（オンライン） ISSN 2436—4940

スーパーコンピューティングニュース

Vol.24 No.3, 2022.5

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）

区分	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 （基本セット） Wisteria-O/A	申込 1 セット当り 60,000 円 （8,640 トークン）		申込 1 セット当り /work 2TB 利用者当り /home 50GB	
公募制度による申込 Wisteria-O	申込 1 セット当り 60,000 円 （8,640 トークン）	申込 1 セット当り 72,000 円 （8,640 トークン）	申込 1 セット当り /work 2TB 利用者当り /home 50GB	利用期間 12 か月の金額・トークン量 （利用期間は 1 ヶ月単位で設定可）
公募制度による申込 Wisteria-A	申込 1 セット当り 180,000 円 （25,920 トークン）	申込 1 セット当り 216,000 円 （25,920 トークン）	申込 1 セット当り /work 6TB 利用者当り /home 50GB	
GPU 専有申込 （公募制度の申込可） Wisteria-A	申込 1GPU セット当り 270,000 円 （25,920 トークン）	申込 1GPU セット当り 324,000 円 （25,920 トークン）	申込 1GPU セット当り /work 6TB 利用者当り /home 50GB	利用期間 12 か月の金額・トークン量 （利用期間は 1 ヶ月単位で設定可） 1, 2, 4GPU のみ申込可， 申込単位は下表参照
ノード固定 （公募制度の申込可） Wisteria-A	2,160,000 円 （207,360 トークン）	2,592,000 円 （207,360 トークン）	/work 48TB 利用者当り /home 50GB	利用期間 12 か月の金額・トークン量 （利用期間は 1 ヶ月単位で設定可） 1 セットのみ申込可
一般申込 （最小セット） Wisteria-O/A	5,000 円 （720 トークン）		/work 2 TB 利用者当り /home 50 GB	利用期間は当該年度末まで
トークン量追加	5,000 円 （720 トークン）	6,000 円 （720 トークン）		
ディスク容量追加	6,480 円／（1TB*年）			1TB 単位で申込可 （/work のみ）

※Wisteria/BDEC-01 においては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。

※Wisteria-O の 1 ジョブで利用可能な最大ノード数は 2,304 ノード，Wisteria-A の 1 ジョブで利用可能な最大 GPU 数は 64GPU

※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トークン量の換算表」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)

区分	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 (8,640 トークン)	申込 1 セット当り 120,000 円 (8,640 トークン)	グループ 1 セット当り /work 4TB 利用者当り /home 50GB	利用期間 12 か月の金額・トークン量 (利用期間は 1ヶ月単位で設定可)
ノード固定	申込 1 セット当り 150,000 円 (8,640 トークン)	申込 1 セット当り 180,000 円 (8,640 トークン)		
一般申込 (最小セット)	8,400 円 (720 トークン)		/work 4 TB 利用者当り /home 50 GB	利用期間は当該年度末まで
トークン量追加	8,400 円 (720 トークン)	10,000 円 (720 トークン)		
ディスク容量追加	6,480 円／(1TB*年)			1TB 単位で申込可 (/work のみ)

※Oakbridge-CX においても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。

※トークン消費係数は 1.00, ただしトークン消費係数 1.50 のノード群(優先利用向け)を全体の 15%程度設ける。

※1 ジョブで利用可能な最大ノード数は 256 ノード。

※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トークン量の換算表」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。

トークン移行におけるトークン量の換算表

移行元＼移行先	Oakbridge-CX システム	Wisteria/BDEC-01 システム
Oakbridge-CX システム	ー	1.6
Wisteria/BDEC-01 システム	0.6	ー

移行先に追加されるトークン量＝移行トークン量×係数

注意事項(Wisteria/BDEC-01, Oakbridge-CX 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。
- 利用負担金は, 原則として利用開始月に応じ, 以下の月に一括して請求する。
  - 利用開始月が 4 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, 11 月の請求となる。
- 利用負担金額が減額となる変更はできない。
- ディスク量は, グループ全体の上限値である。

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
debug-o	1 ～ 144 (6,912)	30 分	28	○	○
short-o	1 ～ 72 (3,456)	8 時間	28	○	○
(regular-o)					
small-o	1 ～ 144 (6,912)	48 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	〃	○	○
priority-o	1 ～ 288 (13,824)	48 時間	28	○	○
challenge-o	1 ～ 7,680 (368,640)	24 時間	28	★	★
(interactive-o) ※4					
interactive-o_n1	1 (48)	30 分	28	○	○
interactive-o_n12	2 ～ 12 (576)	10 分	〃	○	○
prepost	1 (56)	6 時間	340	○	○
prepost1_n1 ～ prepost4_n1	1 (56)	1～6 時間	340	○	○
prepost1_n4	1 ～ 4 (224)	1～6 時間	340	○	○
prepost1_n8	1 ～ 8 (448)	1～6 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
debug-a	1 ノード (8)	30 分	448	○	○	○	○
short-a	1 ～ 2 ノード (16)	2 時間	448	○	○	○	○
(regular-a)							
small-a	1 ～ 2 ノード (16)	48 時間	448	○	○	○	○
medium-a	3 ～ 4 ノード (32)	〃	〃	○	○	○	○
large-a	5 ～ 8 ノード (64)	24 時間	〃	○	○	○	○
share-debug	1, 2, 4 GPU	30 分	56	○	○	○	○
share-short	1, 2, 4 GPU	2 時間	56	○	○	○	○
(share)							
share-1	1 GPU	48 時間	56	○	○	○	○
share-2	2 GPU	〃	〃	○	○	○	○
share-4	4 GPU	24 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	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	一般申込	ノード固定
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"

大規模共通ストレージシステム 利用負担金表

大規模共通ストレージシステム(第1世代、Ipomoea-01) 利用負担金表(2022 年 6 月 1 日)

<table>
<tr><th>区分</th><th>負担金額(税込)</th></tr>
<tr><td>一般申込</td><td>【大学・公共機関等 7,200 円，企業 8,640 円】(1TB 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

<table>
<tr><th>並列ファイルシステム ディスク容量</th><th>大学・公共機関等</th><th>企業</th></tr>
<tr><td>1 TB</td><td>7,200 円/年</td><td>8,640 円/年</td></tr>
<tr><td>[10 TB まで 1 TB 追加当たり]</td><td>[4,200 円/年]</td><td>[5,040 円/年]</td></tr>
<tr><td>10 TB</td><td>45,000 円/年</td><td>54,000 円/年</td></tr>
<tr><td>[100 TB まで 1 TB 追加当たり]</td><td>[3,000 円/年]</td><td>[3,600 円/年]</td></tr>
<tr><td>100 TB</td><td>315,000 円/年</td><td>378,000 円/年</td></tr>
<tr><td>[1,000 TB まで 1 TB 追加当たり]</td><td>[2,400 円/年]</td><td>[2,880 円/年]</td></tr>
<tr><td>1,000 TB</td><td>2,475,000 円/年</td><td>2,970,000 円/年</td></tr>
<tr><td>[以降 1 TB 追加当たり]</td><td>[2,100 円/年]</td><td>[2,520 円/年]</td></tr>
</table>
※ 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。</td></tr>
<tr><td>ディスク容量追加</td><td>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)

<table>
<tr><th>申込時点のディスク容量</th><th>大学・公共機関等</th><th>企業</th></tr>
<tr><td>1 TB 未満</td><td>7,200 円/年</td><td>8,640 円/年</td></tr>
<tr><td>1 TB 以上　10 TB 未満</td><td>4,200 円/年</td><td>5,040 円/年</td></tr>
<tr><td>10 TB 以上　100 TB 未満</td><td>3,000 円/年</td><td>3,600 円/年</td></tr>
<tr><td>100 TB 以上 1,000 TB 未満</td><td>2,400 円/年</td><td>2,880 円/年</td></tr>
<tr><td>1,000 TB 以上</td><td>2,100 円/年</td><td>2,520 円/年</td></tr>
</table></td></tr>
</table>

※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。
※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。
※ディスク容量追加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。
※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。

センターから

サービス休止等のお知らせ

2022 年 5 月下旬からの計算機及びストレージシステムの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
5 月 27 日（金）	9:00 ～ 22:00 までサービス休止	月末処理
6 月 24 日（金）	9:00 ～ 22:00 までサービス休止	月末処理
7 月 29 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
5 月 26 日（木）9:00 ～ 27 日（金）9:00 まで 6 月 23 日（木）9:00 ～ 24 日（金）9:00 まで 7 月 28 日（木）9:00 ～ 29 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
5 月 25 日（水）	9:00 ～ 20:00 までサービス休止	月末処理
6 月 22 日（水）	9:00 ～ 20:00 までサービス休止	月末処理
7 月 27 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水曜日） 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
5 月 24 日（火）9:00 ～ 25 日（水）9:00 まで 6 月 21 日（火）9:00 ～ 22 日（水）9:00 まで 7 月 26 日（火）9:00 ～ 27 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。

大規模共通ストレージシステム（Ipomoea-01）

○ 大規模共通ストレージシステム(第１世代) Ipomoea-01 サービス開始について

6 月 1 日（水）10:00 よりサービス運用を開始します。
詳細については，本誌別記事「大規模共通ストレージシステム(第１世代)運用に関するお知らせ」をご覧ください。

○ Ipomoea-01 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
6 月 24 日（金）	9:00 ～ 20:00 までサービス休止	月末処理

・ Ipomoea-01 は，原則 24 時間サービスを行っています。
ただし，月末処理実施のためサービスを停止する場合もあります。

【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 及び Oakbridge-CX 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

(2022.3.1 － 2022.4.30 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

2. ソフトウェア

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

- Odyssey

LAMMPS	v29Sep2021	(2022.3.31)
SingularityCE	v3.9.5	(2022.3.31)

- Aquarius

CUDA Toolkit	v11.4.4	(2022.3.31)
OpenMPI(CUDA-aware)	v4.1.1(UCX v1.11.2)	(2022.3.31)
NCCL	v2.9.6、v2.9.8、v2.9.9 v2.10.3、v2.11.4	(2022.3.31)
cuDNN	v8.2.0、v8.2.1 v8.2.2、v8.2.4	(2022.3.31)
gdrcopy	v2.3	(2022.3.31)
SingularityCE	v3.9.5	(2022.3.31)
Arm DDT/MAP	v21.1.3	(2022.3.31)
oneAPI HPC Toolkit	v2022.1.2	(2022.3.31)
NetCDF C++	v4.3.1	(2022.4.22)

- Messenger

SingularityCE	v3.9.5	(2022.3.31)

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)

Arm DDT	21.1.3	(2022.3.31)
oneAPI HPC Toolkit	v2022.1.2	(2022.3.31)

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した。

- 2021 年度の Oakforest-PACS の利用者様におきましては、2022 年 6 月に稼働開始する東京大学情

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な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は対応いたしかねますので、ご了承ください。
(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。

大規模共通ストレージシステム(第 1 世代) 運用に関するお知らせ（更新）

スーパーコンピューティングチーム

東京大学情報基盤センター(以下「当センター」)におきまして、当センターで運用する各スーパーコンピュータシステムからアクセス可能なストレージシステム：大規模共通ストレージシステム(第 1 世代) (Ipomoea-01)を 2022 年 6 月からサービス運用いたします。ここでは、ご利用に際して必要な手続き及び負担金をお知らせいたします。なお、運用開始までに、予告なく運用仕様の変更を行う場合がありますので予めご了承ください。

