from 10.0.6.1 port=38607 fd=33
rank0: waitio_isend to rank1
rank1: waitio_irecv from rank0
rank1: waitio_isend to rank0
rank0: waitio_irecv from rank1
```

Odysseyでの実行結果例

```
wa01:1:0/2(87): PBclient: trying connect host 10.11.101.1, port 48155
wa01:1/2(87): PBclient: connected to host 10.11.101.1, port 48155
wa01:1/2(87): PBclient: upstream to WaitIO master Nproc=2 done!
wa01:1:0:waitio_init Procs:
PB No.0 rank=0 IP-addr:port#=10.11.101.1:8000 nprocs=2, nGIO=0
PB No.1 rank=0 IP-addr:port#=10.0.6.1:8000 nprocs=2, nGIO=0
wa01:1/2(87): Multiple WaitIO Connector now ready NPB=2!
rank2: waitio_isend to rank3
rank2: waitio_irecv from rank3
rank3: waitio_irecv from rank2
rank3: waitio_isend to rank2
```

Aquariusでの実行結果例

Wisteria/BDEC01(Odyssey+Aquarius)を用いた試行例

図 7 WaitIO-Socket の API を用いた PingPong プログラムの実行例(バッチによる複数プログラムの結合実行)

図 7 は Odyssey と Aquarius を用いて図 6 の実行バイナリを協調動作した例である。図 6 で生成した実行バイナリをそのまま利用してバッチスクリプトにて環境変数を設定するだけで協調動作することが可能である。バッチ処理の実行は WAITIO_PBID が 0 であるスクリプトから開始する。

図 7 左では Odyssey でのバッチ処理である test-a64fx-1.sh が WAITIO_PBID=0 であるためこのファイルから実行する。バッチ処理では実行時にノードが割り当てられるために、どのノードで実行されたかを知るためにスクリプト内の hostname コマンドを用いて環境変数 WAITIO_MASTER_HOST を設定するほか、Aquarius で実行する WAITIO_PBID=1 のバッチスクリプトに設定するために hostname コマンドでバッチ処理出力に出力している。

Aquarius 向けのバッチスクリプトである test-intel-2.sh ではバッチ処理 test-a64fx-1.sh の出力の hostname コマンド出力結果を test-intel-2.sh の WAITIO_MASTER_HOST と WAITIO_MASTER_PORT 環境変数に設定した後 pjsb コマンドにてバッチ処理を実行する。

図 7 右に Odyssey(上), Aquarius(下)のバッチジョブ出力結果を示す。それぞれのバッチ処理出力では最初に 2 つのバッチジョブが結合されて 4 プロセスの WaitIO Instance が生成され Odyssey の 2 プロセスに WaitIO Gourp(WG)のランク番号 0,1 と Aquarius の 2 プロセスにランク番号 2,3 が割り当てられ 4 プロセスによるプログラム実行が行われたことを示している。この例ではそれぞれ 2 プロセスとしたがそれぞれのプロセス数を増減しても、実行時に反映される他、3 つ以上のバッチジョブについてもスクリプトを準備することで実行可能である。

7 WaitIO-MPI Conversion ライブラリの概要と動作仕様

WaitIO はシンプルな通信に特化するため MPI の Datatype に相当する概念がない。このため、MPI で書かれたプログラムを WaitIO に書き直すのはひと手間かかる。プログラムを書き直すとはプログラム実行の不具合につながる。したがって、可能な限り機械的に書き直せることが望ましい。

このため、MPI とのインタフェース変換を担う WaitIO-MPI Conversion ライブラリを準備した。この WaitIO-MPI Conversion ライブラリは、必要になった関数を随時加えることとしている。API としては FORTRAN 向け API も実装している。

現状、実装されている関数を表 2 に示す。同期・非同期送受信関数(send, recv, isend, irecv)の他、集団通信関数(bcast, reduce, allreduce, gather, allgather, scatter, scatter, gather, battier)であり、扱う Datatype は数種の基本 Datatype(CHAR, INT, FLOAT, DOUBBLE)に限定している。アプリケーションの必要に応じて拡張する予定である。現状、関数名と datatype, op などの MPI オブジェクト表記は WAITIO_ の prefix を機械的に追加することで WaitIO 向けに書き換えるようにしている。

実装されている集団通信アルゴリズムは動作確認と通信負荷を目的としたシンプルな 2 階層のシーケンシャル実装(2 Layered:以降 2L)と Binary(Binominal) Tree ベース(以降 BT)である。シーケンシャル実装の Bcast は root プロセスから各 worker プロセスに送信、reduce は各 worker プロセスから root プロセスに送信、root プロセスで演算する原始的なものである。また、Allreduce は reduce+bcast 実装である。ただし 1 プロセスに同時通信できるソケット数が限られるため、一定プロセス数以上の場合は階層的に通信する実装としている。Binary(Binominal) Tree ベースは Binary Tree 上にプロセスをマップしメッセージ交換を実施する。

表.2 WaitIO-MPI Conversion ライブラリの API 概要

WaitIO-MPI API	概要
waitio_mpi_send	WaitIO実装版MPI_Send
waitio_mpi_recv	WaitIO実装版MPI_Recv
waitio_mpi_isend	WaitIO実装版MPI_Isend
waitio_mpi_irecv	WaitIO実装版MPI_Irecv
waitio_mpi_reduce	WaitIO実装版MPI_Reduce
waitio_mpi_bcast	WaitIO実装版MPI_Bcast
waitio_mpi_allreduce	WaitIO実装版MPI_Allreduce
waitio_mpi_waitall	WaitIO実装版MPI_Waitall
waitio_create_universe	Waitio初期化サポート関数:全ランク
waitio_create_universe_pbhead	Waitio初期化サポート関数:PB先頭
waitio_mpi_gather	WaitIO実装版MPI_Gather
waitio_mpi_allgather	WaitIO実装版MPI_Allgather
waitio_mpi_scatter	WaitIO実装版MPI_Scatter
waitio_mpi_scatter	WaitIO実装版MPI_Scatterv
waitio_mpi_gather	WaitIO実装版MPI_Gatherv
waitio_mpi_barrier	WaitIO実装版MPI_Barrier
waitio_mpi_type_size	WaitIO実装版MPI_Typ_size

8 WaitIO-MPI Conversion ライブラリプログラミング

```

000 /* test-mpi.c */
001 #include <mpi.h>
002 #include "waitio.h"
003 #include "waitio_mpi.h"

004 int main (int argc, char *argv[]) {
005     int data[2], *buf = data;
006     waitio_group_t grp1;
007     int ret;
008     int wrank;

009     if((ret = MPI_Init(&argc, &argv)) != MPI_SUCCESS) {
010         fprintf(stderr, "MPI_Init failed code %d\n", ret);
011         exit(ret);
012     }
013
014     waitio_create_universe (&grp1);
015     if(grp1 == NULL) {
016         fprintf(stderr, "waitio_create_universe failed code %d\n", ret);
017         MPI_Finalize();
018         exit(ret);
019     }

020     waitio_group_rank(grp1, &wrank);
021     if((wrank%2) == 1) {
022         if((ret = waitio_mpi_recv(buf, 4, WAITIO_MPI_CHAR, wrank-1, 0, grp1)) != 0) {
023             fprintf(stderr, "%d waitio_mpi_recv error %d\n", wrank, ret);
024         }
025     } else {
026         fprintf(stderr, "rank%d: waitio_mpi_recv from rank%d\n", wrank, wrank-1);
027     }
028     if((ret = waitio_mpi_send(buf, 4, WAITIO_MPI_CHAR, wrank+1, 0, grp1)) != 0) {
029         fprintf(stderr, "%d waitio_send error %d\n", wrank, ret);
030     }
031     else {
032         fprintf(stderr, "rank%d: waitio_mpi_send to rank%d\n", wrank, wrank+1);
033     }
034 }
035 else {
036     if((ret = waitio_mpi_send(buf, 4, WAITIO_MPI_CHAR, wrank+1, 0, grp1)) != 0) {
037         fprintf(stderr, "%d waitio_send error %d\n", wrank, ret);
038     }
039     else {
040         fprintf(stderr, "rank%d: waitio_mpi_send to rank%d\n", wrank, wrank+1);
041     }
042     if((ret = waitio_mpi_recv(buf, 4, WAITIO_MPI_CHAR, wrank-1, 0, grp1)) != 0) {
043         fprintf(stderr, "%d waitio_recv error %d\n", wrank, ret);
044     }
045     else {
046         fprintf(stderr, "rank%d: waitio_mpi_recv from rank%d\n", wrank, wrank-1);
047     }
048 }

049 waitio_finalize();
050 MPI_Finalize();
051 }

```

waitio_mpi_recv は WAITIO_MPI_Recv
waitio_mpi_send は WAITIO_MPI_Send
としても定義されているため、MPIプログラムの機械的な
置き換えも可能である。

図 8. WaitIO-MPI Conversion ライブラリの API を用いた PingPong プログラム例

図 8 に WaitIO-MPI Conversion ライブラリの API を用いた PingPong プログラムを示す。このプログラムは図 5 のプログラムをそのまま WaitIO-MPI Conversion ライブラリに置き換えたもので、0 から始まる WaitIO のランク番号が偶数の場合は送信後に受信、奇数の場合は受信後に送信する。

図 8 において左半分は初期化部分である。9 行目の MPI の初期化の後 14 行目で WaitIO-MPI

Conversion ライブラリの初期化関数を呼んでいる。ここでは、WaitIO Instance すべてのプロセスを利用する waitio_create_universe()関数を用いている。現在のところ本関数と各 PB の先頭プロセスをグループ化する waitio_create_universe_pbhead()関数が用意されている。

14 行目で生成されたプロセスグループの WaitIO Group(WG)におけるランク番号は 20 行目の waitio_group_rank()関数により取得することができる。

21 行目以降は通信を行うコードであるが、20 行目で取得したランク番号を元に偶数奇数で送受信の順序を変更する。ランク番号が奇数の場合は自己のランク番号より 1 つ小さなランク番号からの受信処理後送信、ランク番号が偶数の場合は自己のランク番号より 1 つ大きなランク番号への送信処理後に受信するコードとなっている。WaitIO ライブラリと異なるのはデータタイプの指定が必要である点である。

WaitIO-MPI Conversion ライブラリ API は Non-Blocking 通信の他 Blocking 通信も準備されているので、本プログラムでは Blocking 通信関数を用いている。

```
[z30xxx@wisteria01 waitio]$ mpicc test-mpi.c -o test-mpi -L$(WAITIOLIB)/-
lwaitio_mpi_intel -lwaitio_intel
[z30xxx@wisteria01 waitio]$ mpifccpx test-mpi.c -o test-mpi-a64fx -
L$(WAITIOLIB)/ -lwaitio_mpi_a64fx -lwaitio_a64fx
[z30xxx@wisteria01 waitio]$ cat test-mpi-a64fx-1.sh
#!/bin/sh
#PJM -l rscgrp=debug-o
#PJM -l node=1
#PJM --mpi proc=2
#PJM -l elapse=00:02:00
#PJM -g jh2xxxxxx
#PJM -j
hostname
export LD_LIBRARY_PATH=$(WAITIOLIB)/:$LD_LIBRARY_PATH
module load fj
module load fjmpl
export WAITIO_MASTER_HOST=hostname
export WAITIO_MASTER_PORT=7100
export WAITIO_PPID=0
export WAITIO_NPB=2
mpirun -x test-mpi-a64fx
exit

[z30xxx@wisteria01 waitio]$ cat test-mpi-intel-2.sh
#!/bin/sh
#PJM -l rscgrp=debug-a
#PJM -l node=1
#PJM --mpi proc=2
#PJM -l elapse=00:05:00
#PJM -g jh2xxxxxx
#PJM -j
module unload aquarius
module unload gcc
module load intel
module load impi
export LD_LIBRARY_PATH=$(WAITIOLIB)/:$LD_LIBRARY_PATH
export WAITIO_MASTER_HOST=wo0009
export WAITIO_MASTER_PORT=7100
export WAITIO_PPID=1
export WAITIO_NPB=2
mpirun -n 2 ./test-mpi
exit

[z30xxx@wisteria01 waitio]$ pjsub test-mpi-a64fx-1.sh
[INFO] PJM 0000 pjsub Job 244535 submitted.
###< Check the output of Job 244535 and set the hostname of the job output
### to test-mpi-intel-2.sh>
[z30xxx@wisteria01 waitio]$ pjsub test-mpi-intel-2.sh
[INFO] PJM 0000 pjsub Job 244537 submitted.
```

```
wo0009
wo0009:0:0:sock_create master_port 7100
wo0009:0:0: WAITIO_MASTER socket bind port 7100
wo0009:0:0 waitio_init Procs:
PB No.0 rank=0 IP-addr:port#=10.11.101.9:8000 nprocs=2, nGIO=0
PB No.1 rank=0 IP-addr:port#=10.0.6.1:8000 nprocs=2, nGIO=0
wo0009:0/2(83): Multiple WaitIO Connector now ready NPB=2!
wo0009:0/2(83): PBserver: waitio_fetch_PBdata accepted from 10.0.6.1 port=30403 fd=33
rank0: waitio_mpi_send to rank1
rank1: waitio_mpi_rcv from rank0
rank1: waitio_mpi_send to rank0
rank0: waitio_mpi_rcv from rank1
```

Odysseyでの実行結果例