1. 利用申込みについて

ご利用になるための利用申込み方法の詳細は当センター Web ページもご参照ください。Ipomoea-01 をご利用になる場合には、当センターのスーパーコンピュータシステム Oakbridge-CX、Wisteria/BDEC-01 をすでにご利用かどうかにより、Ipomoea-01 の利用申込み手続きが必要なケースがあります。なお、Ipomoea-01 におけるディスク容量追加手続きは可能ですが、スーパーコンピュータシステムにユーザ ID を有するかに係わらず、容量追加のための手続きが別途必要になります。

1.1 2021 年度末に Oakforest-PACS にユーザ ID を有していなかった利用者、または、2021 年度末に Oakforest-PACS にユーザ ID を有していたが、Oakforest-PACS のファイル移行サービスを希望しなかった利用者

- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を有する場合
  利用申し込み手続きは不要です。ご利用のユーザ ID で Ipomoea-01 を利用できます(教育利用、講習会を除く)。
  - 利用者ごとの領域に 5 TB の無償分のディスク容量を付与します。
  - 登録されているスーパーコンピュータシステムで付与されているグループのディスク容量の 15 %を各プロジェクトコードのグループごとの領域として無償で付与します。(トークン移行先のシステムを除く)
- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を持たない場合
  利用申し込み手続きが必要です。無償分のディスク容量は設定されません。

1.2 2021 年度末に Oakforest-PACS にユーザ ID を有しており、ファイル移行サービスを希望した利用者

- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を有する場合
  利用申し込み手続きは不要です。ご利用のユーザ ID で Ipomoea-01 を利用できます(教育利用、講習会を除く)。
  - 利用者ごとの領域に 5 TB の無償分のディスク容量を付与します。
  - 移行を希望したグループごとの領域に 5 TB の無償分のディスク容量を付与します。ただし、トークン移行により Oakbridge-CX、Wisteria/BDEC-01 のどちらかに同プロジェクトコードのグループが存在する場合は、グループごとの領域への付与ディスク容量は、登録されているスーパーコンピュータシステムで付与されているグループのディスク容量の 15% とします。
  - 登録されているスーパーコンピュータシステムで付与されているグループのディスク容量の 15 %を各プロジェクトコードのグループごとの領域として無償で付与します。(トークン移行先のシステムを除く)
- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を持たない場合
  利用申し込み手続きは不要で、Oakforest-PACS のユーザ ID で 2022 年 11 月 30 日までは無償でご利用いただけます。2022 年 12 月 1 日以降も移行したファイルを利用するためには有償の利用申し込み手続きが必要です。
  - 利用者ごと、移行を希望したグループごとの領域に 5 TB の無償分のディスク容量を付与します。
  - 2022 年 11 月までに継続利用の申し込みがない場合、2022 年 12 月以降にファイルは削除されます。

2. 利用負担金について

Ipomoea-01 をご利用になる場合の利用負担金について、利用者様への運用を開始する 2022 年 6 月以降発生する予定です。

2.1 2021 年度末に Oakforest-PACS にユーザ ID を有していなかった利用者、または、2021 年度末に Oakforest-PACS にユーザ ID を有していたが、Oakforest-PACS のファイル移行サービスを希望しなかった利用者

- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を有する場合
  無償分のディスク容量の範囲でご利用の場合、負担金は発生しません。別途 Ipomoea-01 でのディスク容量の追加手続きを行った場合に、**追加分の利用負担金が発生します。**
- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を持たない場合
  利用申し込みディスク容量に対する負担金が発生します。

2.2 2021 年度末に Oakforest-PACS にユーザ ID を有しており、ファイル移行サービスを希望した利用者

- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を有する場合
  利用者ごとの領域については、無償分のディスク容量の範囲でご利用の場合、負担金は発生しません。移行を希望したグループごとの領域については 2022 年 11 月まで負担金は発生しません。**2022 年 12 月以降も移行を希望したグループごとの領域を利用するためには有償の利用申し込み手続きが必要です。**別途 Ipomoea-01 でのディスク容量の追加手続きを行った場合に、**追加分の利用負担金が発生します。**
- 2022 年度 Oakbridge-CX、Wisteria/BDEC-01 にユーザ ID を持たない場合
  2022 年 11 月まで負担金は発生しません。2022 年 12 月以降は**利用申し込みディスク容量に対する負担金が発生します。**

3. 運用開始後のサービスについて

3.1 ログインノードサービス

データ転送やそのためのツールを利用するための環境として、ログインノードを複数台用意する予定です。ログインノードは、当センターでサービスを行っている他のスーパーコンピュータシステムと同様に、公開鍵認証方式による接続となります。鍵登録の方法や、接続ホスト名などの詳細については、当センター Web ページや Ipomoea-01 利用支援ポータルにてお知らせいたします。

3.2 システムの利用イメージ

Ipomoea-01 のご利用イメージは、おおよそ図 2 の通りです (ただし、以下の図は、スーパーコンピューティングニュース原稿作成時のものです。今後の検討状況によっては、変更する場合もあります)。

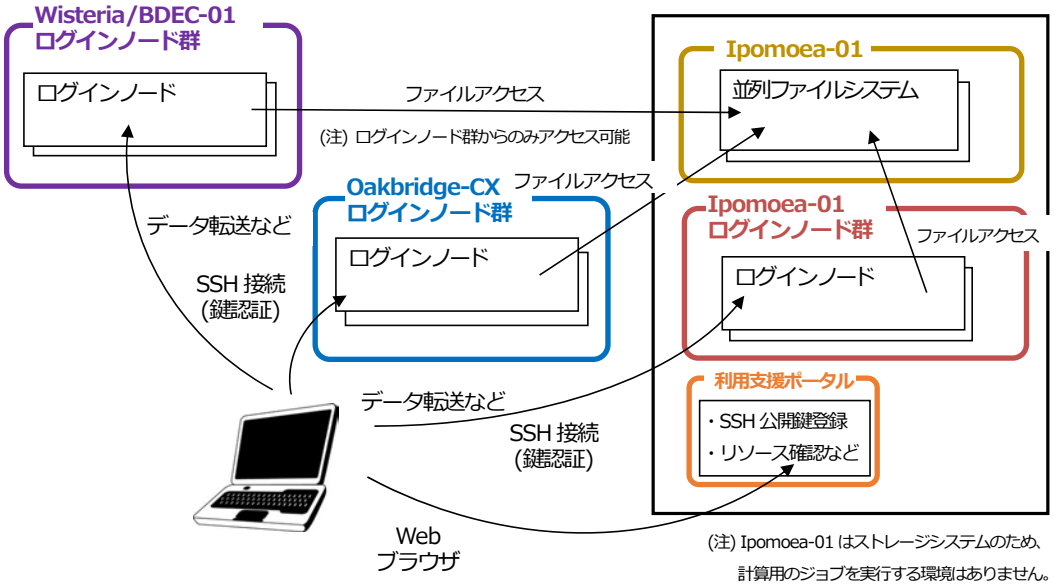


図2．システムのご利用イメージ

3.3 ファイルシステム

ファイルシステムは、システム構成図 (図 1、図 2、表 1) にある通り、並列ファイルシステムのみで構成されます。

並列ファイルシステムは DDN 社製の Lustre ベースの DDN EXAScaler で提供されます。主に以下の 2 つのディレクトリにファイルを保存することができ、Ipomoea-01 のログインノード群からだけではなく、Oakbridge-CX、Wisteria/BDEC-01 のログインノード群からもファイルアクセス(ファイルの読み込み、書き込み)が可能なように設定する予定です。その他、詳細については当センター Web ページや Ipomoea-01 利用支援ポータルにてお知らせいたします。

- /home/ユーザ名(ユーザ ID) : 利用者ごとの領域
- /work/グループ名(プロジェクトコード) : グループごとの領域

4. その他

最新の情報は、当センター Web ページ (https://www.cc.u-tokyo.ac.jp) にて随時ご案内いたします。メールによる問い合わせについては、事前に Web ページで情報がないかご確認の上、受付窓口 uketsuke@cc.u-tokyo.ac.jp までお願いいたします。

2022 年度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講習会実施予定

スーパーコンピューティングチーム

スーパーコンピューティングチームでは、全国のスーパーコンピュータ利用者、および利用を検討している新規ユーザ（企業の技術者・研究者を含む）を対象とした、スーパーコンピュータを用いた実習付きの並列プログラミング講習会（お試しアカウント付き講習会）を定期的に実施しています。

本講習会は、当センターが運用する Oakbridge-CX, Wisteria/BDEC-01 にて実施される予定です。

並列処理に関する基礎知識を必要としない初級編に始まり、数値計算の応用レベルの並列化まで、受講者の習得レベルに応じた講習会に参加が可能です。並列化には MPI (Message Passing Interface)、OpenMP または OpenACC が用いられますので、これらを用いた並列化方法の習得ができます。

講習会は無料です。なお、1 ヶ月間利用できるスーパーコンピュータ（講習会で利用するスーパーコンピュータ）のアカウントが配布されます。講習会期間に加えて講習会終了後も、講習内容に関する演習に配布アカウントが利用できます。

2022 年度の講習会開催予定を以下に掲載します。ふるってご参加のご検討をお願いします。なお本スケジュールは予定であり、日程および内容の変更が生じることがありますので予めご了承ください。講習日時および内容の詳細（過去の講習会の PDF 資料など）は、以下の HP をご覧確認ください。

https://www.cc.u-tokyo.ac.jp/events/lecture/

■2022 年度のお試しアカウント付き講習会開催【予定】

名称及び実施期間	日時（予定）	備考
Wisteria 実践	・6 月 6 日 ・9 月頃 ・2 月頃	本年度より運用を開始する Wisteria/BDEC-01 を利用予定
MPI 基礎	・4 月 26 日 ・10 月頃	MPI による並列プログラミングの基礎に関する講習、実習。Wisteria/BDEC-01 を利用予定

MPI 上級編	・10 月頃	よりハイレベルな MPI による並列プログラミングに関する講習、実習 Wisteria/BDEC-01 を利用予定
スーパーコンピュータ超入門	・4 月 22 日 ・9 月頃	スーパーコンピュータへのログイン方法やプログラムの実行方法など使い方を学ぶ Oakbridge-CX を利用
OpenFOAM	・5 月 31 日 ・9 月 27 日 ・1 月 17 日	オープンソースの CFD ツールキットである OpenFOAM を用いた講習、実習 Wisteria/BDEC-01 を利用予定
科学技術計算の効率化入門	・10 月頃	数値計算ライブラリ利用による科学技術計算の効率化に関する講習、実習 Wisteria/BDEC-01 を利用予定
GPU プログラミング入門	・5 月 18 日 ・10 月頃	GPU プログラミングに関する講習、実習 Wisteria/BDEC-01 を利用予定
GPU ミニキャンプ～HPC 編～	・7 月 12 日 ~19 日	参加者がコードを持ち込み、GPU に詳しいメンターの助言を受けながら、GPU 利用の効率化を目指す。 Wisteria/BDEC-01 を利用予定
GPU ミニキャンプ～ディープラーニング編～	・7 月 12 日 ~19 日	参加者がコードを持ち込み、ディープラーニングに詳しいメンターの助言を受けながら、コードの効率化を目指す。 Wisteria/BDEC-01 を利用予定