```
wa01:1:0/2(87):PBclient:trying connect host 10.11.101.9,port 48155
wa01:1:2(87):PBclient:connected to host 10.11.101.9,port 48155
wa01:1:2(87):PBclient:upstream to WaitIO master Nproc=2 done!
wa01:1:0 waitio_init Procs:
PB No.0 rank=0 IP-addr:port#=10.11.101.9:8000 nprocs=2, nGIO=0
PB No.1 rank=0 IP-addr:port#=10.0.6.1:8000 nprocs=2, nGIO=0
wa01:1:2(87): Multiple WaitIO Connector now ready NPB=2!
rank2: waitio_mpi_send to rank3
rank2: waitio_mpi_rcv from rank3
rank3: waitio_mpi_rcv from rank2
rank3: waitio_mpi_send to rank2
```

Aquariusでの実行結果例

Wisteria/BDEC01(Odyssey+Aquarius)を用いた試行例

図 9. WaitIO-MPI Conversion ライブラリの API を用いた PingPong プログラムの実行例

(パッチによる複数プログラムの結合実行)

図 9 に Odyssey と Aquarius を用いて図 8 のプログラムをバイナリ生成と協調動作した例である。図 7 に示した WaitIO ライブラリのプログラムとコンパイル、実行方法はほとんど同じであるが、コンパイル時に-lwaitio_mpi ライブラリの追加が異なる。

9 WaitIO-MPI Conversion の適用例

本章では WaitIO-MPI Conversion の適応例として MPI+FORTRAN で書かれた pHEAT-3D について述べる。pHEAT-3D は GeoFEM[12]プロジェクトで開発された並列有限要素法アプリケーションを元に開発した三次元定常熱伝導解析コードである。pHEAT-3D の主要な通信部分は MPI_Isend()+MPI_Irecv()で書かれた通信部分と MPI_Allreduce()で書かれた部分である。

図 10 に主要部分のソース変更を示す。赤字で書かれたところが修正部分である。基本的に MPI プログラムの MPI_ のプレフィックスを WAITIO_MPI_ のプレフィックスに置換することでソースコード修正できるように設計している。

一つ MPI と異なる点としては、WaitIO では MPI_Status に相当するオブジェクトが存在しないのと MPI_Request に相当するオブジェクトが MPI はスカラー定義であるが WaitIO は構造体である点に注意が必要である。このため、プログラムにおいては Request オブジェクトを構造体として確保する必要がある。例えば FORTRAN では WAITIO_REQUEST_SIZE の大きさをもつ配列として定義する必要がある。

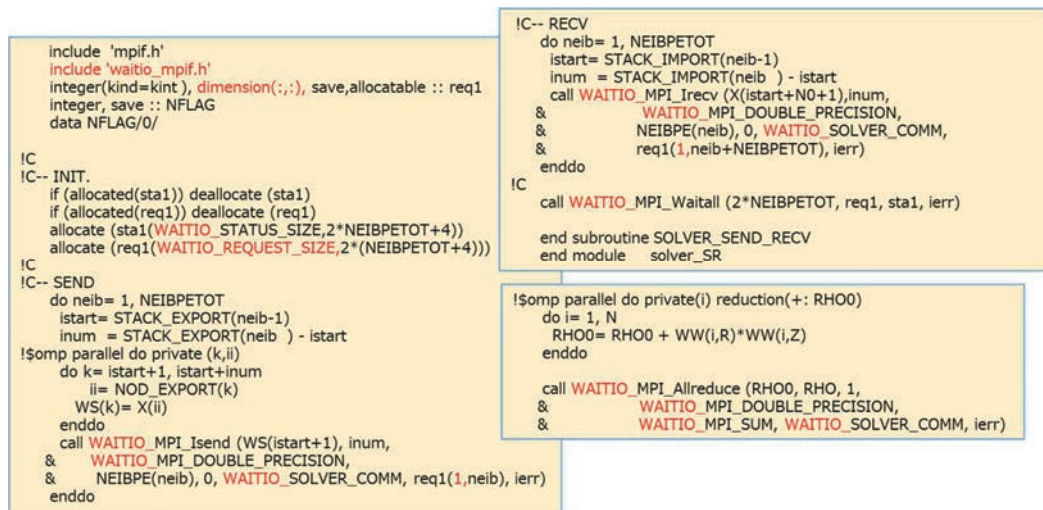


図 10. WaitIO-MPI Conversion ライブラリにおける従来プログラム pHEAT-3D の修正例

10. WaitIO-Socket 通信性能評価

本章は WaitIO-Socket 通信ライブラリが Wisteria システム上でどの程度通信性能を引き出せるかを検証することを目的として基本通信性能評価を実施した結果を述べる。

10.1 評価環境と評価プログラム

WaitIO-Socket の評価においては、より広いシステムでも稼働することを確認するため Wisteria システムの他に Oakbridge-CX, Oakforest-PACS を用いて実施した。評価環境の仕様を表 3 に示す。

表 3 Wisteria/BDEC-01, Oakbridge-CX, Oakforest-PACS の各構成システム仕様

	Wisteria/Odyssey	Wisteria/Aquarius
総理論演算性能	25.9 PFLOPS	7.2 PFLOPS
総ノード数	7,680	45
インターコネクト	Tofu-D (6次元メッシュ/トラス)	InfiniBand HDR (200Gbps) x 4HCA (Full Fat Tree)
プロセッサ/ノード	A64FX(Armv8.1+SVE), 2.2GHz, 1 socket (48 Core+2 or 4アシスタントコア)	Intel Xeon Platinum 8360Y, 2.4GHz 2 sockets(36+36)
メモリ量/ノード	32 GB	512GiB
メモリバンド幅/ノード	1024GB/s	409.6GB/s
GPU/ノード	-	NVIDIA A100 x8
コンパイラ, MPI	富士通コンパイラ, 富士通MPI	Intelコンパイラ, インテルMPI, (Open MPI)
オペレーティングシステム	Red Hat Enterprise Linux 8	Red Hat Enterprise Linux 8

	Oakbridge-CX (OBCX)	Oakforest-PACS (OFP)
総理論演算性能	6.61 PFLOPS	25 PFLOPS
総ノード数	1368	8208
インターコネクト	Omni-Path (100Gbps)	Omni-Path (100Gbps)
プロセッサ/ノード	Intel Xeon Platinum 8280 2.7GHz, 2 socket (28+28)	Intel Xeon Phi 7250 1.4GHz 1 socket (68)
メモリ量/ノード	192 GB	96(DDR4)+16(MCDRAM) GB
メモリバンド幅/ノード	281.6GB/s	115(DDR4)+490(MCDRAM) GB/s
コンパイラ, MPI	Intelコンパイラ, インテルMPI, (Open MPI)	Intelコンパイラ, インテルMPI, (Open MPI)
オペレーティングシステム	Red Hat Enterprise Linux 7, CentOS 7	Red Hat Enterprise Linux 7, CentOS 7

評価プログラムはシステム間のデータ通信性能を評価するため複数ストリームでの PingPong 通信の評価プログラムを作成した。これは、全 PB 内プロセスで2 プロセスペアを作り PingPong 通信を行うものである。すべてのペアの PingPong 通信終了を MPI_Barrier()で待ち合わせて測定している。

実行評価プログラムはそれぞれのシステムにインストールされているベンダ製コンパイラ+ベンダ製 MPI で評価してグラフ化した。Xeon プロセッサ搭載システムでは Open MPI+GCC の組み合わせでも動作確認している。

10.2 1PB 内 1 対 1 通信評価(multi-stream)

本節では 1 PB 内でのマルチストリーム 1 対 1 通信評価について述べる。

10.2.1 PingPong 1/2 RTT 評価

本節では各システムで測定した PingPong1/2 RTT の性能評価について述べる。表 4. に各システムの 8 Byte の 1/2 RTT(ノード内, ノード間)結果のサマリを示す。各データはノード数を固定、ノードあたりのプロセス数を増加させ、複数プロセスによる複数ストリームを 1000 回実行し MPI_Barrier()で同期後の時間を回数分で割り算した平均値を示している。すなわち、各回のストリーム実行の最悪値の平均である。

表 4 Wisteria/BDEC-01, Oakbridge-CX, Oakforest-PACS の 1/2RTT 測定結果

usec(8B)	1/2 RTT(ノード内)	1/2 RTT(ノード間)
Odyssey	26.8	37.1 (Tofu-D)
Aquarius	6.57	21.1 (IB)
OBCX	6.85	14.5 (OPA)
OFP	49.7	83.7 (OPA)

表 4. より、Intel Xeon プロセッサシステムのノード内の測定結果は、ほぼ同等の実行時間であ

る。一方, Odyssey, OFP については Xeon 搭載システムに比べ 4 倍弱(Odyssey), 9 倍弱(OFP)遅い結果となっている。相対的なプロセッサ処理の差であるほか, Odyssey の結果に関して, カーネル内処理はアシスタントコアで実行されるためにその遅延も含まれていると想定される。

10.2.2 PingPong バンド幅評価

本節では各システムで測定した PingPong バンド幅の性能評価について述べる。表 5.に各システムの PingPong バンド幅(ノード内, ノード間)の最大値と 1 ノードあたりの PingPong バンド幅のサマリを示す。各データは複数プロセスによる複数ストリームを 1000 回実行し MPI_barrier()で同期後の時間を回数分で割り算した平均値を示している。

表 5 Wisteria/BDEC-01, Oakbridge-CX, Oakforest-PACS の PingPong バンド幅性能の測定結果

PingPong BW GB/s	BW ノード内	BW ノード間
Odyssey	21.3	0.35 (Tofu-D)
Aquarius	259.7	27.1 (IB)
OBCX	45.7	9.3 (OPA)
OFP	8.2	1.12 (OPA)

表 5.より, 各システムのメモリ帯域, インターコネクト帯域, そしてコピー処理, 通信プロトコルを実行するプロセッサの処理性能の影響が見られる結果となっている。

表 5 の結果の中で最も高い性能を記録したのは, Aquarius である。ノード内性能ではノードあたり 260GB/秒, ノード間性能でもノードあたり 27GB/秒を観測した。PingPong 処理においては WaitIO-Socket ライブラリ-カーネル間コピーが 2 回発生する。このため Aquarius の 960 プロセス結果のメッセージ長が 8MB までの区間で大きく山が出ているのはキャッシュの影響であると想定される。

次にノード間のインターコネクト毎の性能差に着目すると, やはり最新の InfiniBand HDR 4 本を搭載する Aquarius がプロセッサ性能とのバランスに優れ ノードあたり 27.1GB/秒の通信性能を記録している。物理性能が半分の OmniPath を搭載している OBCX は 9.3GB/秒を記録している。その性能差は 2.9 倍であった。

一方, 同じ OmniPath を搭載する OFP の結果については 1.1GB/秒と OBCX に比べ 8.5 倍の性能差である。OFP に関してはノード内, ノード間とも他のシステムに比べ大きな性能差がないために, プロセッサコアの処理性能で律速していると想定される。

ここでノード間の通信性能が 0.35GB/秒の Odyssey について調べてみる。図 11 に Odyssey のノード内とノード間の測定結果を示す。図 11 左のノード内の測定結果に比べ図 11 右のノード間の結果を比較すると, ノード内の通信性能に比べ, ノード間の性能が 61 倍低い結果となった。これはノード内通信が各計算コアによるシステムコール経由のメモリコピーで済むのに対して, ノード間通信はバックグラウンドで実行される TCP/IP 処理が 1 ノードあたり 2 or 4 コアのアシスタントコアで実行されるために, このアシスタントコア処理がボトルネックとなり性能が律速されていると考えられる。

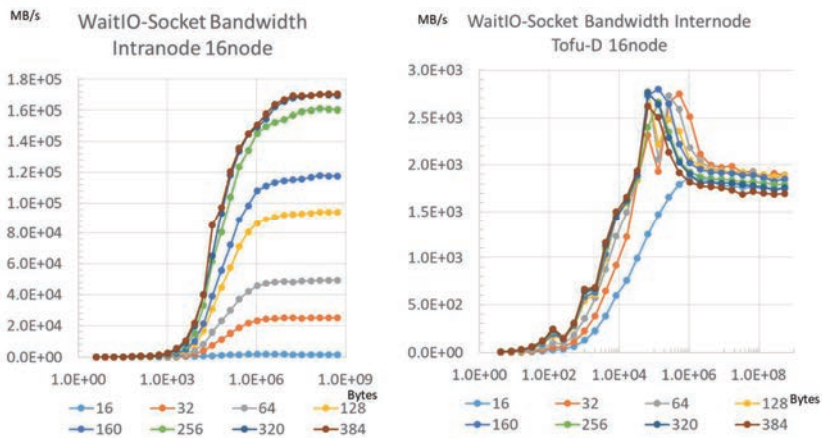


図 11. Wisteria/BDEC-01(Odyssey)の PingPong バンド幅性能の測定結果

同様に Aquarius についても図 12 にノード内とノード間の測定結果を示す。Odyssey の測定結果と異なり、ノード間の通信性能についてもノード内プロセスの増加と共に通信性能が向上していることがわかる。

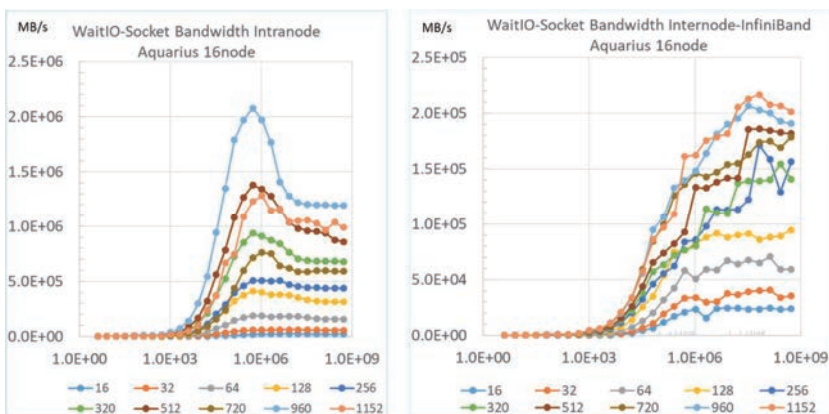


図 12 Wisteria/BDEC-01(Aquarius)の PingPong バンド幅性能の測定結果

10.3 2PB 結合での通信性能評価

本節では複数ジョブを WaitIO-Socket で結合した場合の通信性能について評価する。前節の大規模環境評価を 1 ジョブ(1 PB:Parallel Block)と 2 ジョブ(2 PB)で実行し性能差を評価する。

図 13.左に Odyssey 1 PB(2048node) 2 PB (1024+1024node), 図 13.右に Aquarius 1 PB(36node)と 2 PB(18+18node)のノード間の通信性能の比較結果を示す。

図 13.左より, Odyssey のノード間通信性能については少し性能にバタつきが見られるが傾向は類似という結果となった。

Aquarius については図 13.右より 1PB より 2PB の方が高い性能となっているが概ね同様の傾向の性能結果となった。実装上 1PB でも 2PB でも相手の IP アドレスと Port 番号をベースに相手と通信する仕組みは変わらないので, 1 ジョブ 1PB と 2 ジョブ 2PB の性能差が出る可能性としては, ジョブの割り当て場所, ランク割り当てノードとインターコネクト上の位置の違いが考えられる。

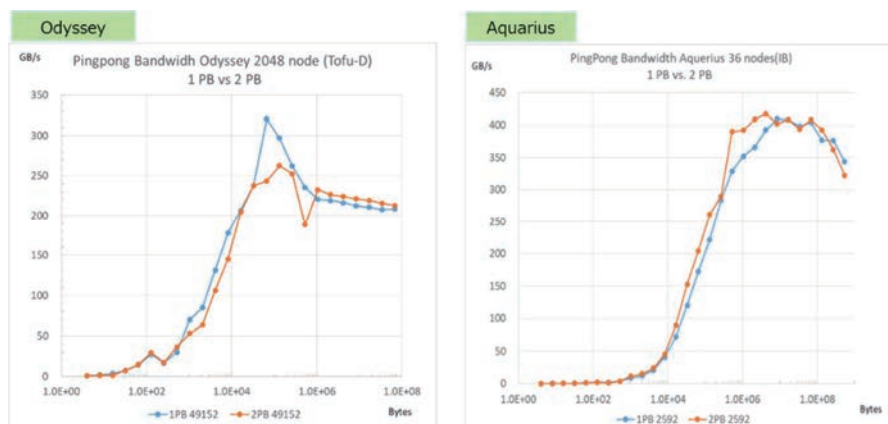


図 13. Wisteria/BDEC-01(Aquarius)の PingPong バンド幅性能の測定結果

10.4 アプリケーションを用いた WaitIO-MPI Conversion ライブラリの評価

WaitIO-MPI Conversion ライブラリを用いて pHEAT-3D アプリケーション[12]を移植し実行時間を評価した。実行環境は OBCX であり比較対象は Intel MPI(OmniPath)である。評価結果はノードあたり 8 プロセス, 1 プロセスあたり 6 スレッド, 1000STEP の実行時間の結果である。移植作業はエディタ上で機械的に実施可能で実際には WAITIO_MPI_Isend(), WAITIO_MPI_Irecv(), WAITIO_MPI_Allreduce() を用いて置き換えている。集団通信の利用アルゴリズムは BT である。

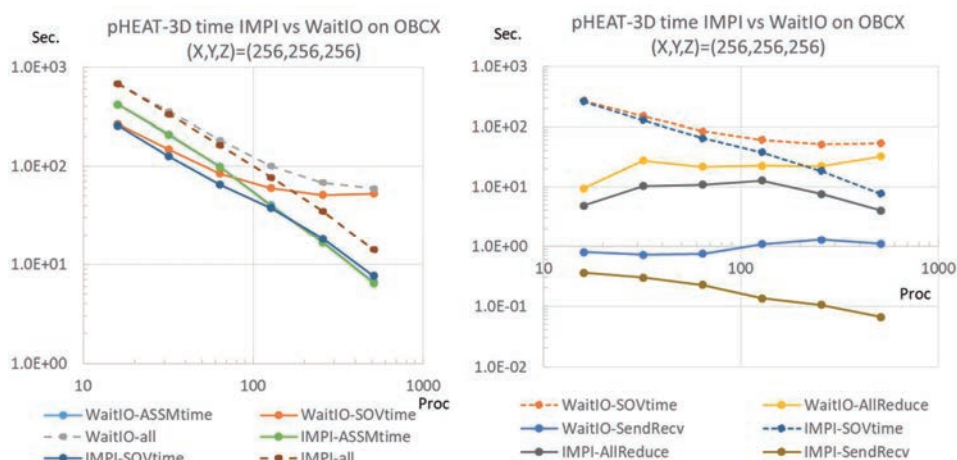


図 14. WaitIO-Socket の pHEAT-3D の性能比較(vs. Intel MPI)

図 14.左に配列サイズ $(X,Y,Z)=(256,256,256)$ の測定結果を示す。図 14.左の結果より, Intel MPI(IMPI) は 512 プロセスまで性能向上しているが, WaitIO-Socket の測定結果では 256 プロセス以上では性能が頭打ちであった。

図 14.右に pHEAT-3D で利用している 4 回の Allreduce, iSend/iRecv で実装された SendRecv 通信時間と Solver 全体の実行時間を示す。図 14. 右の結果より AllReduce 通信が支配的であることが観測された。

なお、WaitIO-ASSTime と Intel MPI-ASSTime の性能差がほとんどないのは計算のみで通信を実施していないからである。

10.5 Wisteria/BDEC-01 全系を用いた WaitIO-Socket の通信性能評価

本節では Wisteria/BDEC-01 全系を用いた WaitIO-Socket の通信性能評価を行う。Wisteria を構成する Odyssey と Aquarius システム間は InfiniBand EDR で結合されている。この InfiniBand EDR は Odyssey の 1 ラックあたり 8 つ存在する GIO に結合されている。Wisteria の各ノードへのネットワークアクセスについて、Aquarius システムは各計算ノードに InfiniBand HDR が搭載されているが、Odyssey システムの各計算ノードは GIO と呼ばれる Gateway 用の計算ノードを除き TofuD インターコネクトのみを搭載している。GIO 以外の計算ノードは GIO を経由した TCP/IP 通信によりアクセス可能である。図 15 に Wisteria/BDEC-01 のインターコネクトの接続イメージ図を示す。

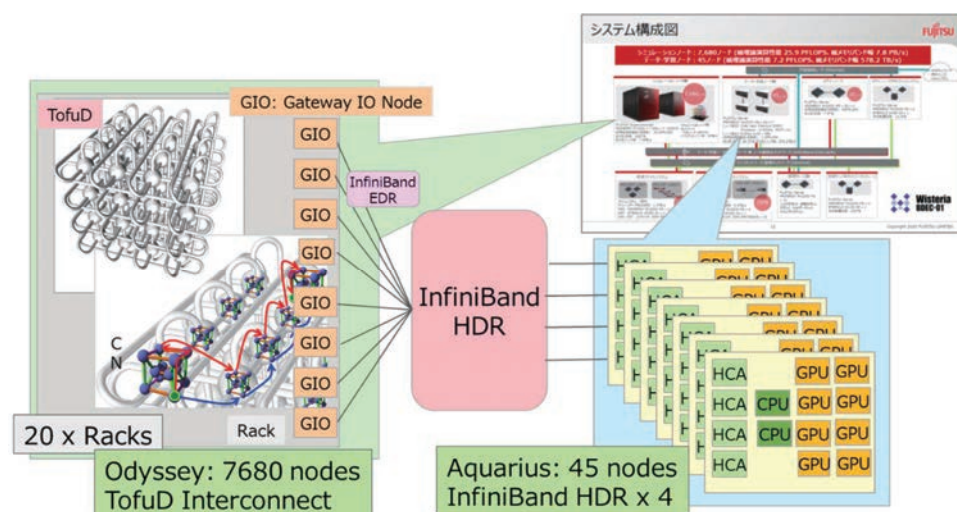


図 15. Wisteria/BDEC-01 の Odyssey-Aquarius 間のインターコネクト接続構成図

図 15 において Odyssey のノードから Aquarius のノードへの通信経路は 2 種類あり、一つは GIO から InfiniBand を用いた直接的な通信と GIO 以外のノードからの GIO を経由した通信の 2 種類である。このため、性能測定も GIO 直接の場合と GIO を経由した場合の 2 種測定した。

Odyssey 計算ノード- Aquarius計算ノード間

1ノードあたり転送性能: 350MB/s
GIO Routing性能1/2: 175MB/s
1ラックあたり 175 x 8: 1.4GB/s
全系通信性能20ラック: 28 GB/s

Odyssey GIOノード- Aquarius計算ノード間

GIO vs Aquarius計算ノード間通信性能: 370MB/s
(iperf3)
1ラックあたり 370 x 8: 2.96GB/s
全系通信性能20ラック: 59.2 GB/s

図 16. Wisteria/BDEC-01 の Odyssey-Aquarius 間の TCP/IP 転送性能ベースの通信性能見積もり

図 16 に GIO 直接と GIO 経由の場合の転送性能を元にした Odyssey-Aquarius 間の総転送性能見積もりを示す。GIO におけるルーティング処理でのオーバーヘッドにより GIO のみを用いた場合に比べ約半分の通信性能となっている。

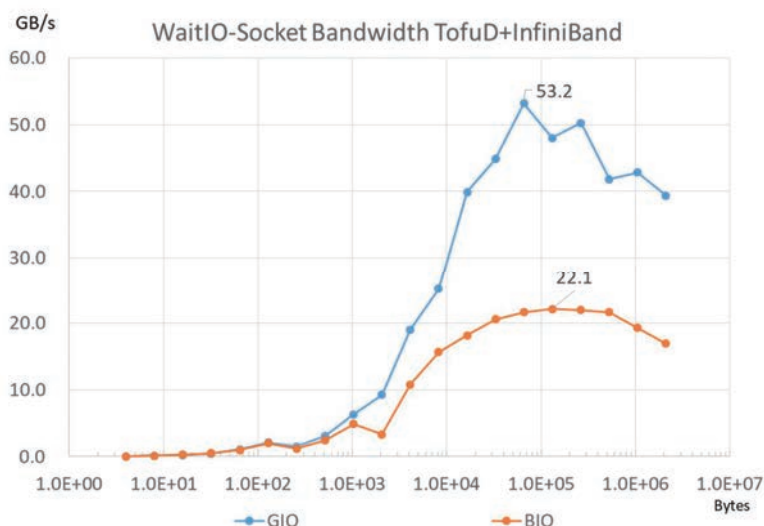


図 17. Wisteria/BDEC-01 の Odyssey-Aquarius 間の WaitIO-Socket の PingPong 転送性能

図 17 に全系を用いた通信性能の測定結果を示す。保守時の試験環境のため Aquarius は 8 ノードに限定されている他、測定時間の関係で Odyssey は全系ノードではなく 1 ラックに 24 ノード搭載されている BIO(Boot IO ノード)からの測定となっている。この GIO, BIO を選別する WaitIO Group の作成には、waitio_create_group()関数の評価関数として GIO 並びに BIO の IP アドレスが一致したものを WG のメンバとして選択する関数を作成してグループ化している。

図 17 の結果より、BIO を用いた場合(GIO のルーティングあり)は 22.1GB/秒、GIO 直接の場合には 53.2GB/秒の通信性能を得た。予測性能に比べて GIO ルーティングありの場合で 79%の性能、GIO 直接の場合で 90%の性能であった。ラックを増加すると通信性能が増しているが GIO 単体の通信性能の向上が今後の課題である。アプリケーション実行上は実行ノード数と配置を分散させることで所要の通信性能を確保できる場合もあるため、適用事例ごとに通信性能高速化の必要性を見極めていく。

11. まとめ

本稿では Wisteria/BDEC-01 利用事例として h3-Open-BDEC と h3-Open-BDEC の一つのモジュールの一つである h3-Open-SYS/WaitIO の概要(1/3)について、サンプルを含めたプログラミングと実行例を交えて紹介した。次号(2/3)では簡単なプログラムを使用したより詳細な実装例と h3-Open-UTIL/MP について紹介する。

参考文献

- [1] Wisteria/BDEC-01 (「計算・データ・学習」融合スーパーコンピュータシステム) :<https://www.cc.utokyo.ac.jp/supercomputer/wisteria>
- [2] 中島研吾, 塙敏博, 下川辺隆史, 伊田明弘, 芝隼人, 三木洋平, 星野哲也, 有間 英志, 河合直

- 聡, 坂本龍一, 岩下武史, 八代尚, 長尾大道, 松葉浩也, 荻田武史, 片桐孝洋, 古村孝志, 鶴岡弘, 市村強, 藤田航平, 近藤正章, 「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC-01」の概要, 情報処理学会研究報告 (2020-HPC-179-01), 2021
- [3] 塙 敏博, 中島 研吾, 下川辺隆史, 芝隼人, 三木洋平, 星野哲也, 河合直聡, 似鳥啓吾, 今村俊幸, 工藤周平, 中尾 昌広, 「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC-01」の性能評価, 情報処理学会研究報告 (2021-HPC-180-22), 2021
- [4] 住元真司, 荒川隆, 坂口吉生, 松葉浩也, 八代尚, 塙敏博, 中島研吾, WaitIO-Socket: 異種システム上の複数 MPI プログラムを結合する通信ライブラリの試作, 情報処理学会研究報告 (2021-HPC-181-07), 2021
- [5] Arakawa, T., Yashiro, H., Nakajima, K., Development of a coupler h3-Open-UTIL/MP, ACM Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2022), 2022
- [6] ppOpen-HPC: <https://github.com/Post-Peta-Crest/ppOpenHPC>
- [7] h3-Open-BDEC : <https://h3-open-bdec.cc.u-tokyo.ac.jp/>
- [8] Iwashita, T, Nakajima, K., Shimokawabe, T., Nagao, H., Ogita, T., Katagiri, T., Yashiro, H., Matsuba, H., h3-Open-BDEC: Innovative Software Platform for Scientific Computing in the Exascale Era by Integrations of (Simulation + Data + Learning), Project Poster, ISC-HPC 2020
- [9] Nakajima, K., Matsuba, K., Hanawa, T., Furumura, T., Tsuruoka, H., Nagao, H., Integration of 3D Earthquake Simulation & Real-Time Data Assimilation on h3-Open-BDEC, MS290: Progress & Challenges in Extreme Scale Computing & Data SIAM Conference on Computational Science & Engineering (CSE21) (Online, March 4, 2021)
- [10] SIAM News (March 10, 2021), Supercomputer Simulations of Earthquakes in Real Time (by Jillian Kunze), <https://sinews.siam.org/Details-Page/supercomputer-simulations-of-earthquakes-in-real-time>
- [11] 中島研吾, 古村孝志, 鶴岡弘, 坂口吉生, 松葉浩也, 住元真司, 八代尚, 荒川隆, 塙敏博, 観測データ同化による長周期地震動リアルタイム予測へ向けた試み, 情報処理学会研究報告 (2021-HPC-182-08), 2021
- [12] Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator. ACM/IEEE Proceedings of SC'03, <https://doi.org/10.1145/1048935.1050164>, 2003

Wisteria/BDEC-01 (Odyssey) における OpenMP によるプログラミング入門 (その 1)

中 島 研 吾

東京大学情報基盤センター

0. はじめに

2021 年 5 月 14 日に運用を開始した「計算・データ・学習」融合スーパーコンピュータシステム (Wisteria/BDEC-01) [1,2] のうち, シミュレーションノード群 (Odyssey) における OpenMP プログラミングについて 2 回 (もしかしたら 3 回) に分けて連載する. Odyssey は FUJITSU Supercomputer PRIMEHPC FX1000¹ 20 ラックから構成され, 「A64FX」を 7,680 ノード (368,640 コア) 搭載する. 各ノードは倍精度浮動小数点演算で 3.3792 TFLOPS の理論ピーク性能, 合計ピーク性能は 25.9 PFLOPS である. 各ノードは 32 GiB の HBM2 メモリを搭載し, シミュレーションノード群 (Odyssey) の総メモリ容量は 240 TiB, 総メモリバンド幅は 7.8 PB/秒である. 各ノードはバイセクションバンド幅が 13.0 TB/秒のノード間相互結合ネットワーク (Tofu インターコネクト D) で結合されている. 本連載では, A64FX の各ノード (48 コア搭載) における OpenMP 並列プログラミングについて紹介する.

紙面の都合もあるため, 詳細は当センターで実施している講習会「OpenMP によるマルチコア・メニョコア並列プログラミング入門 (Wisteria/BDEC-01 (Odyssey, A64FX 搭載))^{1,2}」のホームページ [3,4] をご覧頂きたい. 講習会の詳細な資料の他, ビデオ動画も公開されている. Wisteria/BDEC-01 の登録ユーザーは講習会資料で紹介しているサンプルプログラムを自由にコピーして学習することが可能である.

第 1 回は対象とするアプリケーションの概要と, その OpenMP 並列化について紹介する.

1. 本稿の目的

有限要素法, 差分法等の科学技術アプリケーションは最終的には大規模な疎行列を係数とする連立一次方程式を解くことに帰着される. 安定で効率的な解法の研究開発は重要な技術的課題であり, 特に昨今では大規模並列計算をターゲットとした前処理付き反復法に関する研究が盛んである [5]. 大規模なマルチコアクラスタ, メニョコアクラスタではノード間のメッセージパッシング (例: MPI), ノード内のスレッド並列 (例: OpenMP) に基づくハイブリッド並列プログラミングモデルが広く使用されている [5]. Odyssey でも, 大規模並列計算実行にあたっては, OpenMP/MPI によるハイブリッド並列プログラミングモデルの適用が推奨されている.

本稿は, ハイブリッド並列化に向けて, 各ノードにおける OpenMP 並列化のための注意事項について説明する. 本稿で扱う題材は有限体積法によるアプリケーションである. 本稿の内容は単一のアプリケーションに特化した内容であるが, 基本的な考え方は様々な分野に適用可能である.

本稿では特に有限体積法によってポアソン方程式を解くことによって得られる疎行列を係数行列とする大規模連立一次方程式を代表的な非定常反復法である共役勾配法 (Conjugate Gradient

¹ <https://www.cc.u-tokyo.ac.jp/events/lectures/173/>

² <http://nkl.cc.u-tokyo.ac.jp/seminars/multicore2021/>

Method, CG 法) [6] の並列化に主眼を置いている．ハイブリッド並列化において重要なのは、各ノード内におけるスレッド並列化である．以下、有限体積法の基礎的な考え方から始めて詳細に説明する．なお、OpenMP の文法等については文献 [7-10] を参照されたい．

2. アプリケーション (P3D) の概要

本稿で対象とするアプリケーションは図 1 に示す差分格子によってメッシュ分割された三次元領域において、以下のポアソン方程式を解くもので P3D と呼んでいる：

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f \quad (1)$$

$$\phi = 0 @ z = z_{\max} \quad (2)$$

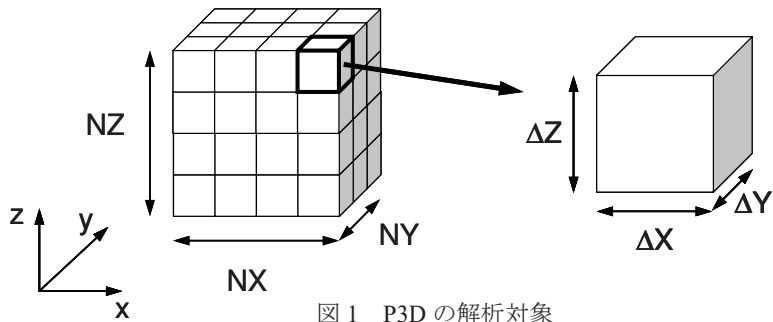


図 1 P3D の解析対象

差分格子の各メッシュは直方体 (辺長は ΔX , ΔY , ΔZ)、 X , Y , Z 各方向のメッシュ数は NX , NY , NZ

ここで、式 (1) の右辺の f は体積あたりのフラックス (flux, 流束) 項で、以下に示すような空間分布を有すると仮定している：

$$f = dfloat(i_0 + j_0 + k_0) \quad (3)$$

$$i_0 = XYZ(icel,1), \quad j_0 = XYZ(icel,2), \quad k_0 = XYZ(icel,3) \quad (4)$$

式 (4) における $XYZ(icel, k)$ ($k=1,2,3$) は X , Y , Z 方向の差分格子のインデックスで各メッシュが X , Y , Z 方向の何番目にあるかを示している．支配方程式 (1) を境界条件 (2) とフラックスの条件 (3), (4) を適用して解いた計算結果の例 (ϕ の分布) を図 2 に示す．

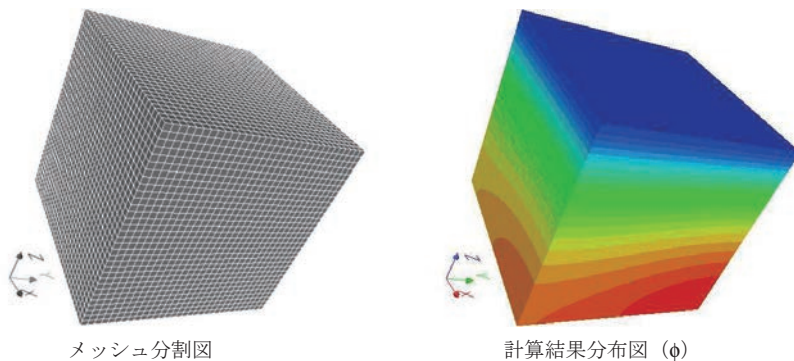


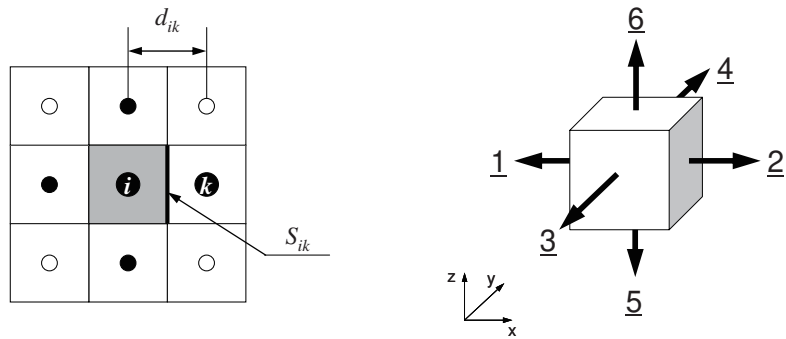
図 2 P3D の計算結果例 ($32^3=32,768$ メッシュ)

形状は規則正しい差分格子であるが、プログラムの中では、一般性を持たせるために、有限体積法に基づき、非構造格子型のデータとして考慮する．図 2 における任意のメッシュ i の各面を通過するフラックスについて、式 (1) により以下に示す式 (5) のような釣り合い式が成立する：

$$\sum_{k=1}^6 \left[\frac{S_{ik}}{d_{ik}} (\phi_k - \phi_i) \right] + V_i f_i = 0 \quad (5)$$

ここで、 S_{ik} ：メッシュ i と隣接メッシュ k 間の表面積、 d_{ik} ：メッシュ i - k 重心間の距離、 V_i ：メッシュ i の体積、 f_i ：メッシュ i の体積あたりフラックスである（図 3 (a) 参照）．三次元問題の場合、各直方体メッシュは 6 個の面を持っているため、隣接メッシュ数は（最大で）6 であり、式

(5) 左辺第一項は $k=1 \sim 6$ の和となっている．図 3 (b) は、三次元問題における各メッシュの局所面番号（すなわち隣接メッシュの番号付けのルール）である．



(a) メッシュ i と隣接メッシュ k の関係

(b) 局所面番号、隣接メッシュナンバリング

図 3 隣接メッシュとの関係

式 (5) を図 1 のような三次元領域に適用すると、メッシュ i について式 (6) が得られる：

$$\begin{aligned} & \frac{\phi_{k=1} - \phi_i}{\Delta x} \Delta y \Delta z + \frac{\phi_{k=2} - \phi_i}{\Delta x} \Delta y \Delta z + \frac{\phi_{k=3} - \phi_i}{\Delta y} \Delta z \Delta x + \frac{\phi_{k=4} - \phi_i}{\Delta y} \Delta z \Delta x + \\ & \frac{\phi_{k=5} - \phi_i}{\Delta z} \Delta x \Delta y + \frac{\phi_{k=6} - \phi_i}{\Delta z} \Delta x \Delta y = f_i \Delta x \Delta y \Delta z \end{aligned} \quad (6)$$

これを整理すると、式 (7) のようになり、三次元差分法の場合と同様の式が得られる：

$$\frac{\phi_{k=1} - 2\phi_i + \phi_{k=2}}{\Delta x^2} + \frac{\phi_{k=3} - 2\phi_i + \phi_{k=4}}{\Delta y^2} + \frac{\phi_{k=5} - 2\phi_i + \phi_{k=6}}{\Delta z^2} = f_i \quad (7)$$

式 (5) にもどって、これを整理すると、式 (8) が得られる：

$$\left[\sum_{k=1}^6 \frac{S_{ik}}{d_{ik}} \right] \phi_i - \left[\sum_{k=1}^6 \frac{S_{ik}}{d_{ik}} \phi_k \right] = +V_i f_i \quad (8)$$

これは各メッシュ i について成立する式であるので、全メッシュ数を N とすると、 N 個の方程式を連立させて、境界条件を適用し、連立一次方程式 $A\phi=b$ を解くことに帰着される。式 (8) の左辺第一項は A の対角項、第二項は非対角項、式 (8) の右辺は b に対応している。式 (8) からわかるように、各メッシュ i に対応する非対角成分の数は最大 6 個であるので、係数行列 A は疎 (sparse) な行列となる。

3. CG 法について

式 (8) を連立させて得られる連立一次方程式 $A\phi=b$ を解く方法として、本稿では反復法 (Iterative Method)、特に Krylov 部分空間法といわれる、非定常型の反復法を使用する [6]。係数行列 A は、式 (7) から類推されるように、対称かつ正定 (Symmetric Positive Definite) であり、このような行列に対しては、通常は共役勾配法 (Conjugate Gradient Method, CG 法) が使用される [6]。Krylov 部分空間法の収束は係数行列の性質 (固有値分布) に強く依存するため、前処理を適用して固有値が 1 の周辺に集まるように行列の性質を改善する。前処理行列を M とすると、 $\tilde{A}=M^{-1}A$ 、 $\tilde{b}=M^{-1}b$ として ($[M]^{-1}$ を右から乗ずる場合もある)、すなわち $\tilde{A}\phi=\tilde{b}$ という方程式を代わりに解くことになる。 M^{-1} が A^{-1} をよく近似した行列であれば、 $\tilde{A}=M^{-1}A$ は単位行列に近くなり、それだけ解きやすくなる。前処理付き CG 法のアルゴリズムの概要はリスト 1 のようになる：