OpenACC と MPI によるマルチ GPU プログラミング入門	・11 月頃	マルチ GPU プログラミングに関する講習、実習 Wisteria/BDEC-01 を利用予定
OpenMP によるマルチコア・メニィコア並列プログラミング入門	・5 月 23 日 ・9 月 6 日 ・12 月 6 日	指示行を記載することで手軽に並列化する OpenMP を用いた「有限体積法から導かれる疎行列を対象とした ICCG 法」を題材にした講習、演習 Wisteria/BDEC-01 を利用予定
並列有限要素法で学ぶ並列プログラミング徹底入門	・4 月 14 日 ・11 月 1 日	有限要素法による一次元・三次元熱伝導解析プログラムを，MPI を使用して並列化するための手順について解説，実習 Wisteria/BDEC-01 を利用予定
MATLAB の実行方法	・4 月 27 日	Wisteria/BDEC-01 Aquarius で MATLAB を実行する一連の操作を演習形式で実習します。
WaitIO の利用方法	・9 月頃	Wisteria/BDEC-01 の Aquarius ノード群と Odyssey ノード群を協調利用するためのソフトウェアである WaitIO について学びます。

以上

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（GPU 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が、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 Page[1]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 Page[1] 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 23 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

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 Page[1] 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

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 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。

2. 採択課題

システム：Oakbridge-CX

募集期間：2022 年度 第 1 回募集　2022 年 2 月 2 日～2 月 28 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	前処理付き並列反復法における通信と計算のオーバーラップ
代表者名 (所属)	中島 研吾（東京大学情報基盤センター）
クリロフ部分空間法による前処理付き反復法は広範な科学技術アプリケーションに使用されている。超並列環境下では，内積おける集団通信，行列ベクトル積等における一対一通信によるオーバヘッドが顕著となる。提案者は，特に一対一通信に着目し，三次元静的弾性問題（構造力学）を有限要素法で解く場合に得られる対称正定な大規模疎行列を共役勾配法（CG 法）で解く場合について，通信と計算をオーバーラップさせ，更に OpenMP の動的スケジューリング機能により，通信オーバーヘッドを削減する手法に関する研究を実施してきた。本研究課題では，ICCG 法等データ依存性を有する処理において，通信と計算を効果的にオーバーラップさせるための手法を提案し，Oakbridge-CX 1,024 ノードを使用して効果を検証する。本研究課題で使用する GeoFEM/ICCG は，構造力学における三次元静的弾性問題を対象とした，有限要素法に基づくプログラムであり，導出された連立一次方程式を前処理付き反復法（ICCG 法）によって解く。	

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

2022年度 前期
東京大学情報基盤センター「若手・女性利用者推薦」採択課題

スーパーコンピューティングチーム

東京大学情報基盤センター（以下、センター）では、若手研究者（2022年4月1日時点において40歳以下、学生を含む）及び女性研究者（年齢は問わない）による、スーパーコンピュータ、データプラットフォームなどの大型計算資源を使用した研究を対象とした公募型プロジェクトを実施しています。センターの教員による審査の上、年間で数十件の優れた研究提案課題を採択する予定です。採択された課題では申請した計算資源を無料で使用することができます。

本制度では、前期と後期に一年または半年単位（後期は半年のみ）で実施する一名で行う研究課題を募集します。

採択された課題のうち、特に優れた課題で「学際大規模情報基盤共同利用・共同研究拠点（JHPCN）」の萌芽型共同研究課題の条件を満たすものについては、本センターより同拠点萌芽型共同研究課題として推薦する予定です。同拠点共同研究課題審査委員会で審査の上、JHPCNの萌芽型共同研究課題としても採択された場合、毎年7月に開催されるJHPCNのシンポジウムにて発表の機会が与えられる場合があります。本制度に採択された課題は終了後、得られた成果をもとに、「学際大規模情報基盤共同利用・共同研究拠点（JHPCN）」の公募型共同研究（一般課題、国際課題、企業課題）等へと進展することが大いに期待されます。

このたび、以下の基準による厳正な審査のうえ、2022年度前期は12件の課題を採択いたしました。

- 本制度が提供する計算機システムを利用することで、学術的にインパクトがある成果を創出できると期待される提案を積極的に採択します。
- スーパーコンピュータの利用環境の改善に寄与すると期待されるソフトウェア開発に関する提案を歓迎します。
- 現状の環境にとどまらず、将来の先端的なスーパーコンピュータ環境を目指した提案は特に歓迎します。
- 特に、mdxについては、理工系・人文系にまたがる多様なデータの収集・整備、研究コミュニティにおけるデータの共有やプラットフォームの整備、そして機械学習等の先端的なデータサイエンス手法を用いたデータ解析など、多様なデータ科学・データ利活用研究を歓迎します。

本制度の詳細は、以下のHPをご覧ください。
https://www.cc.u-tokyo.ac.jp/guide/young/

■ 2022年度 前期 採択課題

課題名	濡れた粉体の変形・流動特性の理解
代表者名（所属）	吉井 究（大阪大学 基礎工学研究科）
利用システム名	Oakbridge-CX
実施期間	一年

粉や砂のような巨視的なサイズの粒子の集合体である粉体は、粒子の密度が閾値より高いと、剛性を有し固体的に振る舞う一方、その閾値より低い場合は剛性を示さず流体的に振る舞う。この力学的特性の変化はジャミング転移と呼ばれ、近年盛んに研究されている。特に、粒子間に接触による反発相互作用だけが働く乾いた粉体系の転移点近傍の振る舞いについては、多くのことが明らかになっている。ところが、現実の粉体は濡れを伴う場合が多く、そのような系で乾いた粉体の挙動がそのまま観測されるかは不明である。実際、粉体は少量の水を加えるだけで、応力ひずみ曲線などのレオロジー特性が大きく変化することが知られている。これは粒子間に入り込んだ液体が表面張力によって架橋を形成し、実行的な引力相互作用を与えることに起因する。しかし、従来の濡れた粉体の研究では、個別の視点に立脚した研究は行われているが、理論的な理解や濡れの影響を系統的に調べた研究はほとんど無く、十分な理解に至っていない。そこで本研究では、濡れた粉体系の大変形や流動状態に着目し、離散要素法を用いたシミュレーションを行い、乾いた粉体系との違いを明らかにする。

課題名	t6A 修飾を含むリボソーム翻訳開始複合体における開始コドン認識動態の分子動力学計算による解析
代表者名（所属）	亀田 健（立命館大学 生命科学部）
利用システム名	mdx
実施期間	一年

申請者は、真核生物のリボソーム翻訳開始複合体における、翻訳開始コドン認識をとりまく分子動態の解明を行っている。これまでに、適切な翻訳開始コドン以外のミスマッチコドンや、修飾コドンなどを対象としてきた。本課題では、それらのミスマッチコドンや修飾コドンの翻訳開始を制御すると考えられている、アンチコドンの隣に位置する塩基に t6A 修飾が入った tRNA を含む、リボソーム翻訳開始複合体の動態に着目する。t6A は生体内での存在と疾患との関連可能性は調べられているものの、リボソーム内での動態や相互作用を含む、翻訳開始制御機構への具体的な作用メカニズムは不明である。本研究課題で行う分子動力学計算を用いた研究により、その作用メカニズムを考察するとともに、医薬研究の基盤技術へと発展させていくことを目指す。

課題名	擬スペクトル MHD コードで狙う磁気回転乱流における慣性領域の解像
代表者名（所属）	川面 洋平（東北大学 学際科学フロンティア研究所）
利用システム名	Oakbridge-CX, Wisteria/BDEC-01 Odyssey
実施期間	一年
ブラックホールは降着円盤と呼ばれるプラズマの乱流に取り囲まれている。この乱流は磁気回転不安定性 (MRI) によって駆動されている。MRI 乱流の直接数値シミュレーションは膨大な数が行われてきたが数値解像度が不十分であり、乱流の特性を捉えるに至っていない。そこで本研究では、我々が新しく開発した擬スペクトル法コードを用いて降着乱流の特性に迫る。このコードはこれまで用いられてきたコードに比べ、飛躍的に MRI 乱流の解像度を上げることができる。本課題では具体的に、これまで解像することができていなかった慣性領域に迫る。慣性領域の情報を得ることができれば、Event Horizon Telescope によるブラックホールシャドー観測の物理的解釈に必要となる理論モデルを高精度化することに繋がる。	

課題名	着陸探査プローブのクラッシャブル材とレゴリスの衝突・干渉作用に関する数値解析
代表者名（所属）	徳永 賢太郎（東京大学 工学系研究科）
利用システム名	Wisteria/BDEC-01 Odyssey
実施期間	一年
近年重力天体への着陸探査において、着陸時に必要な逆噴射装置やパラシュートといった減速装置を用いる代わりに、発泡金属やプラスチック発泡体といった、クラッシャブルなポーラス材料を着陸機構の一部に使用し着陸時のエネルギーを吸収する技術が着目されている。天体の微小重力や真空環境を地上で完全に模擬できないことも踏まえ、ミッションに用いる着陸機構の最適設計のためには、数値解析モデルを構築し各種パラメータが着陸機構に与える影響を評価する必要がある。本課題は、圧縮変形しながらエネルギーを吸収するポーラス材料とレゴリス（粒状体）との衝突・干渉による相互作用を DEM（Discrete Element Method）により数値モデル化し、ポーラス材料の各種パラメータが着陸メカニズムに与える影響を調査するものである。	

課題名	磁気ノズルスラスタにおける中性粒子流れとエネルギー輸送の数値解析
代表者名（所属）	江本 一磨（横浜国立大学 理工学府）
利用システム名	Oakbridge-CX, Wisteria/BDEC-01 Odyssey
実施期間	一年

宇宙機の主推進として利用されることが期待される磁気ノズルスラスタを研究対象とし、particle-in cell / Monte Carlo collisions / direct simulation Monte Carlo (PIC-MCC-DSMC) コードを用いて、スラスタ内部の中性粒子流れとエネルギー輸送を明らかにすることを目的とする。PIC-MCC-DSMC コードは Boltzmann 方程式で構成されるプラズマと中性粒子を確率的に解く手法であり、磁気ノズルスラスタなどの低温プラズマの解析を行う上で強力な手法として知られている。PIC-MCC-DSMC コードを用いることで磁気ノズルスラスタ内部のプラズマ・中性粒子流れを再現し、実験では測定することが困難な中性粒子とエネルギーの輸送現象を解明することが期待される。また、中性粒子とエネルギーの損失過程を理解することで、磁気ノズルスラスタの性能向上を目指す上で必要な指針を得ることができる。

課題名	次世代銀河分光観測に向けたフィールドレベル解析の確立
代表者名（所属）	大里 健（京都大学 基礎物理学研究所）
利用システム名	Wisteria/BDEC-01 Aquarius
実施期間	一年

宇宙物理学における根源的な問題であるダークマター・ダークエネルギーの正体、そして加速膨張の物理的な機構を探るため、大規模な銀河分光観測計画が予定されている。これらの計画では、銀河の三次元位置を測定し背景にある物質分布を描き出すことが主たる観測目的である。従来の解析では、観測された銀河分布から、その情報を要約する統計量を用いてダークマターの存在量に代表される宇宙論パラメータを推定する。しかしながら、統計量は銀河分布の持つ情報の一部しか取り入れられず、統計量で表現できない高次の情報は失われてしまう。そこで、本研究では観測された銀河分布の場が持つ情報の全てを考慮するフィールドレベル解析に着目し、高速で実用的な統計解析基盤を構築する。近年開発された物質密度場の重力進化理論モデル GridSPT を基礎に GPU による高速化を実装することで、ダークマター・ダークエネルギーのモデル決定に最大限迫ることを目標とする。

課題名	階層性が内在するガラスのエネルギー地形における低周波数振動の緩和予言能の起源
代表者名（所属）	白石薫平（東京大学 大学院総合文化研究科）
利用システム名	Oakbridge-CX
実施期間	一年

液体を急冷すると生じるガラスは、複雑なエネルギー地形を持つ。構成粒子がダイマーという「形」を持った粒子のとき、過冷却液体は alpha 緩和と Johari-Goldstein beta 緩和という 2 種類の緩和過程を持つ。この緩和過程の分離は、エネルギー地形からはベイスンとメタベイスンという階層構造の現れとして解釈できる。ベイスンにおける低周波数振動は結晶とは異なる性質を示し、振動粒子は熱緩和の発生位置と相関する。ダイマー系での階層的エネルギー地形という理解を踏まえると、緩和と振動の相関は非自明であり、その起源も明らかではない。本研究では、ダイマー系の大規模分子動力学シミュレーションにより、振動状態の特徴付けと緩和予言能の起源の解明を行う。レプリカ交換法を用いて広範な温度領域の平衡液体配置を生成し、振動特性の温度依存性を調べる。そして、各ベイスンにおけるモード・モード相関という新規な量を導入し、緩和予言能を生み出すエネルギー地形的な起源を明らかにする。

課題名	Constructing deep learning models of biological fitness landscapes from sequencing data
代表者名（所属）	Adam Beattie（東京大学 理学系研究科）
利用システム名	Wisteria/BDEC-01 Aquarius
実施期間	一年

Deep learning has made breakthroughs in biology but requires substantial training data. With a novel high throughput system, over 10^12 peptides can be screened for activity with post translational modification enzymes. We will use the sequencing data output to train substrate fitness landscape models. We will test our method on a biosynthetic enzyme, then build a general platform for modelling fitness landscapes for the study of uncharacterised enzymes and the design of selective therapeutics.

課題名	波形インバージョンによる南大西洋下のマントル最下部領域の地震波異方性構造推定
代表者名（所属）	大鶴 啓介（東京大学 理学系研究科）
利用システム名	Oakbridge-CX
実施期間	一年

地球マントルの最下部数百 km の領域は核-マントル境界（CMB）直上の熱境界層をなしており、外核による加熱を受けて上昇流を発生させることでマントル対流を駆動している。これまでに行われた地震波による構造推定からは、アフリカ下と太平洋下の 2 箇所に巨大な S 波低速度域（LLSVP）が存在することが知られており、これが高温の上昇流に対応するとされている。しかし、最下部マントルでの物質の詳しい流動様式が解明されていないためにこの LLSVP の成因や性質についてもよくわかっていない。そこで本課題では、地震波形に含まれる情報を余すことなく活用することで詳細な構造推定を可能にする波形インバージョン手法と、スペクトル要素法により理論波形を計算するソフトを使用して、アフリカ LLSVP の西側境界域にあたる南大西洋下のマントル最下部の地震波異方性構造を推定する。そしてその結果から、マントル最下部での物質の流動に関する制約を得ることを目指す。

課題名	地震波形インバージョンによるマントル最下部の S・P 波速度構造同時推定—地球深部の熱・化学進化の理解に向けて—
代表者名（所属）	佐藤 嶺（東京大学 理学系研究科）
利用システム名	Oakbridge-CX
実施期間	一年

地球の核-マントル境界 (CMB) は、固体岩石のマントルと液体鉄合金の外核が接する地球内部の最も主要な熱・化学組成境界であるため、CMB 直上数 100km(D″ 領域) は地球の熱・化学進化の理解に重要な領域である。D″ 領域では地温勾配とマントル組成のソリダスが近いために部分溶融に伴う化学分化を起こすマグマが定常的に発生する可能性が高く、また典型的な沈み込み領域下においては海洋プレートの沈み込みに伴う化学組成不均質が形成されると考えられている。従って観測情報から化学組成異常のサイズと度合いを制約することが重要である。しかし、地震波速度不均質を温度・化学組成効果に定量的に分離するには S・P 波速度構造を同程度の解像度で同時に推定する必要があったが、データセットの質や種類が異なる等課題があった。また正確な理論地震波形の計算も必要であるが、短周期までの計算は計算資源の問題から困難であった。本申請研究では地震波形に含まれる情報を余すことなく活用できる波形インバージョン手法と、スペクトル要素法による理論地震波形計算ソフトウェア SPECFEM3D_GLOBE を用いて D″ 領域の S・P 波速度構造を同時推定する。

課題名	臨界レイノルズ数付近における矩形ダクト乱流中の二次流れと熱的制御
代表者名（所属）	関本 敦（岡山大学 学術研究院）
利用システム名	Wisteria/BDEC-01 Odyssey
実施期間	一年
矩形ダクトなどの角を有する乱流中は、角へと向かう主流に垂直方向の平均二次流れが生じる。この二次流れによって、主流の平均的な分布だけでなく乱流熱伝達率に影響を及ぼすことが知られている。 臨界レイノルズ数付近では、乱流渦と管路幅が同程度のスケールになるため、乱流渦の出現位置が側壁によって拘束される。さらに、壁面加熱して、浮力効果を加えることで、乱流渦の挙動をある程度コントロールできると考えられる。 本研究では、矩形ダクト乱流の直接数値シミュレーションを行い、浮力効果が乱流効果と同程度となるような浮力パラメターにおいて、二次流れの安定化効果や乱流パフへの影響を調べ、壁面加熱による乱流制御の新たな指針へとつなげる。	

課題名	Inversion modeling of aquifer deformation for permeability estimation using Automatic Differential and adjoint methods
代表者名（所属）	張 毅（地球環境産業技術研究機構）
利用システム名	Oakbridge-CX, Wisteria/BDEC-01 Aquarius
実施期間	一年
Recently, distributed fiber-optic strain sensing tool has been successfully deployed for the acquisition of aquifer strain along the optical fiber-installed well. To estimate aquifer permeability using the strain array data, inversely solving the coupled fluid flow and mechanical (poroelastic) equations is required. In this study, we aim to develop an Automatic Differential (AD) and adjoint state combination method to accelerate the Jacobian matrix calculation and the inverse modeling	

以上

7大学スーパーコンピュータ比較表

2022年4月現在

大学名	北海道大学	東北大学	東京大学	名古屋大学	京都大学	大阪大学	九州大学
中央処理装置 (理論演算性能, メモリーサイズ)	PRIMERGY CX2550 M4,PRIMERGY CX400 M4(3.084PFLOPS,376.5TB) PRIMERGY CX1640 M1, PRIMERGY CX600 M1(0.877PFLOPS,31.5TB)	NEC SX-Aurora TSUBASA (1.48PFLOPS, 45TB) NEC LX 406Rz-2 (278TFLOPS, 17TB)	FUJITSU Oakbridge-CX (6.61PFLOPS, 256.5TiB) FUJITSU Wisteria/BDEC-01 (Odyssey, Wisteria-O) (25.9PFLOPS, 240.0TiB) FUJITSU Wisteria/BDEC-01 (Aquarius, Wisteria-A) (7.2 PFLOPS, 36.5TiB)	FUJITSU FX1000 (7.782PFLOPS, 72TiB) FUJITSU CX2570M5 (7.489PFLOPS, 82.875TiB) HPE Superdome Flex (77.414TFLOPS, 48TiB) HPE ProLiant DL560 (537.6TFLOPS, 37.5TiB)	CRAY XC40 (5.48PFLOPS, 196.9TB) Cray CS400 2820XT (1.03PFLOPS, 106.3TB) Cray CS400 4840X (42.4TFLOPS, 48.0TB)	NEC OCTOPUS (1,463TFLOPS, 72.9TB) NEC SQUID (16,591TFLOPS, 415.2TB)	ITO サブシステムA (6.91PFLOPS, 384TB) ITO サブシステムB (3.05PFLOPS, 49TB) ITO 基本フロントエンド (0.42PFLOPS, 61TB) ITO 大容量フロントエンド (49.6TFLOPS, 48TB)
総理論演算性能	3.96PFLOPS	1.76PFLOPS	39.71PFLOPS	15.886PFLOPS	6.55PFLOPS	18.054PFLOPS	10.43PFLOPS
主メモリーサイズ計	408TB	62TB	533TiB	240TiB	351.2TB	488.1TB	542TB
ディスク容量	16PB	2PB	38.2PB	30.44PB	24PB	24.1PB	24.6PB
2021年度 年間借料	約 7.6億円	約4.0億円	約25.3億円	約9.0億円	約 8.5億円	約 10.6億円	約 11.1億円

７大学スーパーコンピュータ利用負担金表１（北海道大学）

2022年4月現在

大学名	北海道大学	
中央処理装置（メモリーサイズ）	**スーパーコンピュータ** PRIMEHGY CX2550 M4,PRIMEHGY CX400 M4(376.5TB) PRIMEHGY CX1640 M1,PRIMEHGY CX600 M1(31.5TB) インタークラウド PRIMEHGY CX400/CX2550 M4 (256GB×16) PRIMEHGY CX400/CX2550 M4 (256GB×51) PRIMEHGY 2540 M4 (256GB×4)	
基本サービス	大型計算機システム利用申請時の利用者登録において １件につき ただし、学生において １件につき	年額 12,960円 年額 2,160円
付加サービス　スーパーコンピュータシステム	スーパーコンピュータ利用によるバッチ処理において 共用ノード（演算） サブシステムA A3コース　：演算時間 3,000,000秒まで A15コース　：演算時間 15,000,000秒まで A100コース：演算時間100,000,000秒まで A250コース：演算時間250,000,000秒まで サブシステムB B3コース　：演算時間 3,000,000秒まで B15コース　：演算時間 15,000,000秒まで B100コース：演算時間100,000,000秒まで B250コース：演算時間250,000,000秒まで 占有ノード（演算およびスパコンストレージ） サブシステムA １ノード（３ＴＢのwork領域を含む）につき サブシステムB １ノード（３ＴＢのwork領域を含む）につき スパコンストレージ home領域　１ＴＢにつき work領域　３ＴＢにつき （いずれも年度内利用に限る）	 年額 24,000円 年額 81,000円 年額 405,000円 年額 810,000円 年額 19,500円 年額 66,000円 年額 330,000円 年額 660,000円 年額 93,000円 年額 78,000円 年額 20,000円 年額 30,000円
クラウドシステム	クラウドサーバ利用において 仮想サーバ　１単位につき （ただし，最小２単位とする） 物理サーバ　１台につき ＧＰＵサーバ　１台につき 追加ストレージ１ＴＢにつき	月額 700円 年額 8,400円 月額 14,000円 年額 168,000円 月額 20,000円 年額 240,000円 月額 500円 年額 6,000円
	インタークラウドパッケージ利用において ３拠点　１単位につき ４拠点　１単位につき	月額 42,000円 年額 504,000円 月額 56,000円 年額 672,000円
	移行用サーバ利用において ホスティングサーバ Ｓサーバ　１台につき Ｍサーバ　１台につき Ｌサーバ　１台につき Ｓサーバ，Ｍサーバ，Ｌサーバ 追加ストレージ　１ＴＢにつき	月額 2,268円 年額 27,216円 月額 1,026円 年額 12,312円 月額 4,104円 年額 49,248円 月額 10,260円 年額 123,120円 月額 1,890円 年額 22,680円
	クラウドストレージ利用において １ＴＢにつき （いずれも年度内利用に限る）	月額 500円 年額 6,000円
出力	大判カラープリンタ利用において 普通紙１枚につき 光沢紙１枚につき クロス１枚につき	 432円 1,188円 3,996円
備考	１　一般利用コースにおいて利用できる大型計算機システムは，スーパーコンピュータ，アプリケーションサーバ，スパコンストレージ，クラウドサーバ（仮想サーバ，物理サーバ，ＧＰＵサーバ及び移行用サーバ），クラウドストレージ及び大判プリンタとする。 ２　基本サービスにおいては，以下のサービスを利用することができる。 スーパーコンピュータサービス １）試用・デバッグ用の共用ノードの利用 ２）アプリケーションサーバの利用 ３）home領域（スパコンストレージ） クラウドサービス １）クラウドストレージ ３　基本サービスに係る経費の負担において，学生の適用となるものは学校教育法における「学生」とし，「学生証」等しの提出をもって適用とする。その他の適用については，センター長が特に認めたものとする。 ４　スーパーコンピュータの共用ノード利用において，演算時間の算出方法は利用ノード数に経過時間（秒）を乗じて計算するものとする。	