```

Compute  $r^{(0)} = b - Ax^{(0)}$ 
for  $i = 1, 2, \dots$ 
    solve  $Mz^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if  $i=1$ 
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} z^{(i)}$ 
    endif
     $q^{(i)} = Ap^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence  $|r|$ 
End

```

リスト 1 前処理付き CG 法 (共役勾配法, Conjugate Gradient Method) のアルゴリズム [6]

「solve $Mz^{(i-1)} = r^{(i-1)}$ 」の部分で、前処理行列 M を係数行列とする連立一次方程式を解く必要がある。本稿では、前処理手法として単純だが広く使用されている対角スケーリング (Diagonal Scaling) を使用する。点ヤコビ法 (Point Jacobi Method) と呼ばれることもある [6]。この手法は、前処理行列 M として、元の係数行列 A の対角成分のみを取り出した対角行列 (Diagonal Matrix) D を使用するものである。 D の逆行列の各成分は対角成分の逆数となるので、リスト 1 の「solve $Mz^{(i-1)} = r^{(i-1)}$ 」では、ベクトル r の各成分を対角成分で割ることによって z が得られる。近くスケーリング・点ヤコビは、係数行列の対角成分が非対角成分より十分に大きい、「対角優位」な行列の場合、特に有効である。対角スケーリング・点ヤコビは、後述するように並列化も容易である。

4. P3D（単体 CPU 版）の処理内容と OpenMP による並列化

続いて単体 CPU 用版の P3D³ [11] の処理内容を簡単に説明する．プログラムは Fortran 版，C 版があり，それぞれについて詳細な資料が準備されているので詳しくはそちらをご覧ください [11]．係数行列は対称であるが，対角成分，非ゼロ非対角成分に分け，上下三角成分を全て記憶している．行列の格納形式としては，非零成分のみを効率的に格納できる Compressed Row Storage (CRS) [6] という方法が採用されている．各成分は表 1 のように記憶されている：

表 1 Compressed Row Storage (CRS) による疎行列格納法

変数・配列名	型	Fortran	C	内 容
ICELTOT, N	整数	-	-	全メッシュ数
NPLU	整数	-	-	全ゼロ非対角成分数
D	倍精度実数	(N)	[N]	係数行列の対角成分
X	倍精度実数	(N)	[N]	解ベクトル
W	倍精度実数	(N, 4)	[4] [N]	CG 法で使用されるベクトル，添字は R,Z,P,Q など「リスト 1」中に現れる配列名に対応，DD は対角成分の逆数
B	倍精度実数	(N)	[N]	右辺ベクトル
indexLU	整数	(0:N)	[N+1]	各行の非ゼロ非対角成分数，一次元圧縮配列 (CRS)
itemLU	整数	(NPLU)	[NPLU]	各行の非ゼロ非対角成分に対応した列番号，一次元圧縮配列 (CRS)
AMAT	倍精度実数	(NPLU)	[NPLU]	各行の非ゼロ非対角成分一次元圧縮配列 (CRS)

表 1 に示した CRS 法によって一次元圧縮配列に格納された疎行列成分を使用して CG 法に現れる行列ベクトル積 $Ap=q$ の計算を実施すると，リスト 2 のようになる．

・ Fortran

```
do i= 1, N
  VAL= D(i)*W(i,P)
  do k= indexLU(i-1)+1, indexLU(i)
    VAL= VAL + AMAT(k)*W(itemLU(k),P)
  enddo
  W(i,Q)= VAL
enddo
```

・ C

```
for (i=0; i<N; i++) {
  VAL = D[i] * W[P][i];
  for (k=indexLU[i]; j<indexLU[i+1]; j++) {
    VAL += AMAT[k] * W[P][itemLU[k]];
  }
  W[Q][i] = VAL;
}
```

リスト 2 CRS 法によって格納された疎行列成分による行列ベクトル積 $Ap=q$ の計算

³ <http://nkl.cc.u-tokyo.ac.jp/files/fvm.tar>

前処理付き CG 法（リスト 1）の処理内容は、大きく分けて以下の 4 種類である：

- ① 前処理（対角スケーリング）
- ② ベクトルの内積
- ③ ベクトルの実数倍の加減
- ④ 行列ベクトル積

前処理に対角スケーリング・点ヤコビ法のような簡単な手法を適用すると、各処理は OpenMP の指示行（directive）を挿入するのみで簡単に並列化できる．IC 法，ILU 法等のデータ依存性を含むような前処理を適用する場合には、リオーダーリングによって並列化を抽出する必要があるが、このような場合については、いずれ稿を改めて紹介したい．

リスト 3，リスト 4 は，Fortran，C においてリスト 1 中の①～④に相当する処理を OpenMP によって並列化した例である．基本的にはリスト 1 の各演算は、行列ベクトル積も含めて、各ベクトルの N 個の成分に対する個別の処理であるので、OpenMP 指示行の挿入のみで並列化が実現できるのである：

① 前処理

$\text{solve } Mz^{(i-1)} = r^{(i-1)}$

```
!$omp parallel do
do i= 1, NP
  W(i, Z)= W(i, R) * W(i, DD)
enddo
!$omp end parallel do (省略可能)
```

② 内積

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

```
RHO= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:RHO)
do i= 1, N
  RHO= RHO + W(i, R) * W(i, Z)
enddo
!$OMP END PARALLEL DO (省略可能)
```

③ ベクトル実数倍加減

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

```
!$omp parallel do
do i= 1, NP
  X(i)= X(i) + ALPHA * W(i, P)
enddo
!$omp end parallel do (省略可能)
```

④ 行列ベクトル積

$q^{(i)} = Ap^{(i)}$

```
!$omp parallel do private(i, VAL, k)
do i= 1, N
  VAL= D(i)*W(i, P)
  do k= indexLU(i-1)+1, indexLU(i)
    VAL= VAL + AMAT(k)*W(indexLU(k), P)
  enddo
  W(i, Q)= VAL
enddo
!$omp end parallel do (省略可能)
```

リスト 3 前処理付き CG 法の各処理と OpenMP による並列化 (Fortran)

⑤ 前処理

$\text{solve } Mz^{(i-1)} = r^{(i-1)}$

```
#pragma omp parallel for private (i)
for (i=0; i<N; i++) {
    W[Z][i] = W[R][i] * W[DD][i];
}
```

⑥ 内積

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

```
RHO= 0.0;
#pragma omp parallel for private (i) reduction(+:RHO)
for (i=0; i<N; i++) {
    RHO= RHO + W[R][i] * W[Z][i];
}
```

⑦ ベクトル実数倍加減

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

```
#pragma omp parallel for private (i)
for (i=0; i<N; i++) {
    X[i] = X[i] + alpha * W[P][i];
}
```

⑧ 行列ベクトル積

$q^{(i)} = Ap^{(i)}$