北海道大学つづき

	５　スーパーコンピュータの占有ノード利用は，申請後の承認日から年度末までの利用とする。また，承認日に応じて次のとおりの利用負担金とする。 １）７月～９月の場合，利用負担金額の７５％とする。 ２）１０月～１２月の場合，利用負担金額の５０％とする。 ３）翌年１月～３月の場合，利用負担金額の２５％とする。 ６　スーパーコンピュータシステムの付加サービス（占有ノード等）は，研究グループでの共同利用ができるものとする。研究グループの構成メンバーは，大型計算機システムの利用者の中から，付加サービスの申請者が指定する。 ７　クラウドサーバにおけるサーバ構成は以下のとおりである。 １）仮想サーバ　１単位あたり コア数１，メモリ６ＧＢ，ストレージ５０ＧＢ ２）物理サーバ　１台あたり コア数４０，メモリ２５６ＧＢ，ストレージ２ＴＢ ３）ＧＰＵサーバ　１台あたり コア数４０，メモリ２５６ＧＢ，ストレージ２ＴＢ，ＧＰＵ　２基（Tesla V100） ８　移行用サーバにおけるサーバ構成は以下のとおりである。 １）ホスティングサーバ　１台あたり コア数１（ＨＡ機能），メモリ３ＧＢ，ストレージ１００ＧＢ ２）Ｓサーバ　１台あたり コア数１，メモリ３ＧＢ，ストレージ１００ＧＢ ３）Ｍサーバ　１台あたり コア数４，メモリ１２ＧＢ，ストレージ１００ＧＢ ４）Ｌサーバ　１台あたり コア数１０，メモリ３０ＧＢ，ストレージ１００ＧＢ ９　インタークラウドパッケージについては，一般財団法人高度情報科学技術研究機構（RIST）が公募するHPCIシステム共用計算資源の利用研究課題及び学校教育法施行規則に基づき認定された学際大規模情報基盤共同利用・共同研究拠点（JHPCN）が公募する共同研究課題の採択者に限って利用するものとする。ただし，センター長が適当と認めたときは，採択者以外もその利用を妨げない限度において利用することができる。 10　インタークラウドパッケージにおける物理サーバと拠点については以下のとおりである。 １）３拠点：北海道大学，東京大学，大阪大学において各１サーバ ２）４拠点：北海道大学，東京大学，大阪大学，九州大学において各１サーバ ※　本コースの他，民間企業等利用コースがある。
施行（運用）年月日	2018. 12. 1

7大学スーパーコンピュータ利用負担金表2（東北大学・東京大学）

2023年4月現在

大学名	東北大学		
中央処理装置（メモリーサイズ）	NEC SX-Aurora TSUBASA(45TB) NEC LX 406Rz-2(17TB)		
【大学・学術利用】			
演算負担経費 スーパーコンピュータ	共有（無料）	利用VE数1(実行数、実行時間の制限有)	無料
	共有（従量）	課金対象時間 ＝（利用VE数÷8 を切上げた数）× 経過時間(秒) 課金対象時間1時間につき	125円
	共有（定額）	負担額10万円につき課金対象時間800時間分使用可能	
	占有	利用VE数8　利用期間3ヶ月につき	270,000円
演算負担経費 並列コンピュータ	共有（従量）	課金対象時間 ＝ 利用ノード数× 経過時間(秒) 課金対象時間1時間につき	22円
	共有（定額）	負担額10万円につき課金対象時間4,600時間分使用可能	
	占有	利用ノード数1　利用期間3ヶ月につき	47,000円
ファイル負担経費	共有	5TBまで無料、追加容量1TBにつき年額	3,000円
	占有	10TBまで無料、追加容量1TBにつき年額	3,000円
出力負担経費	大判プリンタカラープリント　フォト光沢用紙1枚につき		600円
		クロス1枚につき	1,200円
【民間機関利用　（成果公開型）】			
演算負担経費 スーパーコンピュータ	共有（無料）	利用VE数1(実行数、実行時間の制限有)	無料
	共有（従量）	課金対象時間 ＝（利用VE数÷8 を切上げた数）× 経過時間(秒) 課金対象時間1時間につき	250円
	共有（定額）	負担額20万円につき課金対象時間800時間分使用可能	
	占有	利用VE数8　利用期間3ヶ月につき	540,000円
演算負担経費 並列コンピュータ	共有（従量）	課金対象時間 ＝ 利用ノード数× 経過時間(秒) 課金対象時間1時間につき	44円
	共有（定額）	負担額20万円につき課金対象時間4,600時間分使用可能	
	占有	利用ノード数1　利用期間3ヶ月につき	94,000円
ファイル負担経費	共有	5TBまで無料、追加容量1TBにつき年額	6,000円
	占有	10TBまで無料、追加容量1TBにつき年額	6,000円
出力負担経費	大判プリンタカラープリント　フォト光沢用紙1枚につき		1,300円
		クロス1枚につき	2,400円
【民間機関利用　（成果非公開型）】			
演算負担経費 スーパーコンピュータ	共有（無料）	利用VE数1(実行数、実行時間の制限有)	無料
	共有（従量）	課金対象時間 ＝（利用VE数÷8 を切上げた数）× 経過時間(秒) 課金対象時間1時間につき	500円
	共有（定額）	負担額40万円につき課金対象時間800時間分使用可能	
	占有	利用VE数8　利用期間3ヶ月につき	1,080,000円
演算負担経費 並列コンピュータ	共有（従量）	課金対象時間 ＝ 利用ノード数× 経過時間(秒) 課金対象時間1時間につき	88円
	共有（定額）	負担額40万円につき課金対象時間4,600時間分使用可能	
	占有	利用ノード数1　利用期間3ヶ月につき	188,000円
ファイル負担経費	共有	5TBまで無料、追加容量1TBにつき年額	12,000円
	占有	10TBまで無料、追加容量1TBにつき年額	12,000円
出力負担経費	大判プリンタカラープリント　フォト光沢用紙1枚につき		2,400円
		クロス1枚につき	4,800円
備考	1　負担額が無料となるのは専用のキューで実行されたものとし、制限時間を超えた場合は強制終了する。 2　演算負担経費の課金対象時間については半期毎(4月から9月及び10月から3月)に合計し、1時間未満を切上げて負担金を請求する。 3　演算負担経費について定額制を選択した場合はスーパーコンピュータ及び並列コンピュータを課金対象時間の範囲内で共用できる。 4　占有利用期間は年度を超えないものとし、期間中に障害、メンテナンス作業が発生した場合においても、原則利用期間の延長はしない。 5　ファイル負担経費については申請日から当該年度末までの料金とする。運用期間が1年に満たない場合は、月割りをもって計算した額とする。		
施行（運用）年月日	2020.10.1		

2022年4月現在

大学名	東京大学			
中央処理装置（メモリーサイズ）	FUJITSU PRIMERGY CX400 M1/CX2550 M5/CX2560 M5(192 GiB/ノード×1,368; Oakbridge-CX) FUJITSU PRIMEHPC FX1000 (32GiB/ノード × 7,680; Wisteria/BDEC-01 Odyssey) FUJITSU PRIMERGY GX2570 M6 ((512GiB+GPU40GiB×8)/ノード × 45 ; Wisteria/BDEC-01 Aquarius)			
Oakbridge-CX スーパーコンピュータシステム				
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（基本セット）	申込1セット当り 100,000 円 (8,640 トークン)	申込1セット当り 120,000 円 (8,640 トークン)	申込1セット当り /work 4 TB 利用者当り /home 50 GB	利用期間12か月の金額・トークン量（利用期間は1ヶ月単位で設定 可）
ノード固定	申込1セット当り 150,000 円 (8,640 トークン)	申込1セット当り 180,000 円 (8,640 トークン)		
一般申込（最小セット）	8,400 円 (720 トークン)		/work 4 TB 利用者当り /home 50 GB	利用期間は当該年度末まで
トークン追加	8,400 円 (720 トークン)	10,000 円 (720 トークン)		
ディスク追加	6,480円／(1 TB＊12か月)			1 TB単位で申込 可 (/workのみ)
備考	※ トークン消費係数は 1.00, ただしトークン消費係数 1.50 のノード群(優先利用向け)を全体の15%程度設ける ※ 1ジョブで利用可能な最大ノード数は256 ノード 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 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，トークン残量が無くなった場合には，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 ・Oakbridge-CX スーパーコンピュータシステムでは，スーパーコンピュータを無償，または通常利用に比べて安価な利用負担金で，一定期間試用できる制度を実施している。負担金等は以下を参照。			
	基本負担金		ディスク量	備考
	大学・公共機関等	企業		
お試しスパコン利用（無料体験）	無償 ※ Oakbridge-CX に一度もアカウント登録されていない場合のみ申込可能		/work 4 TB 利用者当り /home 50 GB	利用期間は利用開始日から1か月後まで 最大ノード数 8 ノード
	基本負担金		ディスク量	備考
	大学・公共機関等	企業		
通常利用（トライアル）/ 企業利用（トライアル）	申込1セット 付与トークン12か月 30,000 円 (8,640 トークン) ※ 最大1セット ※ 最大 12 か月分	申込1セット 付与トークン12か月 27,000 円 (8,640 トークン) ※ 最大 4 セット ※ 最大 12 か月分、3か月分は無償	申込1セット当り /work 4 TB 利用者当り /home 50 GB	利用期間は当該年度末まで 最大ノード数 256 ノード
	※ トークン消費係数は 1.00, ただしトークン消費係数 1.50 のノード群(優先利用向け)全体の1%程度設ける			
施行（運用）年月日	2022.4.1			

７大学スーパーコンピュータ利用負担金表３（東京大学・名古屋大学）

2022年4月現在

東京大学つづき

Wisteria/BDEC-01スーパーコンピュータシステム

コース	負担金額（税込） 大学・公共機関等	負担金額（税込） 企業	ディスク容量	備考
一般申込（基本セット） Wisteria-O/A	申込1セット当り 60,000 円 (8,640 トークン)		申込1セット当り /work 2 TB 利用者当り /home 50 GB	利用期間12か月の金額・トークン量（利用期間は1ヶ月単位で設定可）
公募制度による申込 Wisteria-O	申込1セット当り 60,000 円 (8,640 トークン)	申込1セット当り 72,000 円 (8,640 トークン)	申込1セット当り /work 2 TB 利用者当り /home 50 GB	
公募制度による申込 Wisteria-A	申込1セット当り 180,000 円 (25,920 トークン)	申込1セット当り 216,000 円 (25,920 トークン)	申込1セット当り /work 6 TB 利用者当り /home 50 GB	
GPU専有申込（公募制度の申込可） Wisteria-A	申込1GPUセット当り 270,000 円 (25,920 トークン)	申込1GPUセット当り 324,000 円 (25,920 トークン)	申込1GPUセット当り /work 6 TB 利用者当り /home 50 GB	利用期間12か月の金額・トークン量（利用期間は1ヶ月単位で設定可） 1, 2, 4GPUのみ申込可, 申込単位は下表参照
ノード固定（公募制度の申込可） Wisteria-A	2,160,000 円 (207,360 トークン)	2,160,000 円 (207,360 トークン)	/work 48 TB 利用者当り /home 50 GB	利用期間12か月の金額・トークン量（利用期間は1ヶ月単位で設定可） 1セットのみ申込可
一般申込（最小セット） Wisteria-O/A	5,000 円 (720 トークン)		/work 2 TB 利用者当り /home 50 GB	利用期間は当該年度末まで
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／(1 TB × 12か月)			1 TB単位で申込可 (/workのみ)

※ 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 (1 GPU 当り)
Wisteria-O にはトークン消費係数 1.50 のノード群(優先利用向け)を全体の15 %程度設ける
※ Wisteria-O の1ジョブで利用可能な最大ノード数は2,304 ノード
Wisteria-A の1ジョブで利用可能な最大GPU数は64 GPU
※ GPU専有申込の申込単位

GPU	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

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 Odysseyを使用した場合，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
5 Aquariusを使用した場合，トークンはジョブ実行ごとにGPU時間積（経過時間×GPU数）にトークン消費係数を乗じた値を消費する
6 トークンの残量が無くなった場合には，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
7 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
8 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
9 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
10 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
11 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
12 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
13 追加オプションは利用期間内に限り有効とする
14 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
15 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

・Wisteria/BDEC-01 スーパーコンピュータシステムでは，スーパーコンピュータを無償，または通常利用に比べて安価な利用負担金で，一定期間試用できる制度を実施している。負担金等は以下を参照。

	基本負担金 大学・公共機関等	基本負担金 企業	ディスク量	備考
お試しスパコン利用（無料体験）	無償 ※ Wisteria/BDEC-01 に一度もアカウント登録されていない場合のみ申込可能		/work 2 TB 利用者当り /home 50 GB	利用期間は利用開始日から1か月後まで 最大ノード数/GPU数 Wisteria-O: 12 ノード Wisteria-A: 4 GPU

	基本負担金 大学・公共機関等	基本負担金 企業	ディスク量	備考
通常利用（トライアル）/企業利用（トライアル）	申込1セット 付与トークン12か月 18,000 円 (8,640 トークン) ※ 最大1セット ※ 最大 12 か月分	申込1セット 付与トークン12か月 16,200 円 (8,640 トークン) ※ 最大 4 セット ※ 最大 12 か月 分、30 月 分は無償	申込1セット当り /work 2 TB 利用者当り /home 50 GB	利用期間は当該年度末まで 最大ノード数/GPU数 Wisteria-O: 2,304 ノード Wisteria-A: 64 GPU

※ 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 (1 GPU当り)
Wisteria-O にはトークン消費係数 1.50 のノード群(優先利用向け)を全体の15 %程度設ける
※ 企業利用（トライアル）はノード群 (Wisteria-O, Wisteria-A) ごとの申込

施行（運用）年月日	2022.4.1

大学名	名古屋大学
中央処理装置（メモリーサイズ）	FUJITSU Type I サブシステム FX1000 （ 335 iB× 2,304） Type II サブシステム CX2570 M5 （384 iB× 221） HPE Type II サブシステム HPE Superdome Flex （ 24 TiB× 2） クラウドシステム HPE ProLiant DL560 （384 iB× 100）
負担金の区分	利用負担金額
基本負担金	利用登録 1 件につき年額 10,000 円 この場合において，基本負担金額10,000 円を1 円当たり 1 ポイントに換算したポイント数を取得するものとする。
追加負担金	経費の見込等に基づく任意の金額（1,000円単位） この場合において，追加負担金が500,000 円未満の場合は1 円当たり1ポイントに，500,000 円以上の場合は1 円当たり1.24 ポイントに換算したポイント数を取得するものとする。
備考	1 基本負担金を負担することにより，演算負担ポイント及びファイル使用負担ポイントの合計が1 円当たり 1 ポイントに換算した場合に10,000 ポイントに達するまで，利用を可能とする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 2 基本負担金を負担し，かつ，1 回当たり次表の左欄に掲げる追加負担金を負担するときは，追加負担金について利用した経費がそれぞれ右欄に掲げる換算方法に基づく追加利用可能ポイント数に達するまで利用できるものとする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。

1 回の追加負担金の入金額	追加利用可能ポイント数
500,000 円未満	追加負担金額を1 円当たり1 ポイントに換算したポイント数
500,000 円以上	追加負担金額を1 円当たり1.25 ポイントに換算したポイント数

グループ利用負担金	基本負担金：利用登録10 件につき 年額100,000 円 この場合において，基本負担金を1 円当たり1.0 ポイントに換算したポイント数を取得するものとする。 追加負担金：経費の見込等に基づく任意の金額（一口1,000 円以上） この場合において，利用可能ポイトへの換算は，別表1-1 の追加負担金規程に準ずることとする。 1 グループ利用負担金の申込みは，研究グループ単位で行うものとする。 2 基本負担金（100, 000 円）を負担することにより，別表2 の演算負担ポイント及びファイル使用負担ポイントの合計が，1円当たり1.0 ポイントに換算した場合に100, 000 ポイントに達するまで，利用を可能とする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 3 追加負担金を負担するときは，一口1,000 円以上とし，利用可能ポイントへの換算は，別表1-1 の追加負担金規程に準ずることとする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 4 利用登録において10 件未満となる利用登録件数については，当該登録件数を10件に切り上げるものとする。
リテラシー利用負担金	基本負担金：利用登録25 件につき、一の年度における6箇月の金額10,000円 この場合において，基本負担金を1 円当たり0.5 ポイントに換算したポイント数を取得するものとする。 追加負担金：経費の見込み等に基づく任意の金額（10,000 円単位） この場合において，追加負担金を1 円当たり0.5 ポイントに換算したポイント数を取得するものとする。 1 リテラシー用とは，スーパーコンピュータを利用した研究を目的とした実習を行うものをいう。 2 基本負担金（10,000円）を負担することにより，別表2の演算負担ポイント及びファイル使用負担ポイントの合計を1円当たり5ポイントに換算した場合に50,000ポイントに達するまで，利用を可能とする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 3 基本負担金を負担し，かつ，追加負担金10,000円を負担するときは，追加負担金について利用した経費を1円当たり0.5ポイントに換算した場合に50,000ポイントに達するまで利用できるものとする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 4 利用登録において25 件未満となる利用登録件数については，当該登録件数を25 件に切り上げるものとする。
民間利用負担金	基本負担金：利用登録10 件につき年額 200, 000 円 この場合において，基本負担金を200,000 円当たり100, 000 ポイントに換算したポイント数を取得するものとする。 追加負担金：経費の見込み等に基づく任意の金額（200, 000 円単位） この場合において，追加負担金を200,000 円当たり100, 000 ポイントに換算したポイント数を取得するものとする。 1 民間利用とは，利用規程第3 条第1 項第7号の規定に基づきスーパーコンピュータを利用し，その成果を公表する場合のこと 2 基本負担金（200, 000円）を負担することにより，別表2の演算負担ポイント及びファイル使用負担ポイントの合計が200, 000円当たり100, 000ポイントに換算した場合に100, 000ポイントに達するまで，利用を可能とする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 3 基本負担金を負担し，かつ，追加負担金200, 000円を負担するときは，追加負担金について利用した経費を200, 000円当たり100, 000ポイントに換算した場合に100, 000ポイントに達するまで利用できるものとする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 4 利用登録において10 件未満となる利用登録件数については，当該登録件数を10 件に切り上げるものとする。
成果非公開型民間利用負担金	基本負担金：利用登録10 件につき年額 400, 000 円 この場合において，基本負担金を400,000 円当たり100, 000 ポイントに換算したポイント数を取得するものとする。 追加負担金：経費の見込み等に基づく任意の金額（400, 000 円単位） この場合において，追加負担金を400,000 円当たり100, 000 ポイントに換算したポイント数を取得するものとする。 1 成果非公開型民間利用とは，利用規程第3 条第1 項第7 号の規定に基づきスーパーコンピュータを利用し，その成果を非公開とする場合のことをいう。 2 基本負担金（400, 000円）を負担することにより，別表2 の演算負担ポイント及びファイル使用負担ポイントの合計を400, 000円当たり100, 000ポイントに換算した場合に100, 000ポイントに達するまで，利用を可能とする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 3 基本負担金を負担し，かつ，追加負担金400, 000円を負担するときは，追加負担金について利用した経費を400, 000円当たり100, 000ポイントに換算した場合に100, 000ポイントに達するまで利用できるものとする。ただし，取得したポイントの有効期限は，当該ポイントを取得した年度内に限るものとする。 4 利用登録において10 件未満となる利用登録件数については，当該登録件数を10 件に切り上げるものとする。

７大学スーパーコンピュータ利用負担金表４（名古屋大学・京都大学）

名古屋大学つづき

ノード専占有利用負担金	TypeII サブシステム：１ノードにつき 一ヶ月当たり210,000 円 クラウドシステム　：１ノードにつき 一ヶ月当たり 62,000 円 1 ノード占有利用とは，TypeII サブシステムおよびクラウドシステムにおける一部のノードを優先的に占有して利用する場合のことをいう。 2 の場合におけるファイル使用負担ポイントは，別表1-1 で定める基本負担金及び追加負担金の合計額とし，算定方法は，別表2 に準ずる。 3 準占有できるノード数は，1 ノード単位とし，期間は一ヶ月単位とする。 4 ノード準占有利用は，登録時の利用期間及び年度を越えてできない。 5 ノード準占有は，学術研究の利用のみ利用可とする。
コールドストレージ利用負担金	ファイル負担金：　10 カートリッジを１口（50TB）として初年度190,000円とし，次年度以降継続する場合は，ファイル負担金は徴収しない。 ファイル管理金：　10 スロット１口（50TB）としてファイル管理金を初年度は10,000 円，次年度以降継続された場合年間10,000 円を徴収する。 1 ファイル負担金とは，本センターのコールドストレージシステムを利用する時，センター設置の光ディスクを使用する場合に発生する負担金である。 2 ファイル管理金とは，本センターに設置されているコールドストレージシステムを利用した時，管理運用経費として徴収するものである。
ポイントの区分	ポイントの算定基礎額
演算負担ポイント　バッチ処理	TypeI サブシステム：経過時間１秒につき0.0056 ポイントに使用ノード数を乗じて得たポイント数。 TypeIIサブシステム：経過時間１秒につき0.007 ポイントに使用GPU 数を乗じて得たポイント数。 TypeIII サブシステムおよびクラウドシステム：経過時間１秒につき0.002 ポイントに使用CPU ソケット数を乗じて得たポイント数。 ただし，クラウドシステムの予約システムでの利用の場合は，経過時間1 秒につき0.0001 ポイントに使用CPU コア数をそれぞれのサブシステムで，優先ジョブクラスを使用した場合は，得た負担ポイントに2 を乗じて得たポイントとする。乗じて得たポイント数。
演算負担ポイント　会話型処理	TypeIII サブシステム：CPU コアの演算時間１秒につき0.002 ポイント
ファイル使用負担ポイント	ホットストレージ：　ファイルの使用容量が1TB 以下の場合徴収しない。 ファイルの使用量が1TB を超えた場合超えた部分について１GB につき １日当たり 0.01 ポイント
備考	1 スーパーコンピュータのバッチ処理における演算負担ポイントは，当該ジョブの経過時間及び使用ノード数（ノード共有を含む。）GPU 数，CPU ソケット数又はCPU コア数により算定する。 2 TypeIIIサブシステムの会話型処理における演算負担ポイントは，１日単位で，かつ，システムごとに登録番号により使用したCPUコアの演算時間を集計し，算定する。 3 ホットストレージのファイル使用負担ポイントは，1 日単位で，かつ，登録番号ごとに使用量を集計し，算定する。 4 各区分の利用負担をポイントに換算した場合に1 ポイント未満の端数が生じるときは，これを切り上げる。
施行（運用）年月日	2021.4.1