```
#pragma omp parallel for private (i,VAL,k)
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (k=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[k] * W[P][itemLU[k]];
    }
    W[Q][i] = VAL;
}
```

リスト 4 前処理付き CG 法の各処理と OpenMP による並列化 (C)

次回（以降）は，この並列版プログラムの Odyssey 上での実行，性能評価，更なる最適化，高速化について紹介する．

（第 2 回（5 月号）へ続く）

参 考 文 献

- [1] Wisteria/BDEC-01（「計算・データ・学習」融合スーパーコンピュータシステム）：<https://www.cc.utokyo.ac.jp/supercomputer/wisteria>
- [2] 中島研吾，塙敏博，下川辺隆史，伊田明弘，芝隼人，三木洋平，星野哲也，有間 英志，河合直聡，坂本龍一，岩下武史，八代尚，長尾大道，松葉浩也，荻田武史，片桐孝洋，古村孝志，鶴岡弘，市村強，藤田航平，近藤正章，「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC- 01」の概要，情報処理学会研究報告（2020-HPC-179-01），2021
- [3] 東京大学情報基盤センター：第 173 回お試しアカウント付き並列プログラミング講習会「OpenMP によるマルチコア・メニコア並列プログラミング入門（Wisteria/BDEC-01（Odyssey，A64FX 搭載）」，<https://www.cc.u-tokyo.ac.jp/events/lectures/173/>
- [4] OpenMP によるマルチコア・メニコア並列プログラミング入門（Wisteria/BDEC-01（Odyssey，A64FX 搭載），<http://nkl.cc.u-tokyo.ac.jp/seminars/multicore2021/>
- [5] Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator. ACM/IEEE Proceedings of SC'03, <https://doi.org/10.1145/1048935.1050164>, 2003
- [6] Saad, Y.: Iterative Methods for Sparse Linear Systems Second Edition, SIAM, 2003
- [7] OpenMP: <https://www.openmp.org/>

- [8] Chandra, R.他, Parallel Programming in OpenMP, Morgan Kaufmann Publishers, 2001
- [9] 牛島省, OpenMP による並列プログラミングと数値計算法, 丸善, 2006
- [10] Mattson, T.G., Sanders, B.A., Massingill, B.L., Patterns for Parallel Programming, Software Patterns Series (SPS), Addison-Wesley, 2005
- [11] P3D 関連資料
- ソースコード等 : <http://nkl.cc.u-tokyo.ac.jp/files/fvm.tar>
 - 解説資料 (Fortran) : <http://nkl.cc.u-tokyo.ac.jp/seminars/multicore2021/omp-f-01.pdf>
 - 解説資料 (C) : <http://nkl.cc.u-tokyo.ac.jp/seminars/multicore2021/omp-c-01.pdf>

温水冷却の効率に関する定量的評価の試み

庄司 文由、野中 丈士

理化学研究所計算科学研究センター

埴 敏博

東京大学情報基盤センター/最先端共同 HPC 基盤施設

1. はじめに

循環冷却水を用いた直接・間接水冷技術は HPC システムの冷却方式として広く浸透しており、得に世界最高性能の HPC システムではデファクトスタンダードとなりつつある。また、近年の技術革新により、高性能 CPU はより高温での動作に耐えられるようになり、エネルギー効率改善の目的としての温水冷却 (Warm Water Cooling) は国内外の多くの HPC センターおよびデータセンターで採用されている。中にはアメリカ暖房冷凍空調学会 (ASHRAE) が定める高温直接水冷 (High-Temperature Direct Liquid Cooling) ガイドラインの W3-W5 クラス ($>30^{\circ}\text{C}$) の冷却水供給温度を超える HPC システムも報告されている [1]。しかしながら、温水冷却の効果を正しく評価するためには、冷却の電力を節約できるというポジティブな点に加え、演算性能の低下や消費電力の増加等のネガティブな点も等しく考慮に入れる必要がある。我々はこれまで大規模 HPC チャレンジの一環として、最先端共同 HPC 基盤施設 (JCAHPC) の関係者及びに施設運用主体の富士通株式会社のご協力の元で、2016 年より運用している Oakforest-PACS (OFP) を用いて循環冷水温度の変更による定量的評価を進めてきた (2019 年 10 月および 2021 年 5 月) [2]。冷水温度を高く設定することで、外気による自然な冷却を活用し、冷凍機等を駆動するための電力が節約できることを定量的に評価 [3] した。また、OFP 側にインストールされている Intel Parallel Studio XE 内の単体ノードベンチマーク向け LINPACK (Intel Optimized LINPACK Benchmark for Linux) を用いて、単体ノードでの冷水温度による演算性能への影響について定量的評価も行った [4]。本稿では複数ノード (CPU) を利用するジョブへの影響を評価するために、並列有限要素法コード (GeoFEM) を用いて、冷水温度を上げた場合の消費電力の増加および演算性能の低下について定量的な分析を試みた。その際に単体ノードでの実行性能を基に性能の近い CPU 同士をグループ化し、その影響の大きさを比較する調査も行った。特定のシミュレーションコードを用いた調査結果であるが、この様に単体および複数の CPU を使うジョブを、異なる冷水温度で実行し、分析結果に基づいた効率的な施設運用の実現向けの運用手順確立の手助けになれば幸いである。

2. 実験方法

温水冷却の定量的評価という事で、通常運用時の冷却水設定温度 (12°C) とそれよりも高い温度 (18°C) への変更を施設運用側にお願いした。また、実行時の詳細データを得る目的で CPU のクロック周波数や消費電力データを一般ユーザー権限で取得できるよう turbostat のコマンド実行を可能とする設定も行って頂いた。本大規模 HPC チャレンジでは 4200 ノード (CPU) 提供されたため、 12°C と 18°C の冷却水設定温度で全ノードを用いた単体 GeoFEM を 3 回実行し、実行性

能を基に 16 ノード毎にグループ化し、並列 GeoFEM を 3 回実行し性能のバラつきを調査した。

温水冷却法では外気による自然な冷却を活用するため、冷却能力は外気温に大きく依存する。図 1 の左上は気象庁が公開している我孫子市の気温データ（10 分毎）を基に大規模 HPC チャレンジ当日の気温変化を示す。冷却水温度が 18℃に設定されている時間帯では気温が 18℃以下であったが、冷却水温度設定が 12℃の時間帯では外気温が 12℃を上回っているのが確認できる。図 1 の右上は提供された 4200 ノードのラック間での分布を示す。その際に 1 ラック（120 ノード）を全面（FRONT）、背面（REAR）に分けて表示している。このノード分布から 1 番から 18 番目の縦ラック群（R01-R18）と 19 番から 36 番目の縦ラック群（R19-R36）が主に利用されていることがわかる。図 1 の下段には両ラック群（R01-R18、R19-R36）への冷却水供給温度の推移を示す。これらの図からラックの全面と背面での冷却水供給温度の差のみならず、ラック群によっても冷却水供給温度に差があるのが確認できる。

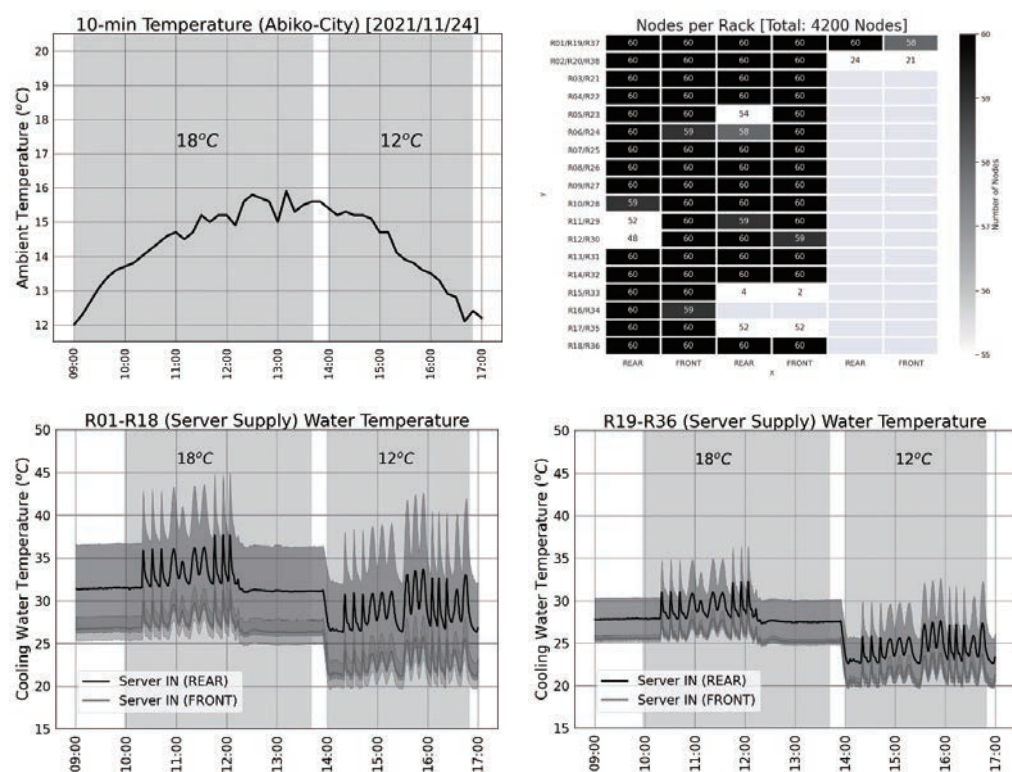


図 1 当日の我孫子市の温度変化（左上）、ラック内の利用ノード数（右上）、R01-R18 ラック群への冷却水供給温度の推移（左下）、R19-R36 ラック群への冷却水供給温度の推移（右下）

3. 単体ノード性能

表 1 に単体ノード性能を得るために利用した GeoFEM に実行パラメータと問題サイズを示す。これはウィークスケーリングのベンチマーク向けに用意された設定ファイルであり、単体ノード（CPU）であっても 8 プロセス（各 8 スレッド）で実行を行う設定である。図 2 の左上は単体ノードでの GeoFEM 実行時間（3 回）の最も早かった時間（BEST TIME）の分布を表している。ピークが 2 つあることが確認できるが、実行時間の分布幅は数%と影響が少ないことから、今回はこの点についての詳細な分析は行っていない。重要なのは、冷却水設定温度を変更しても GeoFEM

の実行性能にはあまり影響がない点である。図 2 の右上は GeoFEM 実行時のノード単位の消費電力を表している。実行性能とは異なり冷却水設定温度が高くなると消費電力がわずかに増えるということも確認できる。図 2 の下段は各冷却水設定温度域での GeoFEM 実行時間を基にグループ化 (16 ノード) した 256 グループの箱ひげ図である。この箱ひげ図の箱の部分 (色が濃い部分) はデータの中央 50% の分布を表しており、ひげの先端は最小値と最大値を表している。また、箱ひげから離れた部分に点として表示されているものは外れ値である。箱ひげ図の下部分は最も早かった時間 (BEST TIME) を表しており、この数値を元に右から左にかけてソートしてプロットしている。この図からは、最も遅かった時間つまり、箱の上部分と、最も速かった時間との間に相関が見られず、その傾向は冷水温度を変えても変わらないことが分かる。

GeoFEM	8 Process per Node
	8 Threads per Process
	Problem Sizes: 256 x 128 x 128 (1 Node)
	1024 x 256 x 256 (16 Nodes)
	1024 x 512 x 256 (32 Nodes)

表 1 GeoFEM 向け実行パラメータ (単体ノード、並列処理)

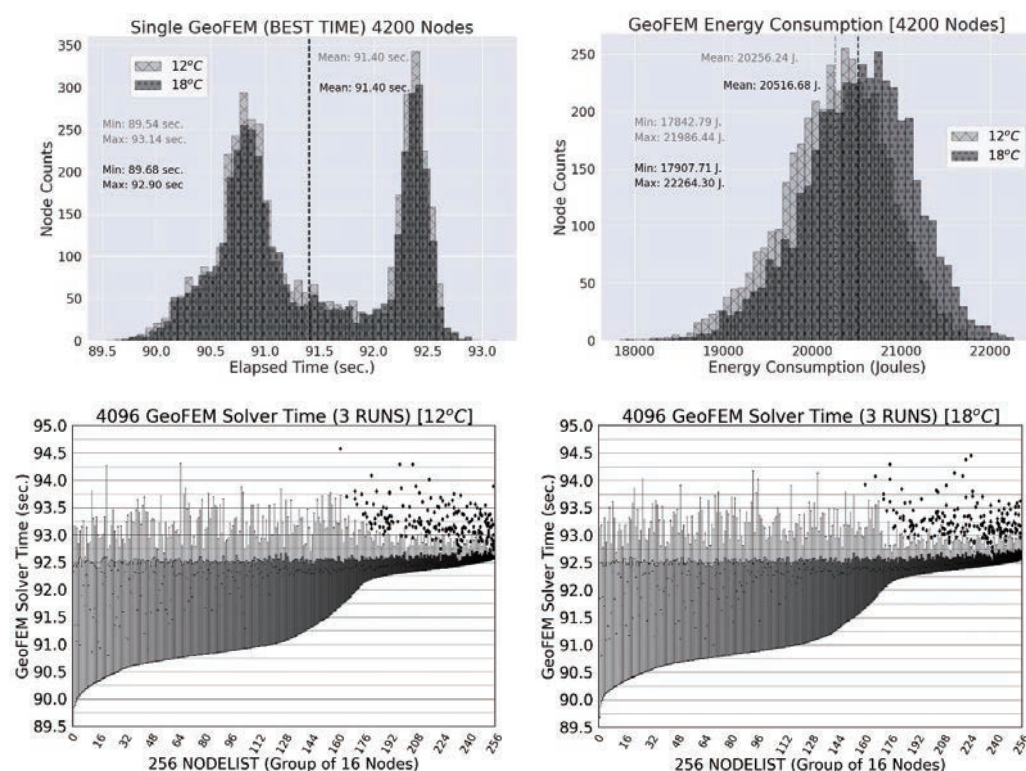


図 2 両設定温度域での単体ノード GeoFEM 実行時間 (左上) と消費電力 (右上) の分布、GeoFEM 実行時間を元にグループ化 (16 ノード) した 256 グループの箱ひげ図 (下段)

図 3 には 12°C の冷却水温度での単体ノード GeoFEM を実行した際の CPU 周波数の挙動を表して

いる。LINPACK での CPU 周波数の挙動[4]と違い、実行時では AVX512 利用による CPU 周波数の低下（周波数スロットリング）[6]が発動されないため、コンスタントに最高周波数（1.5GHz）で実行されるのが確認できる。図 3 の右側には色が濃い時間帯（15 秒～85 秒）で最高周波数を下回る 57 ノードを表している。この 57 ノードの周波数が低下した原因については別途分析が必要である。

全般的に見ると、LINPACK では周波数スロットリングが発動される際に CPU によって異なる周波数で推移していたが、GeoFEM はメモリバンド律速型のため、演算器の稼働率が低く、周波数スロットリングが発動される機会が（ほとんど）ないため、実行性能にも影響が少なく、冷水温度を変えてもその傾向は変わらないと考察できる。

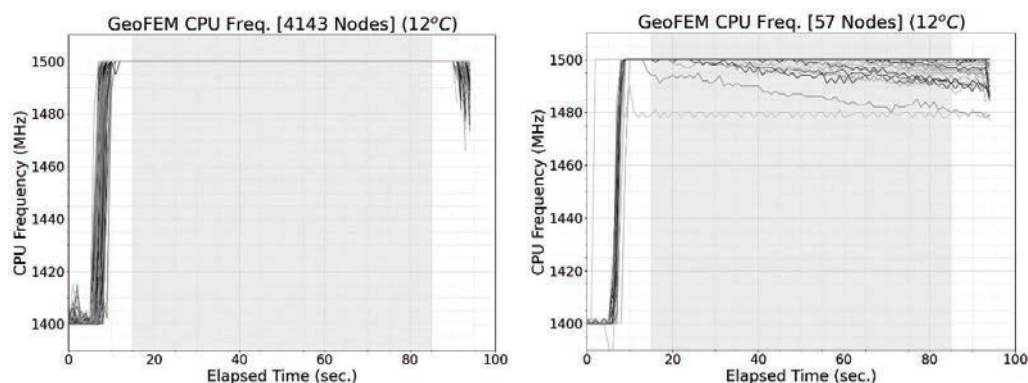


図 3 単体ノード GeoFEM 実行時の CPU 周波数の挙動

4. 並列 GeoFEM 性能

図 4 は 12°C と 18°C の冷却水温度域での並列 GeoFEM を実行した際の実行時間を表している。上段は各 16 ノードの 256 グループでの最も早い実行時間（BEST TIME）をプロットしている。グループ番号は最も早かったノードを順番にグルーピング化しているため、性能順のグループ番号である。左寄りに最も早かったノードが集まっているグループであり、右寄りに最も遅かったノードが集まっているグループであるが、256 のグループ間での性能差がほとんど見られなかった。図 4 の上段は冷却水温度を変更した際の実行時間を表しているが、グループ間での明確な差異が見られなかった。また、下段の左側は 12°C の冷却水温度設定で 32 ノードの並列 GeoFEM の実行時間をプロットしたものであるが、同じようにグループ間（128）での性能差は数%に留まった。右側の図は真上の図から CPU 周波数が異なる挙動を見せていた 57 ノード（図 3）が入っているグループを排除したプロットである。図 5 は既に[2]で報告している冷却水温度設定の変更による単体ノード LINPACK 性能と消費電力の変化であるが、図 2 の単体ノード GeoFEM との比較の為にあえて表している。これらの図を比較すると冷却水温度設定を上げた場合は AVX512 処理も利用する CPU インテンシブな LINPACK では消費電力の上昇幅は少なく、AVX512 を利用しない GeoFEM の方が明確に確認できる。しかしながら、施設の消費電力削減[3]のメリットの方が上回る可能性が高いとも考えられる。また、GeoFEM の様な CPU インテンシブでないジョブを実行する際にはジョブスケジューラが割当てするノードの影響が少ないとも考えられる。しかしながら、AVX512 を利用する CPU インテンシブのジョブは割当てられるノードによっては性能のバラつきが出てくる可能性が否定できないため、更なる調査が必要であると考ええる。

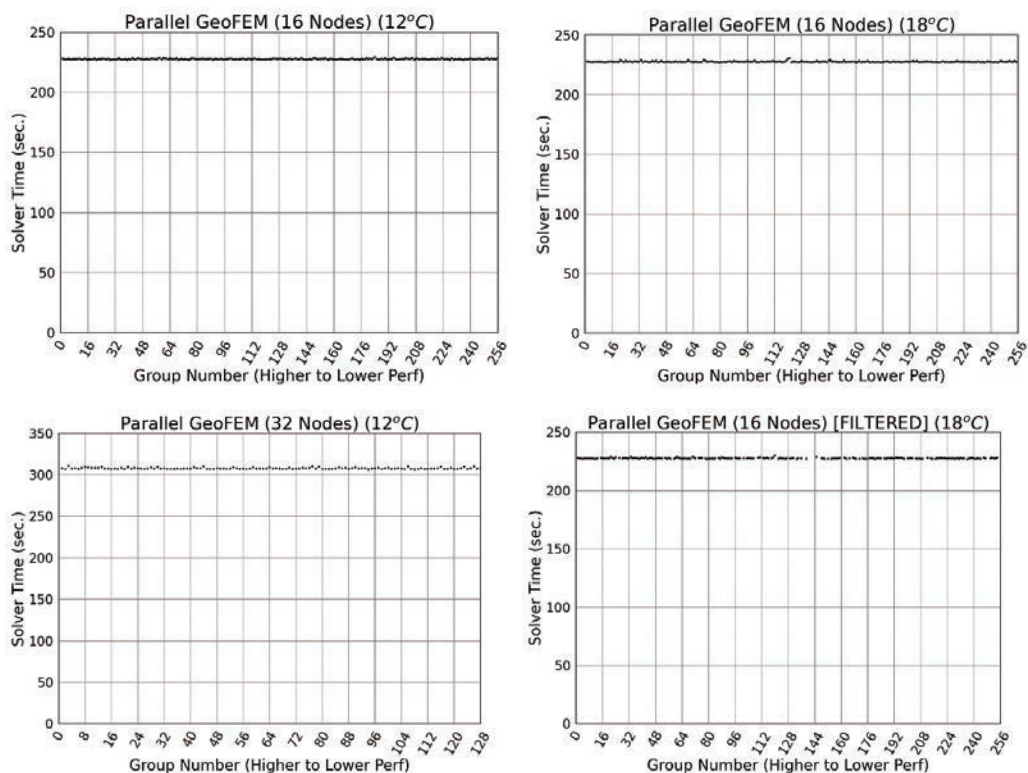


図 4 ソーティングされたグループ毎の並列 GeoFEM (16、32 ノード) の実行時間

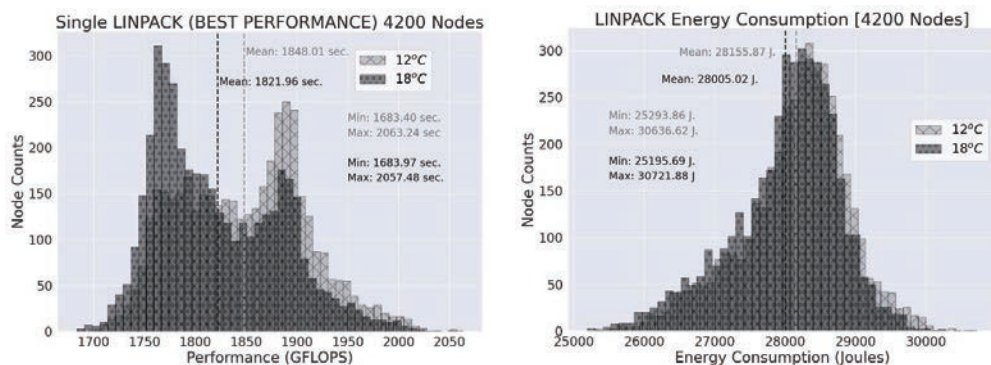


図 5 両設定温度域での単体ノード LINPACK 性能 (左) と消費電力 (右) の分布

5. まとめ

今回の実験は、冷水温度を高くしていった場合の性能に対する影響を、複数ノードジョブに対して定量的に評価するという目的で実施した。ノード数が増えた場合の影響を正しく評価するためにウィークスケーリングの評価が可能な GeoFEM を用いた。制限時間の問題等で主に 16 ノードのグループでの測定のみとなったが、単体での GeoFEM 性能の順に CPU をソーティングし、グループ化して、グループ間の並列性能の比較を行った結果、GeoFEM の様なメモリバンド律速で AVX512 処理を利用しないコードでは、冷却水の温度を上げて運用しても、クロック周波数の低下は起きにくく、結果としてジョブの実行性能への影響は小さいことが観測された。しかしなが

ら、HPLのような演算律速なコードでは、冷却水の温度が上がるとクロック周波数の低下とそれによる性能低下が発生しやすくなり、並列ジョブの場合はその影響がより顕著になると予想される。冷却水温度の上昇に対する反応にはCPU毎に差があり、実行性能に加えて、その再現性にも影響が出る可能性がある。実行のたびにパフォーマンスが変わるとなると、課金制度にも影響する。運用的には、より特性に近いノードを割り当てることで、並列ジョブの実行性能への影響を減らすことができるかも知れない。決定的な結論に至るまでには更なるデータの採取と定量的評価が必要であるが、本稿が分析結果に基づく効率的な施設運用の実現向けの運用手順確立の手助けになれば幸いである。

6. 謝辞

本研究は、大規模HPCチャレンジという制度がなければ到底実現できなかった。このような機会を与えていただいたことに深く感謝致します。また、今回の実証実験を行う上で、JCAHPCの関係者、運用主体の富士通株式会社の皆様に大変なご苦勞をおかけしました。お詫びと共に深く感謝申し上げます。GeoFEMのコードの提供およびOFP上での実行に関するサポートについて、東京大学情報基盤センター 中島教授およびIntel 株式会社 堀越氏に多大なご協力をいただきました。

参 考 文 献

- [1] H. Shoukourian et al. “SuperMUC – the first high-temperature direct liquid cooled petascale supercomputer operated by LRZ,” in Contemporary High Performance Computing: From Petascale toward Exascale, Volume 3, J. S. Vetter, Ed. CRC Press, 2019, ch. 10.
- [2] 庄司文由, 野中丈士, 塙敏博, “温水冷却の効率に関する定量的評価の試み,” スーパーコンピューティングニュース Vol.22 No.6 (2020年11月) https://www.cc.u-tokyo.ac.jp/public/VOL22/No6/09_2020shoji.pdf
- [3] J. Nonaka, T. Hanawa and F. Shoji, “Analysis on the Impact of the Cooling Water Temperature on the HPC System and Facility – Case Study: Oakforest-PACS (OFP) System and Facility,” ISC2020, Frankfurt, Germany, 2020. Available: <https://2020.isc-program.com/presentation/?id=post131&sess=sess325>.
- [4] J. Nonaka, T. Hanawa and F. Shoji, “Analysis of Cooling Water Temperature Impact on Computing Performance and Energy Consumption,” 2020 IEEE International Conference on Cluster Computing (CLUSTER), Kobe, Japan, 2020, pp. 169–175, doi: 10.1109/CLUSTER49012.2020.00027.
- [5] K. D. Stroup and P. Peltz, “Measuring and mitigating processor performance inconsistencies,” in Cray User Group Conference (CUG 2019), 2019. [Online]. Available: <https://cug.org/proceedings/cug2019proceedings/includes/files/pap124s2-file1.pdf>
- [6] Mathias Gottschlag, Frank Bellosa, “Mechanism to Mitigate AVX512-Induced Frequency Reduction,” Technical Report, Karlsruhe Institute of Technology, 2018. <https://arxiv.org/abs/1901.04982>

面内空孔が層状鉱物の原子スケールの摩擦特性へ与える影響

奥田花也

東京大学大学院理学系研究科地球惑星科学専攻

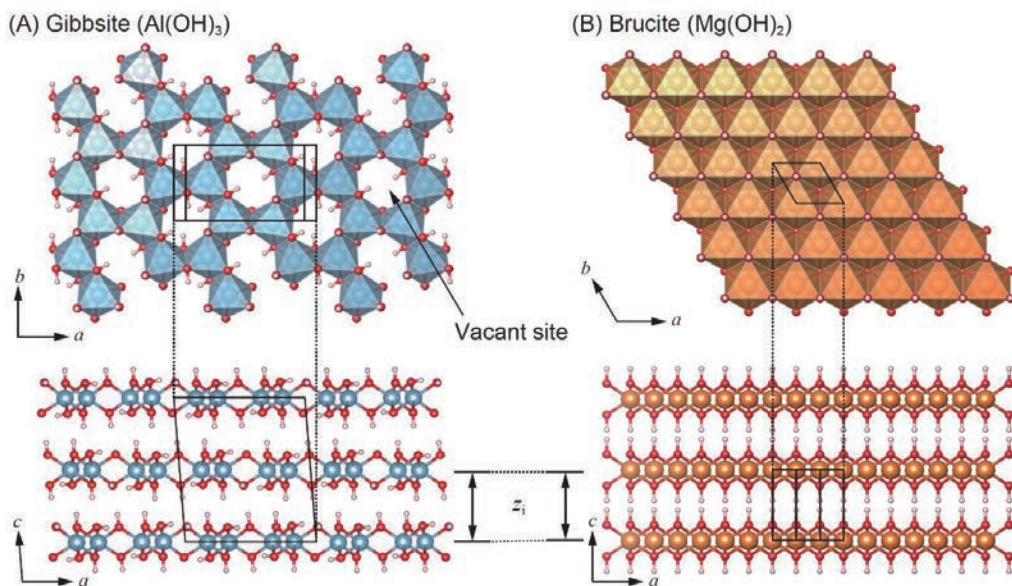
1. はじめに

地震は断層における剪断現象であり、断層がどのように滑るか、すなわち断層の摩擦特性の理解は地震発生プロセスの理解に重要である。天然の断層では結晶構造が層状である層状鉱物が多く存在することが知られる。層状鉱物はその他の一般的な岩石に比べて低い摩擦係数を示すことが実験的に知られており (Byerlee, 1978), 2011 年の東北地方太平洋沖地震で活動した日本海溝のプレート境界断層には層状鉱物の一つであるスメクタイトが多く含まれていることが地震時の巨大な滑りに貢献した可能性が指摘されている (Chester et al., 2013)。また層状鉱物の摩擦係数は物質によって異なることが知られており (Moore & Lockner, 2004; Sakuma & Suehara, 2015), 層状鉱物の低くかつ多様な摩擦係数がどのような物理背景によって決定されているかを明らかにすることは天然の断層挙動の理解に重要である。これまでの研究から、層状鉱物の層間における原子スケール摩擦に着目することによっていくつかの層状鉱物の摩擦係数が理論的に求められる可能性が指摘されてきたが (Sakuma et al., 2018; Okuda et al., 2019), この手法が断層物質のような粉体の集合体の摩擦に適用可能かを知るためにさらなる研究が求められていた (Sakuma et al., 2020)。そこで本研究では結晶構造に近いが摩擦係数の異なる二つの層状鉱物 - brucite と gibbsite - の原子スケール摩擦を計算し、結晶構造が原子スケール摩擦に与える影響を調べた。

2. 対象物質・計算手法

本研究では単位格子当たりの原子数が比較的少ない水酸化マグネシウム $\text{Mg}(\text{OH})_2$ の層状鉱物 brucite および水酸化アルミニウム $\text{Al}(\text{OH})_3$ の層状鉱物 gibbsite を用いた。Gibbsite は層内に空孔が存在する一方で brucite は層内に空孔は存在しない (図 1)。実験で得られた摩擦係数は gibbsite が 0.8 であるのに対し brucite は 0.4 と低いことが知られる (Moore & Lockner, 2004)。なお brucite の計算結果は Okuda et al. (2019) で報告したものである。

層状鉱物の原子スケール摩擦は、隣り合う層同士の相対的な変形 x に伴うポテンシャルエネルギー変化 $V(x)$ の空間微分から記述される (Zhong & Tománek, 1990)。本研究では約 0.3 Å 刻みで (001) 面上での変形および異なる層間距離を再現し、その際の安定な原子配置および $V(x)$ を計算した (gibbsite では合計 7,888 点)。 $V(x)$ は第一原理電子状態計算ソフトウェア Quantum ESPRESSO (Giannozzi et al., 2009) を用いて密度汎関数法に基づいた第一原理電子状態計算により求めた。交換相関汎関数には PBE (Perdew et al., 1996), van der Waals 力の補正には DFT-D2 法 (Grimme, 2006), 擬ポテンシャルには GBRV pseudopotential (Garrity et al., 2014), カットオフエネルギーは 40.0 Ry を波動関数に, 320.0 Ry を電子密度に対して採用, K 点メッシュ (Monkhorst & Pack, 1976) は $6 \times 6 \times 4$, エネルギーの収束条件には 0.01 mRy, 力の収束条件には 0.1 mRy/Bohr を用いた。本研究では東京大学情報基盤センターの Reedbush-U システムを主に用い, 2 ノード 64 コア並列環境によって 3 次元の電子密度の FFT 計算を並列化した。



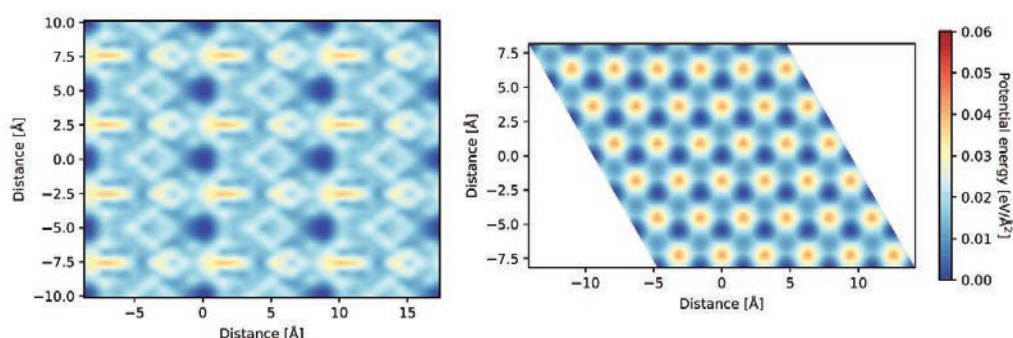
第1図: Gibbsite および brucite の結晶構造。

Gibbsite には空孔 (vacant site) が存在するが brucite には存在しない。空孔の存在以外には二つの鉱物の結晶構造は非常に似ているが、室内実験で得られる摩擦係数は異なる。

3. 計算結果

図2に gibbsite と brucite のポテンシャルエネルギー変化 $V(x)$ の結果を示す。Gibbsite も brucite も、計算した二層のうち上の層から下向きに出ている OH 基と、下の層から上向きに出ている OH 基の距離が近くなる点においてポテンシャルエネルギーが高くなることが明らかとなった。また gibbsite に関しては、OH 基の一部が層と平行に寝ているため、上下の層の OH 基が近くなる頻度が brucite に比べて低く、ポテンシャルエネルギーが高くなる間隔が brucite よりも長いことも見られた。OH 基の一部が寝る現象は面内に空孔が存在することが理由であり、そのため空孔の存在は原子スケールにおける剪断に伴うポテンシャルエネルギー変化を「ならす」効果がある。

得られたポテンシャルエネルギー変化から原子スケールの摩擦特性の計算を行った。Brucite に比べて「ならさ」れている gibbsite のポテンシャルエネルギー変化の空間微分を計算すると、原子スケールにおける gibbsite の剪断応力は brucite のおよそ半分であることが計算された。この原子スケールの剪断応力を室内実験で得られる摩擦係数と比較するためには、物質の降伏強度の情報が必要である (Sakuma et al., 2018)。先行研究の値を用いて gibbsite の摩擦係数を求めると 0.4 程度であり、brucite より少し高い結果が得られた (brucite は 0.3 程度)。一方で室内実験における gibbsite の摩擦係数は 0.8 程度であり、理論と実験結果は整合的ではなかった。これは本研究が単結晶の変形を再現しているのに対し、実験は粉末の集合体で行われたものであり、粉体の相互作用も摩擦係数に大きな影響を及ぼしているということを示している。



第2図: Gibbsite および brucite のポテンシャルエネルギー変化。

左図が gibbsite のポテンシャルエネルギー変化, 右図が brucite のものである。垂直応力 2.5 GPa を仮定。ポテンシャルエネルギーが高い点では上下の層から出ている OH 基同士の距離が近くなっている。

4. まとめと今後の展望

本研究の結果は理論計算と実験の間にまだ差があることを明確にした。これはおそらく粉体の相互作用によるものであり, 本研究のような原子スケールと室内実験のスケールの間をつなぐメソスケールの計算が将来必要になると考えられる。一方で原子スケールの摩擦特性は原子配置に大きく影響を受けており, メソスケールの物性にも影響を及ぼしていると予想されるため, 今回得られた結果は将来のスケールアップに向けた研究の基礎となるものと期待される。

謝 辞

本研究は東京大学情報基盤センターの Reedbush-U システムを用い, 東京大学情報センター「若手・女性利用者推薦」平成 30 年度後期課題および 2019 年度前期課題, 学際大規模情報基盤共同利用・共同研究拠点(JHPCN)「萌芽型研究課題」EX18323 および EX19302 によって行われた。また河合研志准教授(東京大学理学系研究科地球惑星科学専攻), 佐久間博(国立研究開発法人物質・材料研究機構)には多くの指導, 助言をいただいた。

参 考 文 献

- Byerlee, J. (1978). Friction of rocks. *Pure Appl. Geophys.*, 116, 615-626.
- Chester, F. M., Rowe, C., Ujiie, K., Kirkpatrick, J., Regalla, C., Remitti, F., ... Toczko, S. (2013). Structure and composition of the plate-boundary slip zone for the 2011 Tohoku-Oki earthquake. *Science*, 342, 1208-1211.
- Garrrity, K. F., Bennett, J. W., Rabe, K. M., & Vanderbilt, D. (2014). Pseudopotentials for high-throughput DFT calculations. *Comput. Mater. Sci.*, 81, 446-452.
- Giannozzi, P., Baroni, S., Bonini, N., Calandra, M., Car, R., Cavazzoni, C., ... Wentzcovitch, R. M. (2009). QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *J. Phys. Condens. Matter*, 21, 395502.
- Grimme, S. (2006). Semiempirical GGA-type density functional constructed with a long-range dispersion correction. *J. Comput. Chem.*, 27, 1787-1799.
- Monkhorst, H. J. & Pack, J. D. (1976). Special points for Brillouin-zone integrations.

- Phys. Rev. B, 13, 5188-5192.
- Moore, D. E. & Lockner, D. A. (2004). Crystallographic controls on the frictional behavior of dry and water-saturated sheet structure minerals. J. Geophys. Res., 109, B03401.
- Okuda, H., Kawai, K., & Sakuma, H. (2019). First-principles investigation of frictional characteristics of brucite: An application to its macroscopic frictional characteristics. J. Geophys. Res. Solid Earth, 124, 10423-10443.
- Perdew, J. P., Burke, K., & Ernzerhof, M. (1996). Generalized gradient approximation made simple. Phys. Rev. Lett., 77, 3865-3868.
- Sakuma, H., Kawai, K., Katayama, I., & Suehara, S. (2018). What is the origin of macroscopic friction? Sci. Adv., 4, eaav2268.
- Sakuma, H., Kawai, K., & Kogure, T. (2020). Interlayer energy of pyrophyllite: Implication for macroscopic friction. Am. Min., 105, 1204-1211.
- Sakuma, H. & Suehara, S. (2015). Interlayer bonding energy of layered minerals: Implication for the relationship with friction coefficient. J. Geophys. Res. Solid Earth, 120, 2212-2219.
- Zhong, W. & Tománek, D. (1990). First-principles theory of atomic-scale friction. Phys. Rev. Lett., 64, 3054-3057.

科学技術計算の効率的超並列化に向けた

静的負荷分散を行う DSL の開発

西 田 秀 之

東京大学情報理工学系研究科創造情報学専攻

1. はじめに

科学技術計算は、科学の諸原理をプログラムの形で実装し、計算機を用いて物性計算を行う。計算機科学の発展に伴って多岐に渡る分野の研究的・産業的な活用が成されている。計算資源の節約及び時間的な要求により、科学技術計算のプログラムは高速に実行できることが望ましい。しかし、スーパーコンピュータを含めた複雑化している計算機環境を十全に活かせる科学技術計算プログラムの実装は試行錯誤を要する。

実際のプログラム開発においては、対象分野（ドメイン）に関する知識と計算機に関する知識を組み合わせることでプログラムの高速化・最適化が行われる。ライブラリを代表として、ソフトウェア開発効率を向上させる手段は古くから存在する。近年、ドメイン特化言語（Domain Specific Language : **DSL**）が注目されている。

多くの科学技術計算分野では、計算速度・研究の蓄積の点から、Fortran・C++が使われてきた。これらは汎用言語と呼ばれ、汎用的な用途に使えるように設計されている。これに対して、DSL は特定の領域での使用に特化している。そのため、言語の表現力に制限がかかるものの、特定領域においては開発効率性や性能が高いプログラムを書きやすくなっている。中でも、汎用言語との併用に優れる内部ドメイン特化言語（Embedded Domain Specific Language : **EDSL**）が存在する。EDSL は、汎用言語のライブラリとして使用することができ、汎用言語の機能を利用して独自の記法や機能を実現する。本研究では、スーパーコンピュータなどの超並列環境並びに特殊な計算機環境の性能を活かせるようにプログラムを書くことができる EDSL を設計し、科学技術計算プログラム開発者の支援を目指す。

2. 科学技術計算

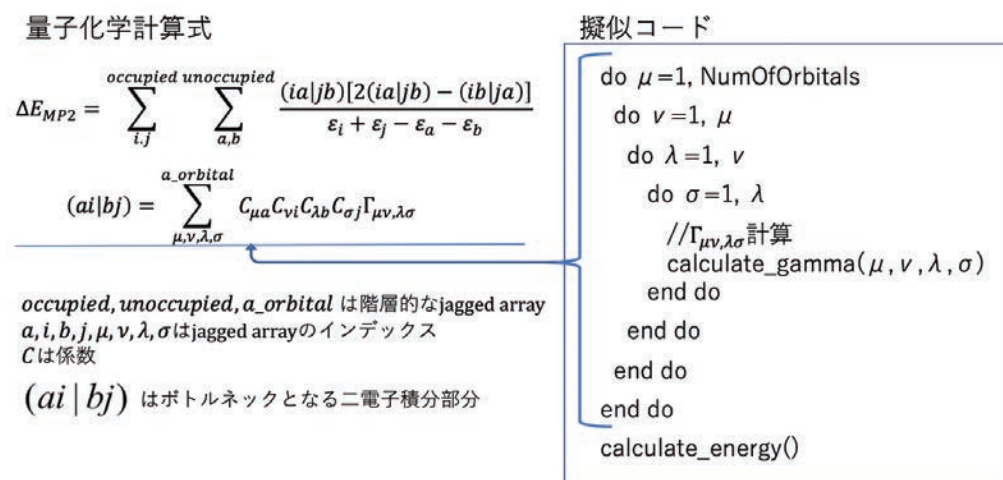
本節では、科学技術計算の一例として量子化学計算を取り上げ、その高速化における難点の一部を紹介する。

量子化学計算は量子力学を用いて物性計算を行う化学の一分野である。分野の発展と計算機技術の発展に伴い、複雑な理論の実装や高速化が行われてきた。アプリケーションプログラムを最適化する他にも、計算機環境を想定した上で理論の調整を行う取り組みも行われて来た。高速なプログラムの実装を困難にする要因の例を以下に挙げる。

(1) プログラムの複雑性により、計算量の把握が難しい。

量子化学計算では、対象物の電子を考慮して計算を行う。(厳密には電子軌道を考慮している。) ある種の N 対計算である電子対の組み合わせの計算が行われ、計算手法にもよるが電子数 n の累乗の計算量が必要と言われる。 N 対計算であるために、ある電子対を特定・計算するためには階

層的で複雑な配列 (jagged array) に多数のインデックスを用いる必要がある。そのため、プログラムは入れ子が深い多重ループ構造をとり、各ループの計算量の把握は困難なものとなる。しかし、コードの高速化には、計算量に応じた最適化も求められる。



第1図：量子化学計算式とその擬似コード。

左側が量子化学計算中でも時間がかかる二電子積分部分の計算式の一例。

右側は積分部分を表す多重ループの擬似コード。

(2) 入力データの人的な把握管理が困難である。

プログラムの多くは、決められたフォーマットの入力ファイルを用いて計算を行う。計算量は入力データの内容に依存するため、高速性を求めるならば入力データに応じた最適化が欠かせない。以下が量子化学計算プログラムの入力データの一例である。

```

$basis gbasis=n31 ngauss=6 ndfunc=1
$end
$coordinate
H 1.0 -0.559000 -8.242000 -1.948000
O 8.0 0.302000 -7.920000 -1.673000
C 6.0 1.298000 -7.649000 -2.677000
H 1.0 0.927000 -6.886000 -3.362000
$end
```

上部には計算手法などの情報、下部には計算対象の物質情報が記述されている。しかし、このファイルには計算に必要な電子の情報は明示的に記載されず、それらは暗黙的なデータとして自動で生成される。この暗黙的なデータは原子数の数倍～数十倍の巨大なデータになるため、人的な計算量の把握は困難である。例えば、ある分子を部分系に切り分けて計算する手法においては、

部分系毎の電子データが暗黙的に生成される。部分系の計算量は電子データの大きさに依存するため、データによっては部分系毎に計算時間に大きな偏りが発生する場合がある。特に超並列計算が必要な巨大な入力データまで想定すると、明示的に記述されていた場合でも人的な把握が困難となる。そのため、入力データを考慮してのプログラムの最適化も困難である。

(3) 最適化手法の適切な選択が難しい。

多くの研究により、プログラムを高速化するための手法は多数開発されてきた。量子化学計算においても、それらの手法の活用が積極的に行われている。しかし、すでに複雑なプログラムを適切に書き換えるのは簡単ではない。例えば頻繁に用いられる、OpenMP と MPI のハイブリッド並列手法が存在する。ここでは簡単のため、OpenMP と MPI はプラグマとして多重ループに挿入されると考える。標準的な量子化学計算と部分系に分割する手法にハイブリッド並列を入れた擬似コードを以下に示す。

```
#pragma MPI
for(y=0;y<MAXLENY;y++){//電子1
  #pragma MP
  for(x=0;x<MAXLENX;x++){//電子2
    --- inside algorithm ---
  }
}

#pragma MPI
for(z=0;z<MAXLENZ;z++){//部分系
  #pragma MP
  for(y=0;y<MAXLENY[z];y++){//電子1
    for(x=0;x<MAXLENX[z];x++){//電子2
      --- inside algorithm ---
    }
  }
}
```

第2図：ハイブリッド並列の例。

左側が標準的な量子化学計算にハイブリッド並列を入れた擬似コード。

右側が部分系に分割する手法にハイブリッド並列を入れた擬似コード。

MPI は粗粒度ループに適用される場合が多いため、それぞれ異なる for 文に対してプラグマが挿入される。しかし、MPI により生じるオーバーヘッドも考慮すると、どのループに MPI を挿入すべきかは入力データに依存する。例えば、部分系の中でもインデックスが 0 と 1 の計算量が多い場合には、以下のようにループ自体を分割したコードも考えられる。

```
for(z=0;z<2;z++){//部分系
  #pragma MPI
  for(y=0;y<MAXLENY[z];y++){//電子1
    #pragma MP
    for(x=0;x<MAXLENX[z];x++){//電子2
      --- inside algorithm ---
    }
  }
}

#pragma MPI
for(z=2;z<MAXLENZ;z++){//部分系
  #pragma MP
  for(y=0;y<MAXLENY[z];y++){//電子1
    for(x=0;x<MAXLENX[z];x++){//電子2
      --- inside algorithm ---
    }
  }
}
```

第3図：ループ分割の例。

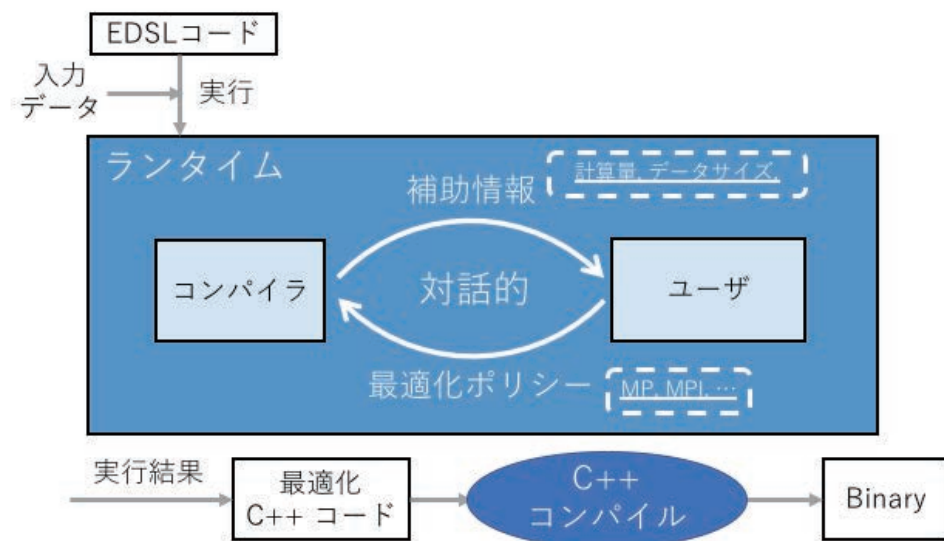
インデックスが 0 と 1 の場合、
別のループで計算させる。

この擬似コードでは、プラグマの挿入位置が異なるループを二つ記述して、MPI が作用するループを分けている。これまでの図は擬似コードであり、実際はより大きな多重ループが用いられる。それらでは最適化可能性が増大するために、最適化がより困難なものとなる。

3. Jupyter 併用型 DSL

本研究では、ユーザとコンパイラの知見を相補的に活用してプログラムの最適化を行う環境構築を目指す。具体的には、入力データを考慮した静的な負荷分散をユーザとともに行うために、インタラクティブな環境でユーザが最適化の指示を対話的に行える DSL を設計する。最適化のための補助情報が DSL のコンパイラから提示され、それを見てユーザは適切な最適化ポリシーを指示する。ユーザが記述する DSL のプログラムは実際に科学技術計算を行うプログラムそのものではなく、科学技術計算プログラムを生成するためのプログラムとなる。

システムの概念図を以下に示す。



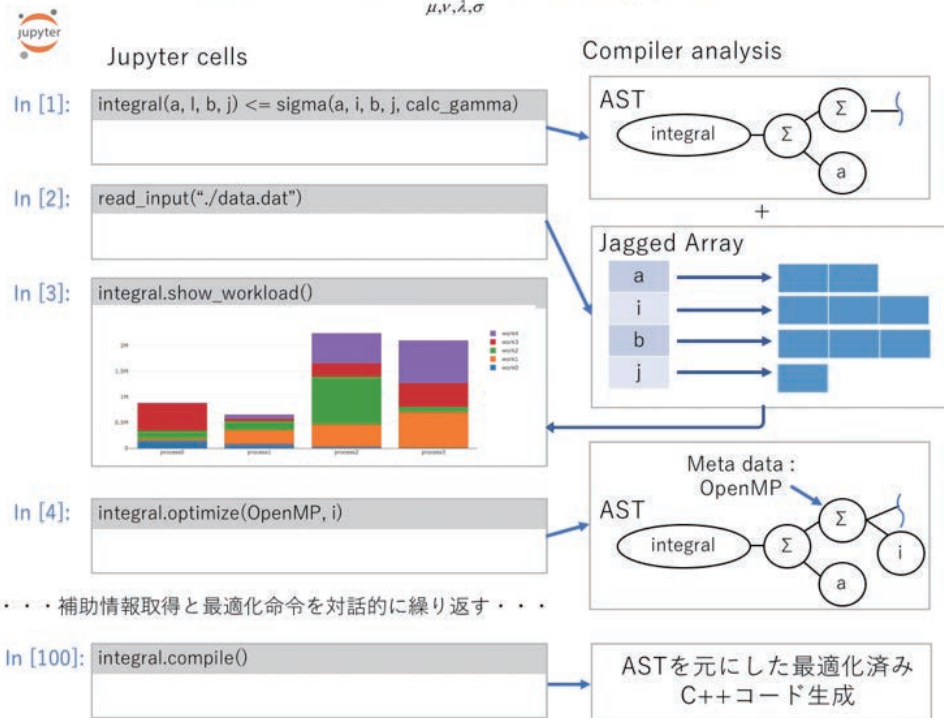
第4図：開発するシステムの概念図。

ユーザが記述した EDSL コードは実行時にユーザとやり取りをしながら、最後に科学技術計算用プログラムを生成する。

本システムではユーザが記述した DSL プログラムは二段階に分けてコンパイル・実行される。本システムの特徴は一段階目の実行時に、ユーザとコンパイラが対話的な環境を通じてプログラムの最適化を行うことである。コンパイラは事前に入力データを読み込んでいるため、データサイズや並列時の計算量などの補助情報を計算しユーザに提供する。これは、人的な把握が困難な部分をコンパイラが吸収・アシストする仕組みとなる。ユーザは補助情報を参考に自身の知見と合わせて、最適化ポリシーを決定する。ユーザの指示により二段階目の実行が行われ、実際に科学技術計算を行うプログラムを生成する。

本研究では、このシステムを Jupyter 上で実現した。Jupyter は広く使われる開発環境であり、対話的な操作を提供している。本研究では、EDSL のホスト言語となるのは Ruby であり、出力するのは実行速度の観点から C++ プログラムである。Jupyter 上での実行の様子を次に示す。

$$\text{計算式} \quad (ai|bj) = \sum_{\mu, \nu, \lambda, \sigma} C_{\mu a} C_{\nu i} C_{\lambda b} C_{\sigma j} \Gamma_{\mu \nu, \lambda \sigma}$$



第5図：Jupyter 上での DSL 実行の様子。

DSL の記法で記述されたプログラムと入力データから、補助情報を提供しつつプログラムの最適化を行う。

システムの実行は次の 3 ステップで行う。

(1) DSL の記述と実行 (セル In[1], セル In[2])

プログラムの開発者は DSL をライブラリとして読み込み、DSL で定められた記法でプログラムを書く。この時は MPI などの最適化は考えずに計算の本質のみを記述する。このセルの実行時点では、deep embedding の手法に基づき、記述した計算式を示す抽象構文木 (Abstract Syntax Tree: AST) が生成される (セル In[1])。次に入力データの読み込みを行い (セル In[2])、暗黙的データが構築される。このステップにより、コンパイラは AST と入力データを組み合わせて計算量などの補助情報を生成できることになる。ここでは、一般的なプロファイリングと異なり、予測値が求められる。

(2) 対話的最適化 (セル In[3], セル In[4])

ユーザは別のセルで、本システムへの補助情報取得命令を実行することができる。これにより計算式や入力データ情報、各種サポート情報を確認する (セル In[3])。例えば、あるループ部分の計算量を取得する操作を行うと、ループ一回毎にどの程度の計算量がかかるかをグラフとして取得することができる。これらを基にユーザは最適化ポリシーを選択する (セル In[4])。このセルの実行時には、最適化ポリシーが AST へのメタデータとして埋め込まれる。このステップを繰り返すことによりユーザが望む最適化ポリシーを含んだ AST が構築される。

(3) コード出力と計算の実行

対話的な最適化が終了した後、別のセルで特別な命令を実行することで AST を元に C++ の最適化科学技術計算コードが生成される（セル In[100]）。

ここでは細かい実装方法の記述は避けるが、超並列計算に向けた静的負荷分散に関連する事項を説明する。簡単のため、MPI を 4 並列で行うケースについて考える。本 DSL では、計算式の構築と入力データの読み込みの時点でコンパイラに計算量を計算させることができるため、MPI 並列計算時のそれぞれのプロセスの計算量も計算することができる。それを活用して静的に負荷分散を行う例を以下に示す。



第 6 図：MPI 並列を単純に導入したケースと静的負荷分散を行ったケース。

左側は、あるループ部分に MPI を単純に導入した場合における各プロセスの計算量を示す。

右側は、同じループ部分に MPI を導入して静的に負荷分散をおこなった場合の計算量を示す。

ユーザは DSL が提供するメソッドを用いて特定のループ部分に MPI のディレクティブを挿入することができる。また、該当ループのプロセス毎の計算量を可視化することができ、このケースでは棒グラフが表示され、色分けされた四本の棒が 4 並列のプロセスに対応する。色分けされた部分の大きさが各プロセスが持つループ一回分（タスク）の計算量に相当する。この補助情報により、単純に MPI を導入した場合（左側）ではプロセス 0 と 1 のタスクの総計算量が小さく、プロセス 2 と 3 のタスクの総計算量大きいことが視覚的に理解できる。このまま実行するとプロセス 0 と 1 の待機時間が大きくなることが想定されるため、ユーザはプロセス 2 と 3 から一部のタスクをプロセス 0 と 1 に移動させる命令を行う。その結果が第 6 図右側であり、このグラフにより計算量の平均化が図られていることが確認できる。

以上が対話的最適化の一連の流れである。補助情報は可視化だけでなく数値としても取得できるため、Ruby スクリプトによって、独自のスケジューラを構築することができる。例えば、以下のようなスケジューラの構築ができる。

```
def scheduler_MPI(func)
  tasks = func.tasks.sortBy(0, UPPER)
  func.parallel(MPI, tasks)
end