2022年4月現在

大学名	京都大学
中央処理装置（メモリーサイズ）	CRAY　XC40（96GB+16GB×1800） CRAY　CS400 2820XT（128GB×850） CRAY　CS400 4840X（3TB×16）

区分	区分			利用負担額	提供サービス						
	コース	タイプ	セット		システム	パッチ	システム資源	経過時間（時間）	ストレージ（TB）	無料利用者数
スーパーコンピュータシステム	エントリ	—	基本	12,600 円/年	B	共有	最大1ノード相当 ((36コア、128GBメモリ)×1)	1	0.2	—
	パーソナル	タイプA	基本	100,000 円/年	A	共有	最大4ノード相当 ((68コア、16+96GBメモリ)×4)	168	3.0	—
		タイプB	基本	100,000 円/年	B	共有	最大4ノード相当 ((36コア、128GBメモリ)×4)		3.0	—
		タイプC	基本	100,000 円/年	C	共有	最大1ノード相当 ((72コア、3072GBメモリ)×1)		3.0	—
	グループ	タイプA1	最小	200,000 円/年	A	優先	4ノード((68コア、16+96GBメモリ)×4)	336	24.0	8
			追加単位	100,000 円/年			2ノード((68コア、16+96GBメモリ)×2)		12.0	4
		タイプA2	最小	240,000 円/年		準優先	8ノード((68コア、16+96GBメモリ)×8)		28.8	16
			追加単位	60,000 円/年			2ノード((68コア、16+96GBメモリ)×2)		7.2	4
		タイプA3	最小	600,000 円/年		占有	8ノード((68コア、16+96GBメモリ)×8)		48.0	16
			追加単位	300,000 円/年			4ノード((68コア、16+96GBメモリ)×4)		24.0	8
		タイプB1	最小	210,000 円/年	B	優先	4ノード((36コア、128GBメモリ)×4)		24.0	8
			追加単位	105,000 円/年			2ノード((36コア、128GBメモリ)×2)		12.0	4
		タイプB2	最小	252,000 円/年		準優先	8ノード((36コア、128GBメモリ)×8)		28.8	16
			追加単位	63,000 円/年			2ノード((36コア、128GBメモリ)×2)		7.2	4
		タイプB3	最小	630,000 円/年		占有	8ノード((36コア、128GBメモリ)×8)		48.0	16
			追加単位	315,000 円/年			4ノード((36コア、128GBメモリ)×4)		24.0	8
		タイプC1	最小	130,000 円/年	C	優先	1ノード((72コア、3072GBメモリ)×1)		24.0	8
			追加単位	130,000 円/年			1ノード((72コア、3072GBメモリ)×1)		24.0	8
		タイプC2	最小	156,000 円/年		準優先	2ノード((72コア、3072GBメモリ)×2)		28.8	16
			追加単位	78,000 円/年			1ノード((72コア、3072GBメモリ)×1)		14.4	8
	大規模ジョブ	タイプA	最小	20,000 円/週（7日）	A	占有	8ノード((68コア、16+96GBメモリ)×8)	168	—	—
			追加単位	10,000 円/週（7日）			4ノード((68コア、16+96GBメモリ)×4)			
		タイプB	最小	21,000 円/週（7日）	B	占有	8ノード((36コア、128GBメモリ)×8)			
			追加単位	10,500 円/週（7日）			4ノード((36コア、128GBメモリ)×4)			
		タイプC	最小	13,000 円/週（7日）	C	占有	2ノード((72コア、3072GBメモリ)×2)			
			追加単位	6,500 円/週（7日）			1ノード((72コア、3072GBメモリ)×1)			
	専用クラスタ	—	最小	630,000 円/年	B	—	8ノード((36コア、128GBメモリ)×8)	—	48.0	16
			追加単位	315,000 円/年			4ノード((36コア、128GBメモリ)×4)		24.0	8
	ストレージ容量追加			10,000 円/年	ストレージ容量10TBの追加につき					
	ライセンスサービス			20,000 円/年	可視化ソフト(AVS,ENVI/IDL)およびプリポストウェアの1ライセンスにつき					

備考

1 利用負担額は，年度単位で算定している。また，総額表示である。パーソナルコース，グループコース又は専用クラスタコースを，年度途中から利用を開始する場合及び年度途中で利用を終了する場合の利用負担額は，上記表中の利用負担額を12で除した後，利用月数を乗じて算出するものとし，100円未満に端数が出た場合は，10円単位を四捨五入するものとする。
なお，月途中から利用を開始する場合及び月途中で利用を終了する場合は，それぞれ1月の利用とする。
2 大型計算機システムの全ての利用者は，上記表のサービスの他，次のサービスを受けることができる。
　1）大判プリンタサービス
　2）その他、大型計算機システムが提供するサービス、機器の利用
3 上記表の大規模ジョブコース、ストレージ容量追加、ライセンスサービスの申請には、スーパーコンピュータシステムの利用者であることが必要である。
4 「共有」　：当該カテゴリのユーザ間で一定の計算資源を共有するベストエフォートのスケジューリングを行う。
「準優先」：定常稼働状況において記載値（以上）の計算資源が確保されるように優先スケジューリングを行う。
　また稼働状況によらず記載値の1/4の計算資源が確保されることを保証する。
「優先」　：定常稼働状況において記載値（以上）の計算資源が確保されるように優先スケジューリングを行う。
　また稼働状況によらず記載値の1/2の計算資源が確保されることを保証する。
「占有」　：稼働状況によらず記載値の計算資源が確保されることを保証する。
5 ストレージ容量はバックアップ領域（最大で総容量の1/2）を含む。
6 グループコース及び専用クラスタコースの利用者番号は利用者あたり年額5,000円を負担することで追加できる。
7 機関・部局定額制度
他機関又は学内における部局（『国立大学法人京都大学の組織に関する規程』第3章第2節から第11節で定める組織をいう。）の組織が，その組織単位でグループコースサービスを利用申請する場合の利用負担額は，別表１に規定する額の1.5倍の額とする。
なお，利用負担額が年額150万円未満の場合は100人、年額150万円を超える場合は、150万円 毎に100人までの利用者を認める。
ストレージは，1.5倍の容量とする。
8 スパコン連携サービス
学術情報メディアセンターのスーパーコンピュータシステムと密な連携により，学内における部局の組織が計算サーバ等を設置する場合，下記の負担額を支払うものとする。

冷却方式	利用負担額	利用負担額算定単位
水冷	9,800 円/月	水冷冷却方式の計算サーバ等の定格電力 1kWにつき
空冷	11,500 円/月	空冷冷却方式の計算サーバ等の定格電力 1kWにつき

スーパーコンピュータシステム（民間機関利用）

システム	システム資源	経過時間（時間）	ストレージ（TB）	無料利用者数	利用負担額
A	8ノード((68コア、16+96GBメモリ)×8)	336	28.8	16	960,000 円/年
	12ノード((68コア、16+96GBメモリ)×12)	336	43.2	24	1,440,000 円/年
	16ノード((68コア、16+96GBメモリ)×16)	336	57.6	32	1,920,000 円/年
B	8ノード((36コア、128GBメモリ)×8)	336	28.8	16	1,008,000 円/年
	12ノード((36コア、128GBメモリ)×12)	336	43.2	24	1,512,000 円/年
	16ノード((36コア、128GBメモリ)×16)	336	57.6	32	2,016,000 円/年
C	2ノード((72コア、3072GBメモリ)×2)	336	28.8	16	624,000 円/年
	3ノード((72コア、3072GBメモリ)×3)	336	43.2	24	936,000 円/年
	4ノード((72コア、3072GBメモリ)×4)	336	57.6	32	1,248,000 円/年

7大学スーパーコンピュータ利用負担金表5（京都大学・大阪大学）

2022年4月現在

京都大学つづき

大学名	京都大学
備考	備考 1 利用負担額は、年度単位で算定している。また、総額表示である。パーソナルコース、グループコース又は専用クラスタコースを、年度途中から利用を開始する場合及び年度途中で利用を終了する場合の利用負担額は、上記表中の利用負担額を12で除した後、利用月数を乗じて算出するものとし、10円未満に端数が出た場合は、10円単位を四捨五入するものとする。なお、月途中から利用を開始する場合及び月途中で利用を終了する場合は、それぞれ1月の利用とする。 2 ストレージ容量はバックアップ領域（最大で総容量の1/2）を含む
施行（運用）年月日	2020.6.1

大学名	大阪大学
中央処理装置（メモリーサイズ）	NEC OCTOPUS（1,463 TFLOPS, 72.9 TB） NEC SQUID（16,591 TFLOPS, 415.2TB）

(1) OCTOPUSの負担額

(A) 占有

基本負担額	占有ノード数
191,000円／年	汎用CPUノード群 1ノード
783,000円／年	GPUノード群 1ノード
154,000円／年	Xeon Phiノード群 1ノード

(B) 共有

	基本負担額	OCTOPUSポイント
コース	10万円	1,000 ポイント
	50万円	5,250 ポイント
	100万円	11,000 ポイント
	300万円	34,500 ポイント
	500万円	60,000 ポイント

(C) ディスク容量追加

基本負担額	提供単位
2,000円／年	1TB

負担額：上記負担額で算出した合計額に消費税（10%）を加えて得た額

備考

1 負担額は上記負担額で算出した合計額に、消費税（10%）を加えて得た額とする。
ただし、産業利用 成果非公開型の負担額は、上記負担額で算出した合計額に5を乗じ、
消費税（10%）を加えて得た額とする。
2 登録時の利用期限または年度を越えて利用はできない。
3 ディスク容量は1申請単位で3TBを割り当てる。ただし、他のディスク容量と合算できない。
4 (A) は占有ノード数を追加する場合のみ変更申請を受け付ける。
5 (A) の2ノード以上の基本負担額は、1ノードを基準に比例するものとする。
6 (A) は資源提供状況により10ノード以上3か月単位の申請を受け付ける場合がある。
その場合の月額の負担額は、1ノード年の基本負担額の1/10とする。
7 (B) は年度の途中でコースの変更はできない。新たにコースを追加する場合は申請を受け付ける。
8 計算ノードの利用に使用するOCTOPUSポイントは、使用したノード時間に対して以下の消費係数および季節係数を乗じたものとする。季節係数は前年の利用状況等を鑑み、0を超える1以下の値を設定する。

ノード群	消費係数	季節係数
汎用CPUノード群	0.0520	大規模計算機システムWEBページに記載
GPUノード群	0.2173	
Xeon Phiノード群	0.0418	
大容量主記憶搭載ノード群	0.3703	

9 (C) は年度の途中は追加申請のみ受け付ける。
10 (C) は1つの申請グループにつき、500TBの追加を上限とする。

(2) SQUIDの負担額

(A) 占有

基本負担額	占有ノード数
575,000円／年	汎用CPUノード群 1ノード
3,516,000円／年	GPUノード群 1ノード
2,168,000円／年	ベクトルノード群 1ノード