def scheduler_complex_parallel(func)
  func1, func2 = func.divide(func.arg[0].lengthOf("child_array") > 1000)
  func.sequential(func1, func2)
  inter_func1 = function("inter_calc").at("sigma").at(func1)
  inter_func1.parallel(MPI)
  func2.parallel(MPI)
end
```

第 7 図：Ruby スクリプトによるスケジューラの例。

関数の引数に、AST が構築された計算式を入れることで、破壊的にメタデータを注入する。

左の関数は AST が構築された計算式を引数にとり、1 ループの計算量毎に昇順に並べ替えて MPI 並列化をすることで、計算量の平衡化を図るものである。右の関数は第 3 図で述べたようなルー

プの分割を行い、別のループに MPI のディレクティブを挿入するスケジューラである。

4. おわりに

本研究では、従来の DSL を参考にして、科学技術計算の超並列化を含めたプログラム最適化のための DSL を設計し、Jupyter と共に用いることで対話的に最適化をするシステムを提案した。科学技術計算例として量子化学計算を取り上げ、入力データの暗黙性や巨大さ、計算式の多重ループ性により生じる最適化の難しさが存在することを示した。その難しさを本 DSL は吸収し、可視化を含めたコンパイラによる補助機能を活用することで、人間とコンパイラが相補的に最適化に取り組むことができる。今後の課題の一つとして、DSL にどこまでの機能を持ち込むかの検討がある。例えば本 DSL のスケジューラは、Ruby の記法に則って実装できることを示したが、DSL にその機能を取り込むことも一つの手である。また、生成される AST をユーザが独自に操作して好きな最適化を組み込む機能も考えられる。本 DSL の開発者が実装した最適化のみではなく、ユーザにより最適化可能性を広げられるような検討を行いたい。

5. 謝辞

本研究は東京大学情報基盤センター若手・女性利用者推薦制度の助成を受けて実施された。

参 考 文 献

- Arie van Deursen, Paul Klint, and Joost Visser. 2000. Domain-specific languages: an annotated bibliography. *SIGPLAN Notices*. 35, 6 (June 2000), 26-36.
- Jeremy Gibbons and Nicolas Wu. 2014. Folding domain-specific languages: deep and shallow embeddings (functional Pearl). *SIGPLAN Notices*. 49, 9 (September 2014), 339-347.
- Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *SIGPLAN Notices*. 48, 6 (June 2013), 519-530.
- Kazuo Kitaura, Eiji Ikeo, Toshio Asada, Tatsuya Nakano and Masami Uebayasi. 1999. Fragment molecular orbital method: an approximate computational method for large molecules. *Chemical Physics Letters*. 313, 3-4 (1999), 701-706.
- A. Szabo and N.S. Ostlund 著, 大野公男, 阪井健男, 望月 祐志翻訳『新しい量子化学 上 電子構造の理論入門』303 p, 東京大学出版会, 1987

1 はじめに

近年、タンパク質-タンパク質相互作用において天然変性タンパク質 (天然変性領域) の果たす役割に大きな注目が集まっている。本研究では、細胞分裂時の有糸分裂期において、インナーキネトコアとアウターキネトコアの結合に重要な役割を果たす Ndc80 複合体と Centromere Protein T (CENP-T) の天然変性領域の結合様式を分子動力学シミュレーション (MD シミュレーション) と自由エネルギー計算を用いて明らかにすることを目的として、理論的な研究を行った。(図 1 を参照)

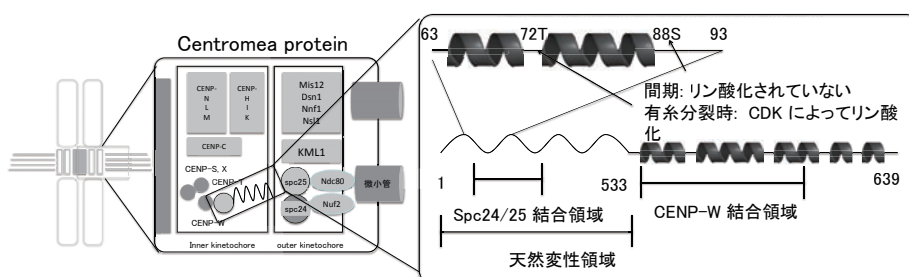


図 1 有糸分裂期におけるインナーキネトコアとアウターキネトコアの結合に関わるタンパク質の模式図と CENP-T の天然変性領域および結合領域。

有糸分裂期には、アウターキネトコアを構成する Ndc80 複合体が、微小管との複合体形成に重要な役割を果たすことが分かっている。一方で、Ndc80 複合体 (アウターキネトコア) とインナーキネトコアを構成するタンパク質の相互作用については、その結合様式の詳細が不明であった。近年、西野らの研究 [4] によって、インナーキネトコア中の CENP-T の天然変性領域の一部と Ndc80 複合体を構成する Spc24/25 複合体が相互作用すること、また、Spc24/25 複合体と CENP-T の結合様式については、CENP-T への翻訳後修飾が重要な役目を果たすることが示唆された。具体的には、CENP-T のリン酸化された 72 残基が 74 残基との間に塩橋を形成して、Spc24/25 結合領域の二つのヘリックスの相対位置を制御し、Spc24/25 複合体と安定的に結合するというというメカニズムが提出された。(図 2 に変異体における T72D と R74 の間の塩橋を示した。)

本研究においては、この提示されたメカニズムの妥当性の検証および原子解像度での精緻化のため、CENP-T の天然変性領域の構造安定性を MD シミュレーションを用いて様々な条件下で計算した。即ち、CENP-T リン酸化類似体 (T72D,S88D) と Spc24/25 複合体の結晶構造 (PDB ID: 3VZA) をもとに、野生型およびリン酸化されたもの (pT72,pS88) をモデリングし、Spc24/25 と CENP-T の複合体および CENP-T 単体の 100 ns 規模の全原子 MD シミュレーションを行った。その結果、(T72D,S88D) および (pT72,pS88) において 72-74 間の塩橋は、複合体においても CENP-T 単体でも安定的に維持され、実験によって提出されたメカニズムの妥当性が示唆された。さらに、Alchemical Free Energy 法を用いて状態間の安定性の差を見積って、塩橋の存在がヘリックス間の相対位置の安定性に果たす影響についても定量的に議論する道を探った。

2 方法

本研究においては、MD シミュレーションと Alchemical Free Energy 法 (AFE) を使ってリン酸化がおよぼす CENP-T の塩橋形成や塩橋による配向制御などへの影響および Spc24/25 複合体との結合能の変化

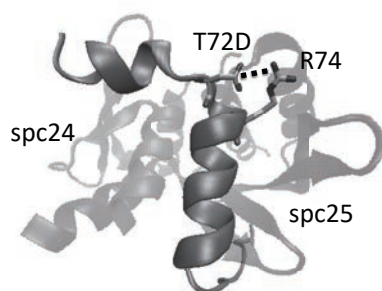


図2 CENP-T 変異体 (63-93 残基) -Spc24/25 複合体の結晶構造を図示 (PDB ID: 3VZA)。CENP-T のリン酸化を受ける残基 R74 は、直接は、Spc24/25 複合体との結合に関与していないが、結合領域の二つのヘリックスの相対的位置を制御し結合の安定化に関与すると考えられる。72-74 残基間の塩橋を点線で示した。

について検証を行なった。AFE は、結合自由エネルギー差を計算するための厳密法で、仮想的な中間状態を複数設定して、Bennet Acceptance Ratio 法 (BAR)、自由エネルギー摂動法 (FEP)、積分法 (TI) などと組み合わせてタンパク質-リガンド結合自由エネルギー差を計算する手法である。本稿の研究では AFE を変異体間でのタンパク質-タンパク質結合に応用した。

CENP-T リン酸化類似体 (T72D,S88D) と Spc24/25 複合体の結晶構造 (PDB ID: 3VZA) をもとに、野生型およびリン酸化体 (pT72,pS88) をモデリングし、初期構造を用意した。水モデルには TIP3P を、タンパク質およびイオンの力場には AMBER99SB を採用した。リン酸化残基のモデルには Homeyer らのモデル [3] を用いた。全ての系において、NaCl イオンを用いて系全体の電荷を中和した。それぞれの変異体について、CENP-T 単体の系および CENP-T-Spc24/25 複合体の系を用意した。(表 1) MD シミュレーションは、GROMACS-2018 [1, 5] を用いて、300K・1bar の定温・定圧アンサンブルで行った。熱浴および圧力浴には、それぞれ Langevin 熱浴と Berendsen 浴を用いた。溶媒をふくんだ総分子数は、CENP-T 単体の系で約 30,000、複合体の系で約 40,000 とした。

表1 計算の対象となる系の組成。順に CENP-T リン酸化類似体 (3VZA)、野生型 (WT)、リン酸化体 (phosho) で、上段は CENP-T 単体、下段は CENP-T-Spc24/25 複合体である。

3VZA(T72D,S88S)			WT		phosho(72pT,88pS)	
	分子数	原子数	分子数	原子数	分子数	原子数
タンパク質	1	499	1	500	1	507
水	9655	28965	9636	28908	9758	29274
Na ⁺	1	1	0	0	2	2
Cl ⁻	0	0	1	1	0	0
3VZA(T72D,S88S)			WT		phosho(72pT,88pS)	
	分子数	原子数	分子数	原子数	分子数	原子数
タンパク質	1	3159	1	3160	1	3167
水	12606	37818	12606	37818	12586	37758
Na ⁺	3	3	1	1	4	4
Cl ⁻	0	0	0	0	0	0

MD シミュレーションは、各々平衡化計算後に 100 ns の計算を行い解析に使用した。

AFE の解析計算には、Alchemical-analysis のパッケージ [2] を用いた。Alchemical-analysis は、MD のポストプロセスとして、自由エネルギー計算を行うための応用性の高いフレームワークである。厳密な自由エネルギーの計算方法として、Bennet Acceptance Ratio 法、自由エネルギー摂動法、積分法などを実行することができる。

3 結果

3.1 リン酸化による塩橋形成や塩橋による配向制御

6 つの系 (表 1) について行なった MD シミュレーションの結果を示す (図 3)。図 3(a) は、各々の計算の平衡化後の CENP-T (60-93) の平均二乗変異 (RMSD) を、(b) は、CENP-T (60-93) のヘリックス構造の存在確率を、(c) は、変異型/リン酸化体について、72 残基が 74 残基との間に塩橋の時間変化を示す。

CENP-T 変異型/リン酸化体という Spc24/25 複合体との結合能が実験において高いと示されたものは、ともに、塩橋を維持しているという結果を示した。

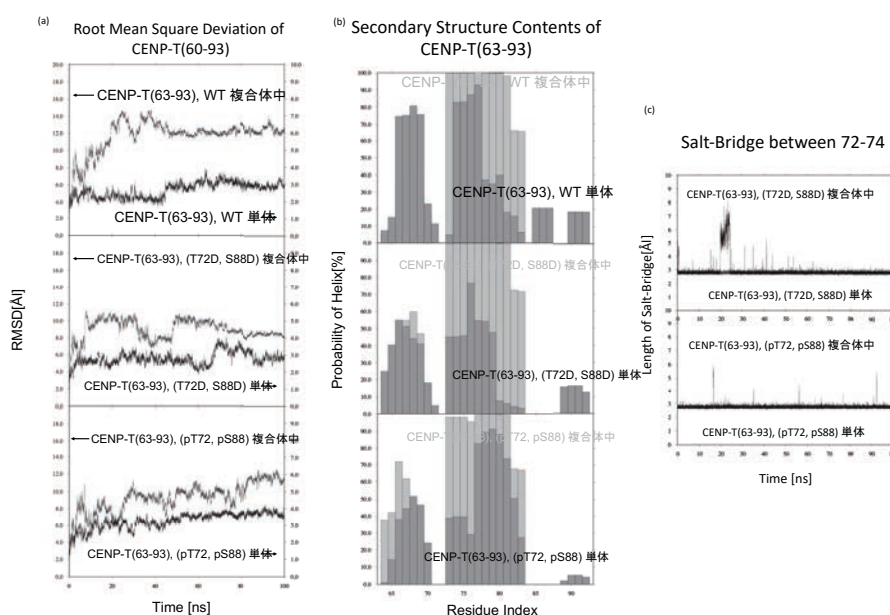


図3 (a) 3vza/野生型/リン酸化体の平衡化後の CENP-T (60-93) の平均二乗変異 (RMSD)、(b) 3vza/野生型/リン酸化体の CENP-T (60-93) のヘリックス構造の存在確率、(c) 変異型/リン酸化体について、72 残基が 74 残基との間に塩橋の時間変化。

3.2 変異/リン酸化による結合自由エネルギー差

MD シミュレーションの結果をもとに、AFE 法によって、変異およびリン酸化による結合結合自由エネルギーへの影響を調べた。この際に使用した熱力学的サイクルを図 4(b) に示す。

BAR 法を用いた計算による野生型からリン酸化類似体 (T72D,S88D) および野生型からリン酸化体 (pT72,pS88) への結合自由エネルギー差をみると (表 2) 変異やリン酸化によって、結合は安定化されることが示唆された。

このことから、実験によって提出されたメカニズムの妥当性が示唆された。

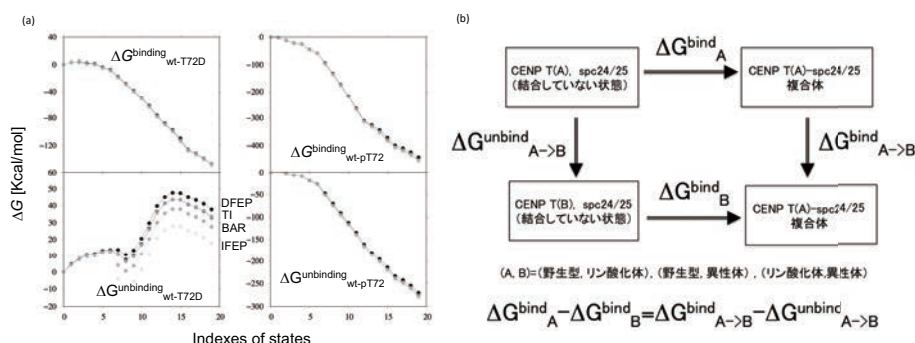


図4 (a) 各状態間の自由エネルギー変化、(b) 熱力学的サイクル。

表2 変異/リン酸化による結合自由エネルギー差。BAR 法による計算。(kcal/mol)

	$\Delta G^{\text{binding}}_{\text{mut.}}$	$\Delta G^{\text{unbinding}}_{\text{mut.}}$	$\Delta G^{\text{binding}}_{\text{mut.}} - \Delta G^{\text{binding}}_{\text{wt}}$
WT $\rightarrow$ T72D	-148.1	27.6	-175.7
WT $\rightarrow$ pT72	-454.9	-277.4	-177.5

4 結論

本稿の研究では、有糸分裂期におけるタンパク質の集合という生物学的に重要なテーマに対して、最近の実験的な知見をもとに、MD シミュレーション及びそれにもとづく自由エネルギー計算を行い、実験結果から示唆されたメカニズムの検証を行なうことをめざした。

本研究では、特にアウターキネトコア中の Ndc80 複合体を構成する Spc24/25 複合体とインナーキネトコア中の CENP-T の結合について、CENP-T の翻訳後修飾が大きな役割を果たす、即ち CENP-T のリン酸化された 72 残基が 74 残基との間に塩橋を形成し天然変性領域の二つのヘリックスの相対的位置を制御することで、Spc24/25 複合体と安定的に結合するというメカニズムの妥当性の検証および原子解像度での精緻化のため、CENP-T の天然変性領域の構造安定性を様々な条件下で計算した。リン酸化類似体 (T72D,S88D) の結晶構造をもとに、野生型およびリン酸化されたもの (pT72,pS88) をモデリングし、Spc24/25 と CENP-T の複合体および CENP-T 単体の 100 ns の全原子 MD シミュレーションを行った。

その結果、(T72D,S88D) および (pT72,pS88) において 72-74 間の塩橋は、複合体においても CENP-T 単体でも安定的に維持され、実験によって提出されたメカニズムの妥当性が示唆された。また、Alchemical Free Energy 法を塩橋の存在が、ヘリックス間の相対位置安定性に及ぼす影響についても定量的に議論する道も探った。

今後は、より詳細かつ精度の高い知見を得るために、より大規模な計算および著者らの研究にもとづく実験データの参照によるシミュレーションデータの精度向上手法 [6] の適用などを検討している。

本研究は、東京大学情報基盤センター 2019 年度前期・後期若手・女性推進制度を通して使用したスーパーコンピュータ Reedbush-U (JCAHPC) で実地した成果を含む。

参考文献

- [1] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. *J. Chem. Theory Comput.*, Vol. 4, p. 435, 2008.
- [2] P. V. Klimovich and D. L. Mobley. *J. Comput. Aided Mol. Des.*, Vol. 29, p. 1007, 2015.
- [3] H. Nadine, A. H. C.Horn, L. Harald, and S. Heinrich. *J. Mol. Modeling*, Vol. 12, p. 281, 2006.
- [4] Tatsuya Nishino, Florencia Rago, Tetsuya Hori, Kentaro Tomii, Iain M Cheeseman, and Tatsuo Fukagawa. *The EMBO Journal*, Vol. 32, No. 3, pp. 424–436, 2013.
- [5] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl. *Bioinformatics*, Vol. 29, p. 845, 2013.
- [6] Y. Yamamori and K. Tomii. *Chem. Phys. Lett.*, Vol. 779, p. 138821, 2021.

マルチスケール分子シミュレーションによる

受容体チロシンキナーゼの構造モデリング

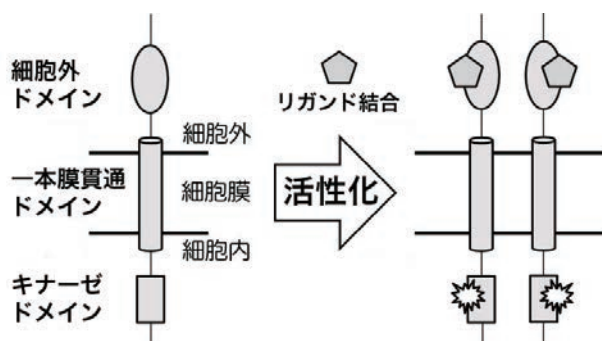
森 義 治

神戸大学大学院システム情報学研究科

1. はじめに

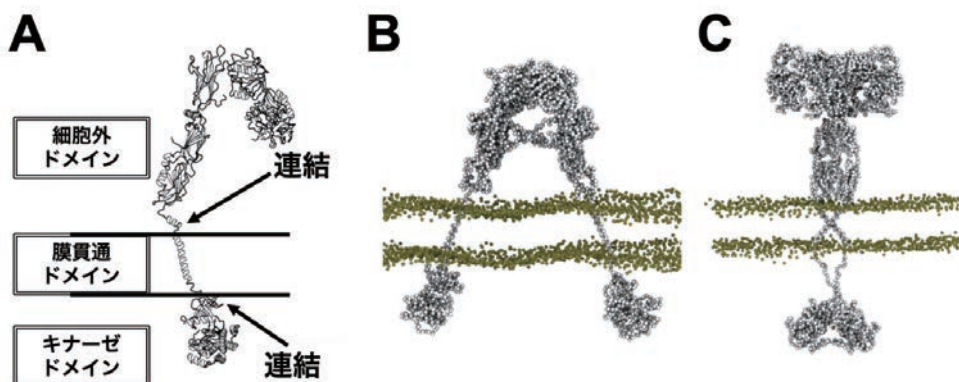
受容体チロシンキナーゼは細胞内領域のリン酸化を介して細胞内外のシグナル伝達を担っており、ヒトをはじめとする生物の細胞において重要な役割を果たしている。受容体チロシンキナーゼの例として上皮成長因子受容体やインスリン受容体などがあり、細胞増殖や代謝過程の調節を行っている。これらのタンパク質の機能が失われたり過剰に発現したりした場合には、がんや代謝の疾患などの原因となると考えられている。原子レベルにおける構造的観点からみれば受容体チロシンキナーゼは膜タンパク質であり、第1図に示したような一般的な構造を持っている。細胞外にはシグナルを受け取るための領域が存在し、細胞内にはチロシンキナーゼドメイン（タンパク質のリン酸化を担う領域）を持つ。これらの構成部分が膜貫通部分でつながれており、この膜貫通部分は一本の α ヘリックス構造からなる。シグナル分子が細胞外の領域に結合することにより受容体が活性化され、細胞内のチロシンキナーゼドメインが機能を発現するようになる。またシグナル分子によって活性化されたチロシンキナーゼは多くの場合に二量体で機能していると考えられているが、多量体構造をとる場合もあると考えられている[1]。

受容体チロシンキナーゼは典型的な膜貫通タンパク質であるが、上記のように重要な機能を担っているにもかかわらず、その原子レベルでの全体構造はX線・NMR・電子顕微鏡のような実験によってほとんど明らかになっていない[2]。これは構造的特徴として述べたように、ただ一本からなる膜貫通領域の非常に柔軟な構造に起因していると考えられる。細胞外からのシグナルを細胞内部に伝える機構を解明するためには、このようなタンパク質全長の構造を明らかにすることが不可欠である。



第1図：受容体チロシンキナーゼの概念図。

受容体チロシンキナーゼは細胞外ドメイン、膜貫通ドメインおよびキナーゼドメインから構成される。リガンド分子が結合することにより一般的に二量体が形成される。



第2図：インスリン受容体の構造モデリング。

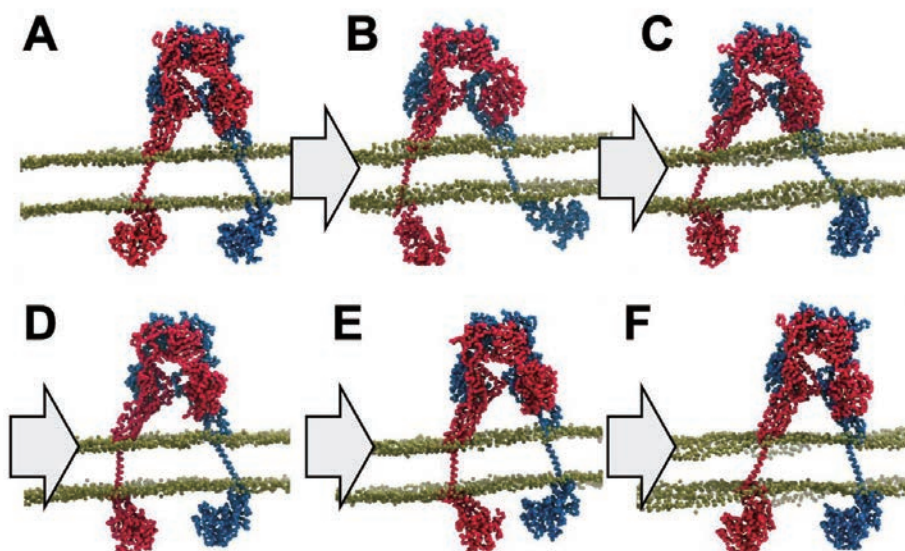
A：それぞれのドメインを連結することによりインスリン受容体の全長構造が作成される。B：リガンド非結合型におけるインスリン受容体の構造モデル。C：活性状態におけるインスリン受容体の構造モデル。

以上のように受容体チロシンキナーゼの機能の全容を解明するためには、シグナル分子の結合による活性化とそれに伴う細胞膜を介した情報の伝達の機構を理解する必要がある。このようなことを実現するために、受容体チロシンキナーゼの全長構造をモデリングし、全体構造における分子動態を明らかにすることを目的とした。詳細は次節において述べるが、概要は以下の通りである。実験的に細胞外ドメイン・膜貫通ドメイン・キナーゼドメインのそれぞれの構造は決定されている場合もあり、これらを鋳型としてお互いを連結したモデルを作成した（第2図）。このように得られた受容体チロシンキナーゼの構造モデルを次の方法で精密化した。まず粗視化分子シミュレーションを利用して全体的な構造のサンプリングを実行した。さらに、このように得られた様々な粗視化構造の妥当性を全原子分子シミュレーションを実行することにより理解した。

2. 受容体チロシンキナーゼの構造モデリングと分子動力学シミュレーション

対象とする系である受容体チロシンキナーゼは膜貫通領域に柔軟な部分を持っており、実験においてもいまだに全体構造の原子レベルでの十分な構造決定ができていない。また標準的な全原子分子動力学シミュレーションでは、そのような部分における構造揺らぎを十分にサンプリングすることは計算量が非常に多くなるため困難であると考えられる。そこで本研究ではまず実験的に決定されている領域を連結することにより粗視化された系を構築し、受容体チロシンキナーゼ・細胞膜・水溶媒の粗視化分子動力学シミュレーションを実行した。この方法により全原子モデルと比較して長い時間スケールを追いかけることができ、大規模な構造変化をとらえることが可能となる。

本研究においては受容体チロシンキナーゼの一種であるインスリン受容体を用いた[3]。インスリン受容体は生体内の代謝に関わっており、ホルモンであるインスリンを結合し血糖値のコントロールに関わるタンパク質のひとつである。インスリン受容体の構造的特徴として、リガンド（インスリン）非結合の状態においてジスルフィド結合を介してすでに二量体を形成していることが挙げられる。その他の受容体チロシンキナーゼは一般的にリガンドが結合することにより多



第3図：シミュレーションにおけるインスリン受容体のスナップショット。

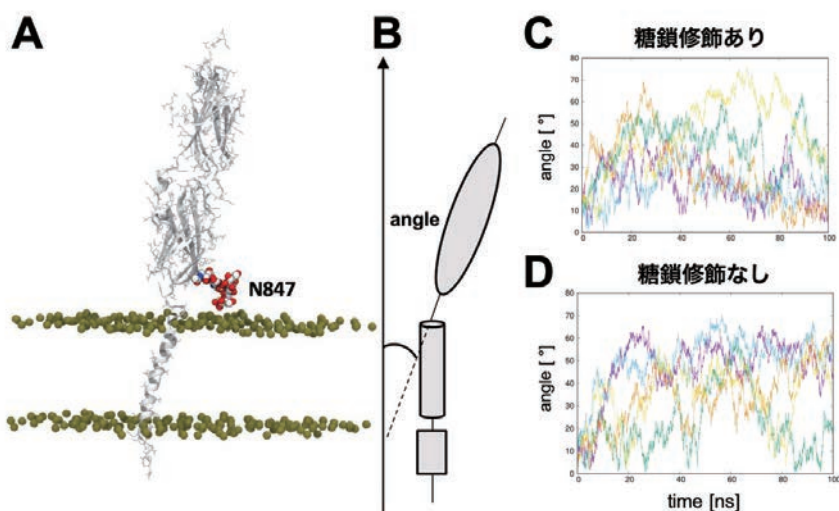
A-F：リガンド非結合型インスリン受容体の時間発展を示している。A から F にかけて時間が経過している。

量体化すると考えられている。現在までにインスリン受容体のリガンド非結合状態および活性状態（リガンド結合状態）における細胞外ドメイン、膜貫通ドメイン、ならびに細胞内ドメインのそれぞれの構造が X 線結晶構造解析等により決定されている。しかしながら全長の構造は決定されていないため、リガンド非結合状態および活性状態のインスリン受容体の全長構造モデリングを行った。これまでに得られている構造を鋳型として Modeller ソフトウェアによりホモロジーモデリングを実行することにより全長の構造を作成し（第2図 A）、分子動力学シミュレーションの初期構造とした。リガンド非結合状態におけるインスリン受容体の全体構造を第2図 B に、活性状態におけるインスリン受容体の全体構造を第2図 C に示す。

作成した初期構造を基にして粗視化分子動力学シミュレーションを実行することにより、全体構造の構造緩和が生じるようにし、より適切な全体構造を得た。粗視化シミュレーションは次のような条件で行った。粗視化モデルとしてタンパク質・脂質・水分子・イオンすべてを MARTINI 力場(ver. 2.2) で取り扱った。またタンパク質の三次構造が鮮明な領域においては Elastic Network Model[4]を使用することにより分子構造が壊れるような大規模な構造変化が生じないようにした。リガンド非結合状態・活性状態ともに系の全粒子数はおよそ 20 万となった。計算は Reedbush-H スーパーコンピュータシステムにより実行し、それぞれの系において $3\mu\text{s}$ の粗視化分子動力学シミュレーションを 5 回実行した。分子動力学シミュレーションは GROMACS (ver. 2018.8) により実行された。

3. 結果・議論

粗視化シミュレーションの結果、受容体チロシンキナーゼにおける様々な構造をサンプルすることができた（第3図 A-F）。まずリガンド非結合状態の構造モデルにおいては細胞内のキナーゼドメインが膜に接触するような構造変化が見られた（第3図 A）。また活性状態の構造モデル



第4図：インスリン受容体細胞外ドメインにおける糖鎖の効果。

A：糖鎖が結合したインスリン受容体部分の全原子シミュレーションモデル。B：細胞外ドメインの傾きの定義。
C, D：糖鎖修飾がある系(C)およびない系(D)における細胞外ドメインの傾きの時間発展。

においてはキナーゼドメインが二量体化していることが重要であるため、適切な構造を保ったまま脂質と結合していることが分かった。膜貫通領域は、特にリガンド非結合状態の構造が大きく変化した。膜貫通領域の初期構造は細胞外ドメインの構造から予測されるように作成したため平衡構造から大きく離れている可能性が高かったが、実際に粗視化シミュレーションを実行することで二量体の膜貫通領域同士がより近くなるような位置に配置されることが分かった(第3図E, F)。以上の二つの領域の大きな構造変化にもかかわらず、細胞外ドメインは多くの場合実験で得られた構造に近い構造が保持されることが分かり、全体として安定なシミュレーションが実行できていることを確認することができた。

以上のような結果とともに、リガンド非結合状態・活性化状態のどちらの状態においてもいくつかの構造不安定性も見られた。特に目立った構造変化は細胞外ドメインが直立した構造を維持できないために生じているようであった。つまり、いくつかの計算においては細胞外ドメインが脂質膜に接触するような構造変化が生じた。このような構造変化は細胞外ドメインが直立した構造で安定的に得られていることと不整合である。このように受容体チロシンキナーゼ全長の構造はいくらか不安定であることが分かった。この原因として、通常受容体チロシンキナーゼの細胞外ドメインに付加されている糖鎖の影響を無視していたことが考えられた[5]。この糖鎖付加による構造安定性を明らかにするために、糖鎖を付加した細胞外ドメインの一部と膜貫通ドメインおよび脂質二重膜からなる系における全原子分子動力学シミュレーションを実行した。全原子分子動力学シミュレーションは次のような条件で行った。インスリン受容体部分構造においてアミノ酸残基 N847 に生体中に見られる最低限の糖鎖修飾をした系を用意した(第4図A)。また比較のために糖鎖修飾を行っていない同様の系も用意した。全原子数はそれぞれの系でおよそ 23 万であった。力場として CHARMM36m を使用し、100 ns の分子動力学シミュレーションをそれぞれの系で 5 回実行した。

以上の計算から糖鎖修飾されたインスリン受容体の細胞外ドメインは直立した構造を安定的

に取ることが分かった（第 4 図 B-D）。糖鎖は細胞膜と相互作用し、タンパク質が傾いた場合にそれを防ぐような働きをしていると考えられる。一方で糖鎖修飾がない細胞外ドメインは粗視化シミュレーションで見られたものと同様な構造変化が見られた。つまり、細胞外ドメインが倒れこみ、細胞膜の脂質分子とタンパク質の全領域が接触するような構造を見ることができた。

以上述べてきたように、インスリン受容体を用いた受容体チロシンキナーゼにおける構造モデリングおよびその構造に基づく粗視化分子動力学シミュレーション、そしてこれらの考察を基にした全原子分子動力学シミュレーションから構造変化の詳細に係る知見を得ることができた。本研究により行われたような方法を用いることにより、受容体の構造変化を詳細に理解できるようになり、ひいては細胞内外を介したシグナル伝達機構を明らかにすることができるようになっていくことが期待される。

謝 辞

本研究は 2019 年度東京大学情報基盤センター「若手・女性利用者推薦」課題「マルチスケール分子シミュレーションによる受容体チロシンキナーゼの構造モデリング」の支援により実施されました。

参 考 文 献

- [1] R. Trenker and N. Jura, *Curr. Opin. Cell Biol.*, 63, 174-185, 2020.
- [2] T. Gutmann et al., *J. Cell Biol.*, 217, 1643-1649, 2018.
- [3] G. Scapin et al., *Nature*, 556, 122-125, 2018.
- [4] X. Periole et al., *J. Chem. Theory Comput.*, 5, 2531-2543, 2009.
- [5] G. Enkavi et al., *Chem. Rev.*, 119, 5607-5774, 2019.

機械学習による時系列データの学習過程の解明

中井拳吾

東京海洋大学 学術研究院

1. はじめに

機械学習は物理学の様々な分野で注目されている。近年、機械学習の一種で、再帰的ニューラルネットワーク構造をもったリザーバーコンピューティング [1, 5] が時系列やリャプノフ指数などの予測において有効であることが報告されている。我々はこの機械学習手法を流体のエネルギー変数に適用し、時間発展の予想等を成功させている [2, 3]。

一方で、単に時系列を予測できるだけではその現象のモデルを構成したと言うには不十分である。そこで、本研究では学習量(データの量)などに注目しつつ学習により得た時間発展モデルがもとの力学系構造をどの程度再現できるかを明らかにする。

本稿は小林 幹氏 (立正大学), 斉木 吉隆氏, 堤 夏輝氏 (一橋大学) との共同研究に基づくものである。

2. 機械学習手法

まずリザーバーコンピューティングによる学習方法について説明する。 $\frac{d\phi}{dt} = \mathbf{f}(\phi)$ で表される力学系の変数, $\mathbf{u} = \mathbf{h}_1(\phi) \in \mathbb{R}^M$ と $\mathbf{s} = \mathbf{h}_2(\phi) \in \mathbb{R}^P$ について考える。ただし、時系列データ $\mathbf{u}$ は $\mathbf{s}$ の時系列を予測する時刻にないとする。リザーバーベクトル $\mathbf{r} \in \mathbb{R}^N$ ($N \gg M$) は次の式で支配されている。

$$\mathbf{r}(t + \Delta t) = (1 - \alpha)\mathbf{r}(t) + \alpha \tanh(\mathbf{A}\mathbf{r}(t) + \mathbf{W}_{\text{in}}\mathbf{u}(t)) \quad (1)$$

ただし、 Δt は時間幅を表す。また、行列 $\mathbf{A}$, $\mathbf{W}_{\text{in}}$ は $N \times N$, $N \times M$ 行列である。パラメータ α ($0 < \alpha \leq 1$) は $\mathbf{r}$ の力学系の非線形性を表し、式(1)の Δt に依存して決める。リザーバーベクトルは $[0, 1]$ の一様乱数から定めた $\mathbf{r}(-\tau)$ を初期値とする。ただし、 $\tau/\Delta t (\gg 1)$ はトランジェント時間に相当する。

行列 $\mathbf{W}_{\text{in}}$ の各列はただ一つのみに $[-\sigma, \sigma]$ の一様乱数から値を与え、それ以外は 0 とする。行列 $\mathbf{A}$ は $D \times N$ 個の非 0 の要素を持つ疎行列とする。 $D \times N$ 個の非 0 成分は $[-1, 1]$ の一様乱数から値を与える。さらに、行列 $\mathbf{A}$ については最大固有値が ρ になるように行列のすべての要素を調整する。

リザーバーベクトル $\mathbf{r}$ は次のようにして得たいベクトルに変換する。

$$\hat{\mathbf{s}}(t) = \mathbf{W}_{\text{out}} \mathbf{r}(t) + \mathbf{c}.$$

これにより $L = T/\Delta t$ ステップのリザーバーベクトル $\{\mathbf{r}(l\Delta t)\}_{l=1}^L$ を得る。

$0 < t \leq T$ (学習時間と呼ぶ) での時系列 $\hat{\mathbf{s}}$ が既知の時系列 $\mathbf{s}$ の近似になるように $\mathbf{W}_{\text{out}}$ と $\mathbf{c}$ を決定する。この決定がリザーバーコンピューティングの学習に相当する。これは次の二次形式が最小化するように $\mathbf{W}_{\text{out}}$ と $\mathbf{c}$ を決定することに対応する。

$$\sum_{l=1}^L \|(\mathbf{W}_{\text{out}} \mathbf{r}(l\Delta t) + \mathbf{c}) - \mathbf{s}(l\Delta t)\|^2 + \beta [\text{Tr}(\mathbf{W}_{\text{out}} \mathbf{W}_{\text{out}}^T)] \quad (2)$$

ただし $\mathbf{q}$ に対して $\|\mathbf{q}\|^2 = \mathbf{q}^T \mathbf{q}$ とし, $\beta \geq 0$ を含む項は $\mathbf{W}_{\text{out}}$ の過剰適合を避けるために導入している. この決定が妥当であれば, $t > T$ (予測時間と呼ぶ) での $\hat{\mathbf{s}}(t)$ は $\mathbf{s}(t)$ を再現できるはずである. 二次形式 (式 (2)) を最小化する解を $\mathbf{W}_{\text{out}}^*$ と $\mathbf{c}^*$ とすると予測時間では

$$\hat{\mathbf{s}}(t) = \mathbf{W}_{\text{out}}^* \mathbf{r}(t) + \mathbf{c}^*$$

として時系列 $\hat{\mathbf{s}}$ を得る. $\mathbf{W}_{\text{out}}^*$ と $\mathbf{c}^*$ は次のように書けることが知られている [参考文献 1].

$$\mathbf{W}_{\text{out}}^* = \delta \mathbf{S} \delta \mathbf{R}^T (\delta \mathbf{R} \delta \mathbf{R}^T + \beta \mathbf{I})^{-1}$$

$$\mathbf{c}^* = -[\mathbf{W}_{\text{out}}^* \bar{\mathbf{r}} - \bar{\mathbf{s}}]$$

ただし, $\bar{\mathbf{r}} = \sum_{l=1}^L \mathbf{r}(l\Delta t) / L$, $\bar{\mathbf{s}} = \sum_{l=1}^L \mathbf{s}(l\Delta t) / L$ とし, $\mathbf{I}$ は $N \times N$ の単位行列, $\delta \mathbf{R}$ は l 行が $\mathbf{r}(l\Delta t) - \bar{\mathbf{r}}$ となるような行列とする. 同様に, $\delta \mathbf{S}$ は l 行が $\mathbf{s}(l\Delta t) - \bar{\mathbf{s}}$ となるような行列とする.

ただし, 予測時間 $t > T$ での時系列データ $\mathbf{u}$ は利用できないため, $\mathbf{u}(t)$ として式 (1) で得た $\hat{\mathbf{s}}(t)$ を用いる:

$$\mathbf{r}(t + \Delta t) = (1 - \alpha) \mathbf{r}(t) + \alpha \tanh(\mathbf{A} \mathbf{r}(t) + \mathbf{W}_{\text{in}} \hat{\mathbf{s}}(t)).$$

3. 設定と結果

ローレンツ方程式 $\frac{dx}{dt} = -10x + 10y$, $\frac{dy}{dt} = -xz + rx - y$, $\frac{dz}{dt} = xy - 8z/3$ について考える.

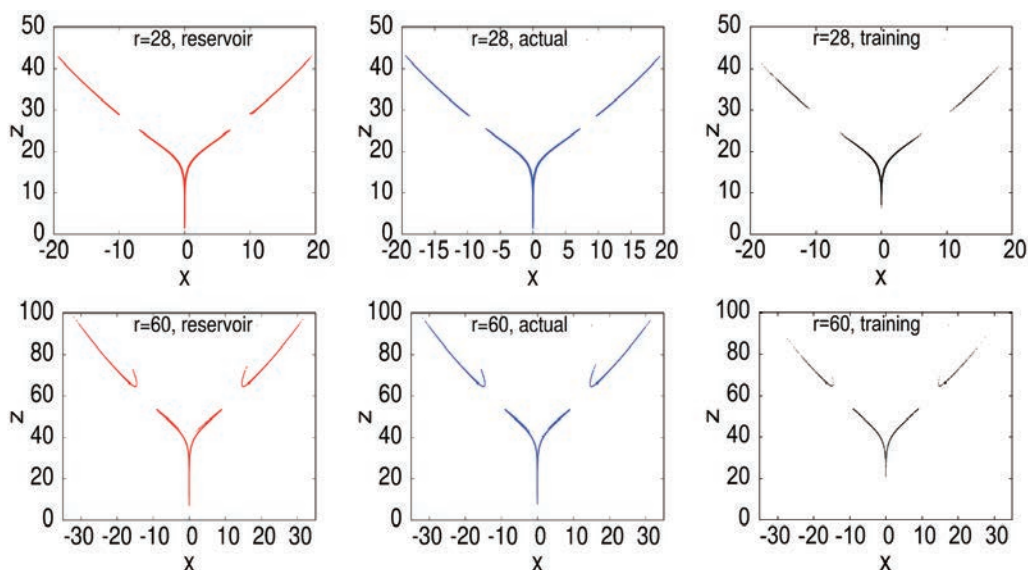
この方程式系はカオス性がある. 本研究ではパラメータ r として 28 (もしくは 60) として得られる時系列を扱う. $r = 60$ 付近では力学系は非双曲的であり, パラメータの微小な変化により力学系の構造が大きく変化する一方で, $r = 28$ 付近では力学系は双曲的で構造安定であり, そのような構造変化は起こらないことが知られている. ローレンツ方程式の時間発展により得られた時系列データを $\{\mathbf{u}, \mathbf{s}\}$ としてリザーバーコンピューティングにより学習させて得られた時間発展モデルが, 元の力学系構造をどの程度再現するかを解析する. 特に本誌では会軌道の断面や不変密度分布といった不変集合に注目する.

3.1 ポアンカレ断面

学習により得た機械学習モデルのアトラクタの大きさや概形について, ローレンツ方程式のものとの比較をする. 3次元では比較が困難であるため, 2次元平面に落として考察する. 具体的には解軌道が適当な平面を横切るときにプロットして得られるポアンカレ断面というものを用いて比較をする. 学習により得た機械学習モデルから得られた 10^6 時間の時系列データとローレンツ方程式の時間発展により得られた 10^6 時間の時系列データ, 学習に用いた 5000 時間の時系列データのそれぞれに対して, 適当な平面として $x = y$ となる平面にポアンカレ断面を書いた (第 1 図).

まず, 学習に用いた 5000 時間の時系列データにより得たポアンカレ断面はローレンツ方程式の 10^6 時間の時系列データにより得られたポアンカレ断面に比べて小さいことが見て取れる. 他方で, 機械学習モデルから得られたポアンカレ断面はローレンツ方程式の 10^6 時間の時系列

データにより得られたポアンカレ断面と似ていることがわかる。このことから、ローレンツ方程式の 10^6 時間の時系列データにより得られたポアンカレ断面に比べて学習に用いた 5000 時間の時系列データにより得たポアンカレ断面は小さいにもかかわらず、機械学習モデルから得られたポアンカレ断面はローレンツ方程式の断面を十分に再現していることがわかる。



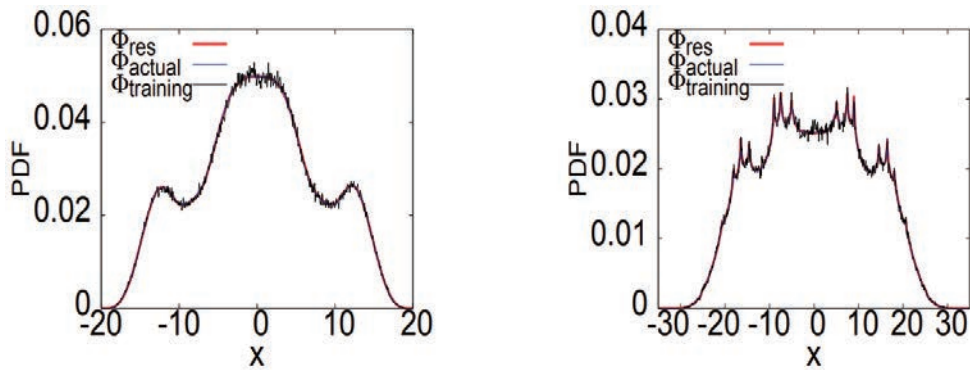
第 1 図：ポアンカレ断面の様子 ($r = 28$ (上段) $r = 60$ (下段))。

機械学習モデルの時間発展によって得られた時系列データと比較のためローレンツ方程式の時間発展によって得られた時系列データ，この機械学習モデルを作るために用いた時系列データのそれぞれに対して， $|x - y| < 0.05$ となったときにプロットした。それぞれの時間長は 10^6 , 10^6 , 5000 である。

3.2 密度分布

機械学習モデルから得られた 10^6 時間の時系列とローレンツ方程式の時間発展により得られた 10^6 時間の時系列，学習に用いた 5000 時間の時系列に対して， x 変数の時系列データの滞在時間密度分布を計算した (第 2 図)。

学習に用いた 5000 時間の時系列データにより得た密度分布はローレンツ方程式の 10^6 時間の時系列データにより得られた密度分布に比べて激しく振動していることが見て取れる。他方で，機械学習モデルから得られた密度分布とローレンツ方程式の 10^6 時間の時系列データにより得られた密度分布の差はかなり小さいことが見て取れる。このことから，学習に用いた時系列データから得られた密度分布は時間が短い，つまりデータ量が少ないために激しく振動しているにもかかわらず，機械学習モデルから得られた密度分布はローレンツ方程式の密度分布を十分に再現していることがわかる。特に， $r = 60$ の密度分布は特異な構造があるが，この構造も機械学習モデルは再現していることが見て取れる。



第2図: ローレンツ方程式の滞在時間分布 ($r = 28$ (左図) $r = 60$ (右図))。

機械学習モデルの時間発展によって得られた x 変数の密度分布 (Φ_{res})と比較のためローレンツ方程式の時間発展によって得られた x 変数の密度分布 (Φ_{actual})を描いた。また、この機械学習モデルを作るために用いたトレーニングデータの x 変数の密度分布 (Φ_{training}) も描いた。

4. まとめ

決定論的な力学系の時系列から機械学習力学系モデルを構築し、それが時系列のどのような構造を再現しうるかを明らかにした。本研究では、一見すると学習には不十分に思えるデータ量の学習データから構築した機械学習モデルであってもポアンカレ断面や密度分布といった不変密度分布が高精度に再現できることが明らかになった。なお本結果は、論文 [4] としてまとめて査読付き国際誌に投稿した。

参考文献

- [1] H. Jaeger, and H. Haas, Science, 304, pp. 78-80, (2004).
- [2] K. Nakai, and Y. Saiki, Physical Review E 98, 023111:1-6, (2018).
- [3] K. Nakai, and Y. Saiki, Discrete and Continuous Dynamical Systems Series S 14, 3, (2021).
- [4] M. Kobayashi, K. Nakai, Y. Saiki, and N. Tsutsumi, Dynamical system analysis of a data-driven model constructed by reservoir computing, Physical Review E 104, 044215:1-7, (2021).
- [5] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, Chaos, 27, pp. 041102. pp. 391-403, (2017).

2021 年 12 月 22 日、BlueJeans を使ったオンライン形式でフラグメント分子軌道(FMO)計算プログラム ABINIT-MP に関するハンズオンセミナーが実施されました。ABINIT-MP は、20 年以上に渡って研究開発が続けられており、現在は Open シリーズとして Ver. 1 Rev 22 (2020 年 6 月リリース)と Ver. 2 Rev. 4 (2021 年 9 月リリース)が併存にて、「富岳」を頂点とする国内 HPCI センター群にライブラリ提供されており、「Wisteria/BDEC-01」でも A64FX 系の Odyssey、Xeon 系の Aquarius の両方に導入されています。実装されている機能によって、両者の実行環境を適宜使い分けていただくためです。

今回のハンズオンセミナーは、(一財)高度情報科学技術研究機構(RIST)の主催で、東京大学情報基盤センター、東京大学生産技術研究所革新的シミュレーション研究センター、「富岳」成果創出加速プログラム課題『スーパーシミュレーションと AI を連携活用した実機クリーンエネルギーシステムのデジタルツインの構築と活用』(東京大学工学部拠点)に共催をいただきました。

可視化インターフェースである BioStation Viewer を用いる関係で、Ver. 1 Rev. 22 の利用で進めました。実施担当者は望月の他、望月研究室の研究員の奥脇弘次氏(図 1 を参照)、それにジョブ投入や可視化操作などをデモンストレーションする学生の秋澤和輝氏でした。また、情報基盤センターの中島研吾先生には Wisteria システムのご紹介をいただきました。ハンズオンセミナーの参加者数は、民間企業 7 名、公的研究機関 1 名、大学 2 名、その他所属 1 名の合計 11 名(想定定員)でした。当日の進行は以下のとおりです。

13:00-13:20	ABINIT-MP の整備状況、新機能 【望月】
13:20-13:35	ABINIT-MP による相互作用解析の事例 【望月】
13:35-13:50	スーパーコンピュータ「Wisteria/BDEC-01」の紹介 【中島先生】
13:50-14:00	Wisteria/BDEC-01(Odyssey)システムへの接続
14:00-15:00	実習 1 : 相互作用エネルギー解析の事例 【奥脇氏】
15:00-15:20	休憩
15:20-17:00	実習 2 : PIEDA を用いた詳細解析の事例、ものづくり系の事例 【奥脇氏】
17:00-17:10	ABINIT-MP の今後の研究開発 【望月】

ABINIT-MP の説明では、基本機能と Open シリーズの展開、ならびにスーパーコンピュータでの高パフォーマンスをご紹介し、さらに現在準備中の Ver. 2 Rev. 8 (2022 年 8 月リリース予定)についても触れました。

実習 1 では、Odyssey の実習用ジョブクラスを用いて実際に例題を投入し、FMO 計算か

ら得られる最も有用な数値指標であるフラグメント間相互作用エネルギー(IFIE)による対象系の解析を BioStation Viewer による可視化を交えて行いました。休憩後の実習 2 では、IFIE の成分分解である PIEDA による詳細な理解の仕方を解説すると共に、可視化を伴う軌道レベルの相互作用解析もご紹介しました(実際の実行は Aquarius で行う必要あり)。また、生体分子系の解析に加え、シリカのナノ粒子モデルへのエチレンの吸着なども取り上げ、ものづくり系問題への FMO の応用の可能性の一端を視ていただきました。オンラインでの実施でしたので、オンサイトよりも少しペースを控えての進行となりましたが、大きなトラブルもなく終えることができました。

終了後のアンケートは概ね好評で、説明や実習が「分かりやすい」の項目で 60-80% の評価を頂戴しましたので、ABINIT-MP による FMO 計算の概要をご理解いただけたようです。また、実習の内容・事例を増やして欲しいとのご意見もいただきましたので、次の機会には午前からの開始とするなど、より充実したセミナーとなるように検討したいと思います。



図1 オンラインでのハンズオンセミナーの様子(画像キャプチャ)

【ABINIT-MP の参考アドレス】

http://www.cenav.org/abinit-mp-open_ver-1-rev-22/

http://www.cenav.org/abinit-mp-open_ver-2-rev-4/

東京大学工学部・工学系研究科共通科目「スパコンプログラミング(1)および(I)」

埴 敏博

東京大学情報基盤センター

1. はじめに

工学部および工学系研究科の共通科目「スパコンプログラミング(1)および(I)」を通年科目(Sセメスター、Aセメスター共通)として開講している。前任者から引き継いで2016年度から実施しているものであり、2021年度Aセメスターで連続30回の開講を数える。後期教養教育科目にも登録されており[1]、後で述べるように、工学部や工学系研究科以外の学生も数多く受講している。また、2017年度より計算科学アライアンスの認定講義にもなっている[2]。

2021年度は、本講義の受講生に対して、2021年5月に運用を開始したWisteria/BDEC-01を用いた演習を行った。Sセメスターでは、運用開始直後の5月18日に、いち早くアカウントを配布し、富士通A64FX CPUを搭載したWisteria-Odysseyを用いて演習を開始した。また、講義の終盤には、Intel Xeon IceLake, NVIDIA A100 GPUを搭載したWisteria-Aquariusを用いた演習も行った。受講生にとって、スーパーコンピュータ富岳と同じアーキテクチャを持つA64FX CPUなど、最新、最先端のスパコン環境を使用できることで、より高い動機付けになると考えている。

Wisteria-Odysseyでは12ノード(48コア×12ノード=576コア)、Wisteria-Aquariusでは4GPUまで1ジョブで利用可能であり、教育利用であっても十分に高い並列数を扱うことができている。

2021年度については引き続き、コロナ禍により、Sセメスターの開始時点から、全てZoomを用いたオンライン講義とした。同様の形式になって2年目であり、概ね順調に進めることができた。これまでも講義資料はWebに掲載していたが、加えて、Zoomの録画をLMSに掲載することで、学生からは復習ができて良いと好評であった。演習の際に、Zoom共有画面での操作のデモと、自分の端末とを見比べて作業がしやすいとのコメントがあった。以上を踏まえて、今後も引き続きオンラインでの講義を想定している。

2. 講義内容

本講義で行った講義内容を表1に示す。内容については、C, Fortranに加えて、Pythonを用いた並列処理や機械学習フレームワークの使用方法についても紹介した。Sセメスターにおいては、Wisteria/BDEC-01が稼働を開始する5月15日以降にスパコンを利用開始するため、第2回より先行して第3、4回を実施し、その間は手元のPCを用いて演習を行ってもらった。アプリケーションは、行列-ベクトル積、ベキ乗法(行列-ベクトル積を応用した固有値・固有ベクトルの初等的な数値計算法)、行列-行列積、LU分解法の4種類である。

加えて、講義の一環として、受講生が参加できる「プログラミングコンテスト」を開催した。すべての出題を解答できるプログラムを提出したコンテストの参加者には、成績に加点した。さらに、コンテストにおける入賞者(1位~3位)においては、無条件で「優」以上を与えた。

演習には、表 2 に示す 11 本のサンプルプログラムを教材として用いている。受講者はこれらをダウンロードした上で、動作確認した上で演習を実施する。表 2 最後の Python によるものを除いて、それぞれ、C 言語版と Fortran 版を用意している。

表 1 講義内容

講義回数	講義内容
ガイダンス	初回ガイダンス、高性能計算の基礎
第 1 回	並列数値処理の基本演算
第 2 回 (S: 第 4 回)	スパコンを利用しよう：スパコン(Wisteria-Odyssey)を用いた実習
第 3 回 (S: 第 2 回)	高性能プログラミング技法の基礎(1)：階層メモリ、パイプライン処理、ループアンローリング、キャッシュヒット率
第 4 回 (S: 第 3 回)	高性能プログラミング技法の基礎(2)：キャッシュブロック化、OpenMP 超入門
第 5 回	行列-ベクトル積
第 6 回	べき乗法
第 7 回	行列-行列積(1)：ループ交換法、ブロック化(タイリング)法、Cannon のアルゴリズム、Fox のアルゴリズム、簡単な並列化
第 8 回	行列-行列積(2)：完全な並列化
第 9 回	LU 分解法(1)：LU 分解法(ガウス・ジョルダン法、ガウス消去法、枢軸選択、LU 分解法(外積形式、内積形式、クラウト法、ブロック形式ガウス法、縦ブロックガウス法、前進・後退代入))、コンテスト課題発表
第 10 回	LU 分解法(2)：LU 分解のアルゴリズム詳細
第 11 回	GPU プログラミング(1)：(OpenACC による Wisteria-Aquarius の利用、性能比較)
第 12 回	Python & 機械学習

表 2 サンプルプログラム一覧

	サンプルプログラム内容
Samples-wo (第 4 回(S), 第 2 回(A))	並列版 Hello プログラム、並列円周率計算プログラム、逐次転送方式による並列総和演算プログラム、二分木通信方式による並列総和演算プログラム、時間計測方法の並列プログラム
Mat-Mat-noopt-pc (第 2 回(S)) Mat-Mat-noopt-wo (第 3 回(A))	行列-行列積の逐次プログラム(逐次チューニング用)
Mat-Mat-openmp-pc (第 3 回(S))	行列-行列積の逐次プログラム(OpenMP 並列化用)

Mat-Mat-openmp-wo (第 4 回(A))	
Mat-vec-wo (第 5 回)	行列-ベクトル積の逐次プログラム
PowM-wo (第 6 回)	ベキ乗法の逐次プログラム
Mat-Mat-wo (第 7 回)	行列-行列積の逐次プログラム (お手軽並列用)
Mat-Mat-d-wo (第 8 回)	行列-行列積の逐次プログラム (完全分散並列用)
LU-wo (9, 10 回)	LU 分解法による連立一次方程式の求解の逐次プログラム
Isend-ofp (第 10 回)	非同期通信の並列プログラム
Mat-mat-acc (第 11 回)	行列-行列積の逐次プログラム (OpenACC 並列化用)
Python (第 12 回)	Python による並列プログラム、機械学習サンプル

3. 受講者についての統計データ

(1) 受講者数

2021 年度の S セメスターの履修登録者は 43 名 (学部 : 17 名、大学院 : 26 名)、A セメスターの履修登録者は 39 名 (学部 : 21 名、大学院 : 18 名) であった。そのうち、単位取得者は、37 名 (学部 : 10 名、大学院 : 27 名) であった。

(2) 受講者の所属

今年度の履修登録者について、以下の通りである。

- 医学部: 1 名
 - 医学部医学科 1
- 工学部: 31 名
 - 工学部社会基盤学科 1
 - 工学部建築学科: 1
 - 工学部機械工学科: 3
 - 工学部航空宇宙工学科: 1
 - 工学部精密工学科: 2
 - 工学部電子情報工学科: 8
 - 工学部電気電子工学科: 1
 - 工学部計数工学科: 1
 - 工学部マテリアル工学科: 4

- 工学部応用化学科：1
- 工学部化学システム工学科：3
- 工学部化学生命工学科：1
- 工学部システム創成学科：4
- 理学部：3名
 - 理学部情報科学科：1
 - 理学部物理学科：1
 - 理学部生物情報科学科：1
- 経済学部：1名
 - 経済学部金融学科：1
- 教養学部：1名
 - 教養学部学際科学科：1
- 教育学部：1名
 - 教育学部総合教育科学科：1
- 総合文化研究科：1名
 - 総合文化研究科広域科学専攻：1
- 理学系研究科：5名
 - 理学系研究科物理学専攻：2
 - 理学系研究科地球惑星科学専攻：3
- 工学系研究科：27名
 - 工学系研究科社会基盤学専攻：1
 - 工学系研究科建築学専攻：1
 - 工学系研究科機械工学専攻：3
 - 工学系研究科航空宇宙工学専攻：4
 - 工学系研究科システム創成学専攻：2
 - 工学系研究科電気系工学専攻：6
 - 工学系研究科物理工学専攻：4
 - 工学系研究科マテリアル工学専攻：3
 - 工学系研究科化学システム工学専攻：1
 - 工学系研究科原子力国際専攻：2
- 医学系研究科：2名
 - 医学系研究科社会医学専攻：1
 - 医学系研究科内科学専攻：1
- 新領域創成科学研究科：5名
 - 新領域創成科学研究科先端エネルギー工学専攻：2
 - 新領域創成科学研究科複雑理工学専攻：1
 - 新領域創成科学研究科人間環境学専攻：2
- 情報理工学系研究科：4名
 - 情報理工学系研究科コンピュータ科学専攻：1
 - 情報理工学系研究科電子情報学専攻：3

以上から、工学系・情報系を中心に、学内の多様な学部・学科から受講生が集まっていることがわかる。

本講義を通じて、本センターのスーパーコンピュータシステムの利用者を育成するだけでなく、スーパーコンピュータの活用を通じて学際領域分野に貢献できる人材を輩出していると考えられる。また、スパコン利用の経験が多くの学生に広がることにより、学界のみならず、産業界においてスパコンを利活用できる人材育成につなげていけるものと考えている。

Wisteria/BDEC-01 については、導入直後のタイミングから使用を開始したが、Fortran と C 言語でコンパイラの違いによる性能差が時折見られるものの、安定して利用できた。関係諸氏に深く感謝する。

参 考 文 献

[1] 東京大学 後期教養教育科目について

<http://www.u-tokyo.ac.jp/stu04/koukikyouyou.html>

[2] 東京大学 計算科学アライアンス <http://www.compsci-alliance.jp/>

第 170 回お試しアカウント付き並列プログラミング講習会

「Wisteria 実践」実施報告

埴 敏博

東京大学情報基盤センター

2021 年 12 月 13 日（月）に、第 170 回お試しアカウント付き並列プログラミング講習会「Wisteria 実践」が開催されました。新型コロナウイルス感染症対策のために Zoom を用いたオンライン講習会として実施されました。

本講習会は、東京大学内および学外における当センターのスーパーコンピュータの利用を考えているユーザに加え、社会貢献の一環として、高性能計算や並列処理の技術習得を目的にした企業に所属する研究者、技術者の方が参加可能になっております。

受講者は、学部学生：1 名、大学院学生：3 名、大学・研究機関教職員：1 名、計 5 名の方にご参加いただきました。

1 ヶ月有効となるお試しアカウントが与えられ、5 月 14 日に運用を開始したばかりの Wisteria/BDEC-01 スーパーコンピュータシステムを用いて、Wisteria/BDEC-01 の利用方法、特に、OpenMP, OpenACC および MPI (Message Passing Interface) を用いた実践的なプログラミングと実行方法について、講義と演習が以下の日程で行われました。

当日のプログラムを、以下に掲載します。

- 12 月 13 日（月）
 - 10:30 – 12:00 Wisteria/BDEC-01 システム紹介
 - 13:30 – 15:15 Odyssey ノード：A64FX における OpenMP 最適化、MPI+OpenMP ハイブリッド並列、性能分析（講義＋演習）
 - 15:30 – 17:15 Aquarius ノード：A100 GPU の利用、OpenACC, MPI 並列化、性能分析（講義＋演習）
 - 17:15 – 17:30 Q&A

3 名の参加者について、講習会に関するアンケートをご提出いただきました。主要な項目の集計結果を以下に示します。

プログラミング経験については、1 年から、30 年という方もいらっしゃいました。並列プログラミングについては、知識を前提にしておりましたが、経験なしの方も 1 名いらっしゃいました。使用しているプログラミング言語については、Fortran が 3 名と多く、Python, C と C++ が同数の 1 名（複数回答可）でした。

講義内容や配布資料については高評価で、講習会の満足度は全員が3.0(ふつう)でした。

今回はZoomを用いての完全オンライン開催であり、東京に行かなくても受講できる点が良かったとのコメントをいただきました。

同様の講習会があれば、「また受けない」という回答が2名、「どちらともいえない」が1名で、感想からもその他の講習会にも期待されていることが伺えます。

「Wisteria 実践」については、今後も内容を見直しながら、定期的を開催していきたいと考えております。

また、本講習会資料、録画データも公開しております。復習に役立てたり、参加が適わなかった方に自習に使っていただければ幸いです。

しばらくは新型コロナウイルス感染症対策でオンラインのみの開催が続きます。オンライン講習会にはオンサイト講習会にない利点があることも分かってきたので、今後オンサイト開催が可能になってもオンラインを考慮しながら内容を検討していく予定です。

以上

第 171 回お試しアカウント付き並列プログラミング講習会

一日速習：有限要素法プログラミング徹底入門（オンライン）

中 島 研 吾

東京大学情報基盤センター

本稿では、2021 年 12 月 21 日（火）にオンライン開催した第 160 回お試しアカウント付き並列プログラミング講習会「一日速習：有限要素法プログラミング徹底入門¹」（共催：東京大学情報基盤センター、PC クラスタコンソーシアム（実用アプリケーション部会・HPC オープンソースソフトウェア普及部会））について紹介する。

有限要素法（Finite Element Method, FEM）は偏微分方程式の数値解法として、様々な科学技術計算に使用されている。基礎的な研究開発の他、NASTRAN に代表される有限要素法による商用コードも既に半世紀近く産業利用も含む様々な分野で設計・安全評価などに使用されている。有限要素法は要素単位のローカルな演算から構成されているところから、並列化が比較的容易であることが知られている。

本センターでは、本センターのスーパーコンピュータの利用促進と並列プログラミング技術の普及を目的として、様々な並列有限要素法の講習会²を実施してきた。

有限要素法の理論、手法は大学教養程度の数学、物理の知識があれば十分に理解できるものがあるが、並列化を実施するためには、単に個々の事項の理解に留まらず、数学的背景、数値アルゴリズムとプログラミングを結びつけて理解している必要がある。有限要素法そのものと並列化を 1 日で全て解説するのは、スケジュール的にも困難で、細かいところは省略せざるを得ないため、受講者にとっても消化不良となる傾向があった。

本センターでは、2020 年 6 月 12 日に第 134 回講習会³で、初めての試みとして、並列化には立ち入らず、**スパコンを使用したハンズオンも実施せず**、有限要素法を構成する基礎的な理論、数値アルゴリズムとその実装に主眼を置いた講習会を 1 日で実施したところ、比較的好評であり需要が高いことも確認できた。その後、2020 年 11 月 10 日に同様の内容で第 145 回講習会⁴を実施し、今回はそれらのフィードバックから得られた知見も踏まえ、更に改良した詳細な教材を提供した。

本講義の内容（表 1）は 2020 年度冬学期に講義として実施した、科学技術計算Ⅱ（大学院情報理工学系研究科数理情報学専攻）／コンピュータ科学アライアンス特別講義Ⅱ（同 コンピュータ科学専攻）／ハイブリッド分散並列コンピューティング（大学院工学系研究科電気系工学専攻）「並列有限要素法入門」⁵の導入部と同じ内容を日本語で実施したものであり、最初の「有限要素法入門⁶」については、時間の都合もあり、受講者に教材を予習してもらい、当日は簡単な説明に留めた。

今回は、前述のようにスパコンは利用せず、各自の PC に一次元・三次元有限要素法プログラ

¹ <https://www.cc.u-tokyo.ac.jp/events/lectures/171/>

² <https://www.cc.u-tokyo.ac.jp/events/lectures/116/>

³ <https://www.cc.u-tokyo.ac.jp/events/lectures/134/>

⁴ <https://www.cc.u-tokyo.ac.jp/events/lectures/145/>

⁵ <http://nkl.cc.u-tokyo.ac.jp/20w>

⁶ <http://nkl.cc.u-tokyo.ac.jp/FEM/01-FEMIntro.pdf>

ム (Fortran・C) をダウンロードしてもらい、数学的・理論的な背景から、実問題（定常熱伝導）まで、実習も交え、大規模連立一次方程式の反復解法（前処理付き共役勾配法）も含めて解説した。詳細はホームページ (<https://www.cc.u-tokyo.ac.jp/events/lectures/171/>) を参照されたい。当日利用した資料、サンプルプログラムの他、録画したビデオを見ることもできる。

表 1 本講習会の概要

09 : 00-09 : 45	有限要素法入門
09 : 45-12 : 30	一次元有限要素法
13 : 30-16 : 00	三次元有限要素法
16 : 00-17 : 00	並列有限要素法への道
17 : 00-17 : 15	討論・質疑等

事前登録者は 4 名、出席者は 4 名（学生：3 名，大学・研究機関：0 名，企業：1 名）となった。講習会終了後にアンケートを実施した（回収本数：3）。全体的な満足度の平均値は 5 点満点で 4.67 となった。オンライン講習会も大分定着してきており，気軽に参加できるということで概して好評である。

アンケートの自由記述欄のコメントを以下に示す：

- 丸一日の講習でしたが，有限要素法についてここまで説明をいただける講習会は無いと思いました。全体的な流れ，簡単なプログラム，その構成や説明等非常に勉強になりました。特に，プリコンディション，疎行列格納法，CG 法等ここまで分かりやすく（式が多いので資料の見直しは必要ですが），Paraview へのはきだしも参考になります。合間をみて，再度勉強をやりたくなりました。

第 172 回お試しアカウント付き並列プログラミング講習会

「OpenFOAM 中級・3 次元ダムブレイク流れ」実施報告

今野 雅

東京大学情報基盤センター客員研究員

2022 年 1 月 18 日 (火), PC クラスタコンソーシアム (実用アプリケーション部会・HPC オープンソースソフトウェア普及部会)、オープン CAE 学会との共催で、第 172 回お試しアカウント付き並列プログラミング講習会「OpenFOAM 中級・3 次元ダムブレイク流れ」がオンラインで開催された。本講習会は、センターに設置されたスーパーコンピューター(以降、スパコン)の利用促進とスパコンを用いた数値流体解析の普及を目的として実施されたものである。なお、本講習会はセンターのお試しアカウント付き並列プログラミング講習会として行われた 21 回目の OpenFOAM の講習会である。受講者は、大学院学生 6 名、企業の方 12 名であり、事前申込者 20 名、受講者合計は 18 名であった。センターが運営するスパコン Wisteria/BDEC-01 を用い、Wisteria/BDEC-01 の概要、利用方法、OpenFOAM の演習が 1 日終日の日程で行われた。当日のプログラムを表 1 に掲載する。なお、講習会終了後約 1 ヶ月有効なお試しアカウント(Wisteria-0, 最大ノード数 12, 最大実行時間 15 分)が受講者に与えられた。

表 1 講習会プログラム

【2022 年 1 月 18 日 (火)】

10 : 00 - 12 : 00 3 次元ダムブレイク流れ演習 I

イントロダクション (約 20 分間)

講習会の準備 (約 15 分間)

module による OpenFOAM の環境設定 (約 10 分間)

解析対象(Kleefsman らによる段波・構造物衝突流れ実験) (約 7 分間)

OpenFOAM の二相流解析ソルバの概要 (約 12 分間)

解析手順 (約 4 分間)

blockMesh によるベース格子生成 (約 24 分間)

ParaView によるベース格子の可視化 (約 22 分間)

13 : 00 - 18 : 00 3 次元ダムブレイク流れ演習 II

snappyHexMesh による格子生成 (約 31 分間)

ParaView による格子の可視化 (約 19 分間)

interFoam を用いた流体解析 (約 100 分間)

実測値との比較 (約 4 分間)

ParaView による計算結果の可視化 (約 16 分間)

interIsoFoam を用いた解析 (約 13 分間)

解適合格子を用いた解析 (約 8 分間)

二相流解析ソルバのカスタマイズ例の紹介 (約 3 分間)

解析条件のベイズ最適化例の紹介 (約 10 分間)

講習会終了後のアンケート集計結果(回答数 15)を表 2 に示すが、参加した満足度の平均は 5 点満点中、4.13 と高かった。また、参加者から表 3～5 に示すご意見を頂いた。今後の講習会の参考にしたい。

表 2 アンケート集計結果

評 点	講習会の時間		講習会の講義内容 (プレゼン)		配布資料の内容		サンプルプログラム 内容		参加した満足度	
1	短い	0	簡単	0	簡単	0	簡単	1	不満	0
2		0		2		2		2		0
3	適切	9	適切	7	適切	10	適切	11	普通	4
4		5		5		1		1		5
5	長い	1	難	1	難	2	難	0	満足	6
	平均	3.47	平均	3.33	平均	3.20	平均	2.80	平均	4.13

表 3 本講習会に対するご意見(原文ママ)

- linux コマンド等の予習が必要だった。普段は汎用のソルバーを使用しているが、OpenFoam については今後も学習していきたい。
- ありがとうございます。
- スパコンならではの OpenFOAM の使い方や、OpenFOAM の細かい設定について知ることができて良かったです。
- 非常に有意義な講習でした！
- 講習会の開催、ありがとうございます。大変勉強になりました。
- 講習内容とは直接関係ありませんが、VNC で接続したログインノードがかなり短い時間でロックしてしまうので、毎回パスワードを入力する必要がありました。できれば、画面をロックしない設定の説明資料があるとよかったかなと思います。
- お忙しいところ講習会を開催していただきありがとうございます。気液二層流にフォーカスした様々な解析手法やポスト処理方法など、自分のレベルよりワンランク上の情報が得られて有益でした。
- 説明が丁寧でよかった。スパコンの使用方法よりも、OpenFOAM の使用方法で学ぶことが多かった。最後のほうが駆け足になったのが残念。
- OpenFOAM とスパコンの使用方法について勉強できたため、参加して良かった。

表 4 Zoom によるオンライン講習会で良かったこと (原文ママ)

-
- 会場へ行く手間が省ける
 - 画面を比較しながらプログラミングができること
 - 自宅で受けることができたのが良かったです
 - 操作画面が見やすいこと。遠隔地からでも参加できること。
 - 丁寧に対応してくださった事
 - 移動の手間がない。参加の許可が下りやすい。
 - 録画できることでしょうか。
 - 音声、映像ともにスムーズだった。
 - 以前受講した時よりも参加者とのコミュニケーションが盛んで、よりオフラインに近いように感じました。
 - 対面の講習会と遜色なし。自席のサイズの大きいモニタが使用できるので、資料と操作画面の見比べができるなど効率がよかったと思う。
 - 自宅から手軽に参加できること
 - 特になし
-

表 5 Zoom によるオンライン講習会で悪かったこと (原文ママ)

-
- 途中でつまって脱落してしまうとキャッチアップしにくい
 - 質問を聞きにくいこと
 - 特になし。
 - 悪かったことは特になかったです。
 - 特になし。
 - 特にありません。
 - 特になし
 - Zoom と pdf と Apache Guacamole の 3 つのウィンドウを同時にみたいですが、スクリーンにすべて表示しきれず、結局講師の操作をよく見逃してしまいます。
 - 普段 zoom を使用していないため、使いづらいと感じた。例えば誰かが画面共有した際に一度全画面表示になるため、ワンディスプレイのみの参加者の場合、一々全画面表示を解除する必要があり作業性が悪いと感じた(設定があるのかもしれないが)。普段は teams というアプリを使用している。
-

原稿募集

本誌では利用者の皆様からの原稿を募集しています。以下の執筆要項に基づいて投稿してください。

執筆要項

- 1 内容は、本センターのスーパーコンピューターシステムの利用者にとって有意義な情報の提供となる原稿とします。
- 2 掲載可否については当編集委員会で決定させていただきます。
- 3 掲載可とした投稿原稿に対して、加除訂正を行うことがあります。
- 4 原稿枚数には特に制限はありませんが、シリーズに分割することもあります。
- 5 プログラムの実例が大量になる場合（概ね1頁を超える）は、本文には一部のみを記述し、投稿者の Web ページ等に全体を掲載し、その URL を引用するようにしてください。
- 6 原稿は横書きにしてください。
- 7 原稿は、A4 サイズで、ページの余白は上下 20mm、左右 26mm、ヘッダー15mm、フッター 10mm に設定してください。詳しくは原稿様式をご参照ください。PDF 形式(フォント埋め込み)の完全原稿を電子メールにて uketsuke@cc.u-tokyo.ac.jp まで提出願います。
- 8 採用された原稿は、本センターの Web ページに掲載いたします。
<https://www.cc.u-tokyo.ac.jp/public/news.php>

【スーパーコンピュータシステム利用案内】

お知らせ	Web ページ
サービス案内、運転状況など	https://www.cc.u-tokyo.ac.jp/
公開鍵登録、マニュアル閲覧など	https://wisteria-www.cc.u-tokyo.ac.jp/ (Wisteria/BDEC-01) https://obcx-www.cc.u-tokyo.ac.jp/ (Oakbridge-CX) https://ofp-www.jcahpc.jp/ (Oakforest-PACS)

お問い合わせ内容	お問い合わせ先
利用申込関係	スーパーコンピュータシステム利用申込書提出先 uketsuke@cc.u-tokyo.ac.jp 東京大学情報システム部 情報戦略課研究支援チーム
プログラム相談・システム利用に関する質問	https://www.cc.u-tokyo.ac.jp/supports/contact/#SOUDAN
システムに関する要望・提案	voice@cc.u-tokyo.ac.jp

【IP ネットワーク経由時のホスト名】

シ ス テ ム	ホ ス ト 名
Wisteria/BDEC-01 スーパーコンピュータシステム (Wisteria-0/A)	wisteria.cc.u-tokyo.ac.jp 以下のホストの何れかに接続します※ wisteria0{1-6}.cc.u-tokyo.ac.jp
Oakbridge-CX スーパーコンピュータシステム	obcx.cc.u-tokyo.ac.jp 以下のホストの何れかに接続します※ obcx0{1-6}.cc.u-tokyo.ac.jp
Oakforest-PACS スーパーコンピュータシステム	ofp.jcahpc.jp 以下のホストの何れかに接続します※ ofp0{1-6}.jcahpc.jp

※どのホストに接続しても同じです。

【編集】

東京大学情報基盤センタースーパーコンピューティング研究部門
 東京大学情報システム部情報基盤課スーパーコンピューティングチーム
 〃 情報戦略課研究支援チーム

【発行】

東京大学情報基盤センター
 〒277-0882 千葉県柏市柏の葉6-2-3
 (電話) 04-7133-4663 (ダイヤルイン)

目 次

センターから

サービス休止等のお知らせ	1
システム変更等のお知らせ	3
利用負担金の改正について	5
共通ストレージシステム利用規程及び別表の制定について	10
「大規模HPCチャレンジ」採択課題のお知らせ	14
「大規模HPCチャレンジ」課題募集のお知らせ	15
研究成果登録のお願い	18
12月・1月のジョブ統計	19

研究報告

第41回ASE研究会報告 Ode to Numerical Linear Algebra	23
Wisteria/BDEC-01 利用事例 (3) データ受け渡しライブラリ h3-Open-SYS/WaitIO(1/2)	24
Wisteria/BDEC-01 (Odyssey) における OpenMP による プログラミング入門 (その1)	44

ユーザーから

温水冷却の効率に関する定量的評価の試み	52
面内空孔が層状鉱物の原子スケールの摩擦特性へ与える影響	58
科学技術計算の効率的超並列化に向けた静的負荷分散を行うDSLの開発	62
CENP-T 天然変性領域の翻訳後修飾が結合機構へ及ぼす影響の研究	69
マルチスケール分子シミュレーションによる受容体チロシンキナーゼの 構造モデリング	74
機械学習による時系列データの学習過程の解明	79

教育活動報告

ABINIT-MP講習会－Wisteria/BDEC-01 (Odyssey) を用いた ハンズオン－実施報告	83
東京大学工学部・工学系研究科共通科目 「スパコンプログラミング(1)および(I)」	85
第170回お試しアカウント付き並列プログラミング講習会 「Wisteria実践」実施報告	90
第171回お試しアカウント付き並列プログラミング講習会 「一日速習：有限要素法プログラミング徹底入門」(オンライン)	92
第172回お試しアカウント付き並列プログラミング講習会 「OpenFOAM中級・3次元ダムブレイク流れ」実施報告	94

原稿募集	97
------	----