(B) 共有

	基本負担額	OCTOPUSポイント
コース	10万円	1,000 ポイント
	50万円	5,250 ポイント
	100万円	11,000 ポイント
	300万円	34,500 ポイント
	500万円	60,000 ポイント

(C) ストレージ容量追加

基本負担額	提供単位
2,000円／年	HDD 0TB
5,000円／年	SSD 1TB

備考

1 負担額は上記負担額で算出した合計額に、消費税（10%）を加えて得た額とする。
ただし、産業利用 成果非公開型の負担額は、上記負担額で算出した合計額に5を乗じ、
消費税（10%）を加えて得た額とする。
2 登録時の利用期限または年度を越えて利用はできない。
3 ストレージ容量は1申請単位でSQUID 0TBを割り当てる。ただし、他のストレージ容量と合算できない。
4 (A) は占有ノード数を追加する場合のみ変更申請を受け付ける。
5 (A) の2ノード以上の基本負担額は、1ノードを基準に比例するものとする。
6 (A) は資源提供状況により3か月単位の申請を受け付ける場合がある。
その場合の月額の負担額は、1ノード年の基本負担額の1/10とする。
7 (B) は年度の途中でコースの変更はできない。新たにコースを追加する場合は申請を受け付ける。
8 計算ノードの利用に使用するSQUIDポイントは、使用したノード時間に対して以下の消費係数および季節係数を乗じたものとする。季節係数は前年の利用状況等を鑑み、0を超える1以下の値を設定する。

ノード群	消費係数			季節係数
	高優先度	通常優先度	シェア	
汎用CPUノード群	0.1873	0.1499	0.1124	大規模計算機システムWEBページに記載
GPUノード群	1.1467	0.9174	0.6881	
ベクトルノード群	0.707	0.5656	0.4242	

9 (C) は年度の途中は追加申請のみ受け付ける。
10 (C) は1つの申請グループにつき、HDD 500TB、SSD 10TBの追加を上限とする。

(3) ONION(オブジェクトストレージ)の負担額

基本負担額	提供単位
12,000円／年	1TB

備考：年度の途中は追加申請のみ受け付ける。

施行（運用）年月日：2021.8.1

７大学スーパーコンピュータ利用負担金表６（九州大学）

2022年4月現在

大学名		九州大学			
中央処理装置（メモリーサイズ）		ITO サブシステムA（192GB×2000）	ITO サブシステムB（384GB×128）	ITO 基本フロントエンド（384GB×160）	ITO 大容量フロントエンド（12TB×4）
一般利用	区分	4ノード 月額：3,000円 16ノード 月額：12,000円 64ノード 月額：48,000円 128ノード 月額：96,000円 256ノード 月額：192,000円	1ノード 月額：2,200円 4ノード 月額：8,800円	Sプラン［864コア時間積］ 月額：920円 Mプラン［3,456コア時間積］ 月額：3,680円 Lプラン［6,912コア時間積］ 月額：7,360円	Mプラン［4,224コア時間積］ 月額：5,280円 Lプラン［8,448コア時間積］ 月額：10,560円
	共有タイプ				
	ノード固定タイプ	4ノード 月額：24,000円 16ノード 月額：96,000円 64ノード 月額：384,000円	1ノード 月額：17,000円	1ノード 月額：7,400円	
	公募型プロジェクト	無料	無料	無料	無料
	ストレージ	10TB　月額：350円 100TB 月額：3,500円			
	備考	・上記の金額は消費税を含む。 ・１ヶ月未満の利用期間については、当該利用期間を１ヶ月とみなす。			
民間利用	区分	4ノード 月額：3,960円 16ノード 月額：15,840円 64ノード 月額：63,360円 128ノード 月額：126,720円 256ノード 月額：253,440円	1ノード 月額：2,900円 4ノード 月額：11,600円	Sプラン［864コア時間積］ 月額：1,220円 Mプラン［3,456コア時間積］ 月額：4,880円 Lプラン［6,912コア時間積］ 月額：9,760円	Mプラン［4,224コア時間積］ 月額：6,960円 Lプラン［8,448コア時間積］ 月額：13,920円
	共有タイプ				
	ストレージ	10TB　月額：460円 100TB 月額：4,600円			
	備考	・上記の金額は消費税を含む。 ・１ヶ月未満の利用期間については、当該利用期間を１ヶ月とみなす。			
施行（運用）年月日		2019.10.1			

スーパーコンピュータ利用による成果報告（2021 年）

利用者の皆様には、スーパーコンピュータシステムを利用して得られた研究成果（論文、口頭発表、著書、受賞情報）の登録にご協力いただき、誠にありがとうございます。今回はその中の 2021 年分（2021 年 1 月～2021 年 12 月）を掲載いたします。

研究成果の登録は、Web ページ（https://www.cc.u-tokyo.ac.jp/achievements/）にある「研究成果登録」から行うことができます。何卒ご協力のほどよろしくお願い申し上げます。

－Wisteria/BDEC-01－

● 口頭・ポスター発表

【材料工学】

1. [Odyssey] 大野直子、河村創憲: ODS 鋼に形成される複合酸化物の照射安定性 ～第一原理計算による酸化物の照射耐性評価～: 材料照射研究会「原子力イノベーションを見据えた材料研究」, 材料照射研究会 予稿集, O-22.

－Oakbridge-CX－

● 論文

【応用物理学】

2. Mizuki Tani, Tomohito Otobe, Yasushi Shinohara, and Kenichi L. Ishikawa: Semiclassical description of electron dynamics in extended systems under intense laser fields: Physical Review B, The American Physical Society, 104, 7.

【物理学】

3. 越智正之: 第一原理計算を用いた物質設計：その電子状態の次元性に注目して: 生産と技術, 生産技術振興協会, 73, 1.
4. Hiroshi Takatsu, Masayuki Ochi, Morito Namba, Haobo Li, Aurelien Daniel, Takahito Terashima, Kazuhiko Kuroki, and Hiroshi Kageyama: Strain-Assisted Topochemical Synthesis of La-Doped SrVO2H Films: Crystal Growth & Design, American Chemical Society, 21, 7.
5. Y. Yamashita, G. Lim, T. Maruyama, A. Chikamatsu, T. Hasegawa, H. Ogino, T. Ozawa, M. Wilde, K. Fukutani, T. Terashima, M. Ochi, K. Kuroki, H. Kitagawa, and M. Maesato: Heavy carrier doping by hydrogen in the spin-orbit coupled Mott insulator Sr2IrO4: Physical Review B, American Physical Society, 104, 4.
6. R. Mizuno, M. Ochi, and K. Kuroki: Development of an efficient impurity solver in dynamical mean field theory for multiband systems: Iterative perturbation theory combined with parquet equations: Physical Review B, American Physical Society, 104, 3.
7. K. Higashi, M. Ochi, Y. Nambu, T. Yamamoto, T. Murakami, N. Yamashina, C. Tassel, Y. Matsumoto, H. Takatsu, C. M. Brown, and H. Kageyama: Enhanced Magnetic Interaction by Face-Shared Hydride Anions in 6H-BaCrO2H: Inorganic Chemistry, ACS Publications, 60, 16.
8. K. Miyazaki, M. Ochi, T. Nishikubo, J. Suzuki, T. Saito, T. Kamiyama, K. Kuroki, T. Yamamoto, and M. Azuma: High-Pressure and High-Temperature Synthesis of Anion-Disordered Vanadium Perovskite Oxyhydrides: Inorganic Chemistry, American Chemical Society, 60, 20.

【地球惑星科学】

9. Nishizawa, S., T. Yamaura, and Y. Kajikawa: Influence of sub-mesoscale topography on daytime precipitation associated with thermally driven local circulations over a mountainous region: Journal of Atmospheric Science, American Meteorological Society, 78, 8.

【総合工学】

10. Sugaya, K., and Imamura, T.: Unsteady turbulent flow simulations on moving Cartesian grids using immersed boundary method and high-order scheme: Computers and Fluids, Elsevier, Vol. 231.
11. 張毅: Toward Retrieving Distributed Aquifer Hydraulic Parameters From Distributed Strain Sensing: Journal of Geophysical Research: Solid Earth, American Geophysical Union, 126.

● 口頭・ポスター発表

【物理学】

12. 越智正之: 複合アニオン化合物の物質機能の第一原理的研究: 日本物理学会第 76 回年次大会(2021年).
13. 越智正之: 複合アニオン化合物における理論物質設計の試み: 第 12 回複合アニオン ウェブセミナー.

【総合工学】

14. Keisuke Sugaya and Taro Imamura: Unsteady Flow Simulation using Immersed Boundary Method on Cartesian Grid with Moving Grid Technique: AIAA Scitech 2021 Forum.
15. 菅谷圭祐, 原惇, 今村太郎: 階層型直交格子と再帰的なフィッティングを用いた低速・高迎角条件における NASA-CRM 巡航形態の空力予測: 第 53 回流体力学講演会／第 39 回航空宇宙数値シミュレーション技術シンポジウム, 1A17.
16. 菅谷圭祐, 今村太郎: 埋め込み境界法と移動格子による Caradonna-Tung 回転翼の非定常乱流解析: 第 53 回流体力学講演会／第 39 回航空宇宙数値シミュレーション技術シンポジウム, 2E07.

－Oakforest-PACS－

● 論文

【計算基盤】

17. Hiroshi Murakami: Single-Precision Calculation of Iterative Refinement of Eigenpairs of a Real Symmetric-Definite Generalized Eigenproblem by Using a Filter Composed of a Single Resolvent: ICPS Proceedings of HPC Asia 2021 Companion "ISBN 978-1-4503-8303-5", ACM.

【物理学】

18. Noritaka Shimizu, Yusuke Tsunoda, Yutaka Utsuno, and Takaharu Otsuka: Variational approach with the superposition of the symmetry-restored quasiparticle vacua for nuclear shell-model calculations: Physical Review C, American Physical Society, 103.

【地球惑星科学】

19. Takeshi Nishimura, Kentaro Emoto, Hisashi Nakahara, Satoshi Miura, Mare Yamamoto, Shunsuke Sugimura, Ayumu Ishikawa and Tsunehisa Kimura: Source location of volcanic earthquakes and subsurface characterization using fiber-optic cable and distributed acoustic sensing system: Scientific Reports, Nature Research, 11.
20. Yohei Onuki, Sylvain Joubaud, Thierry Dauxois: Simulating turbulent mixing caused by local instability of internal gravity waves: Journal of Fluid Mechanics, Cambridge University Press, 915.

【土木工学】

21. Masahide Otsubo, Reiko Kuwano, Catherine O"Sullivan, Thomas Shire: Using geophysical data to quantify stress transmission in gap-graded granular materials: Géotechnique, Thomas Telford Ltd. .
22. Yang Li, Masahide Otsubo, Reiko Kuwano: DEM analysis on the stress wave response of spherical particle assemblies under triaxial compression: Computers and Geotechnics, Elsevier, Vol.133, 104043.

－Reedbush－

● 論文

【生物科学】

23. [Reedbush-H] Yushiro Endo, Tomohiro Koga, Hiroki Otaki, Kaori Furukawa, Atsushi Kawakami: Systemic lupus erythematosus overlapping dermatomyositis owing to a heterozygous TREX1 Asp130Asn missense mutation: Clinical Immunology, Elsevier, 227.

【内科系臨床医学】

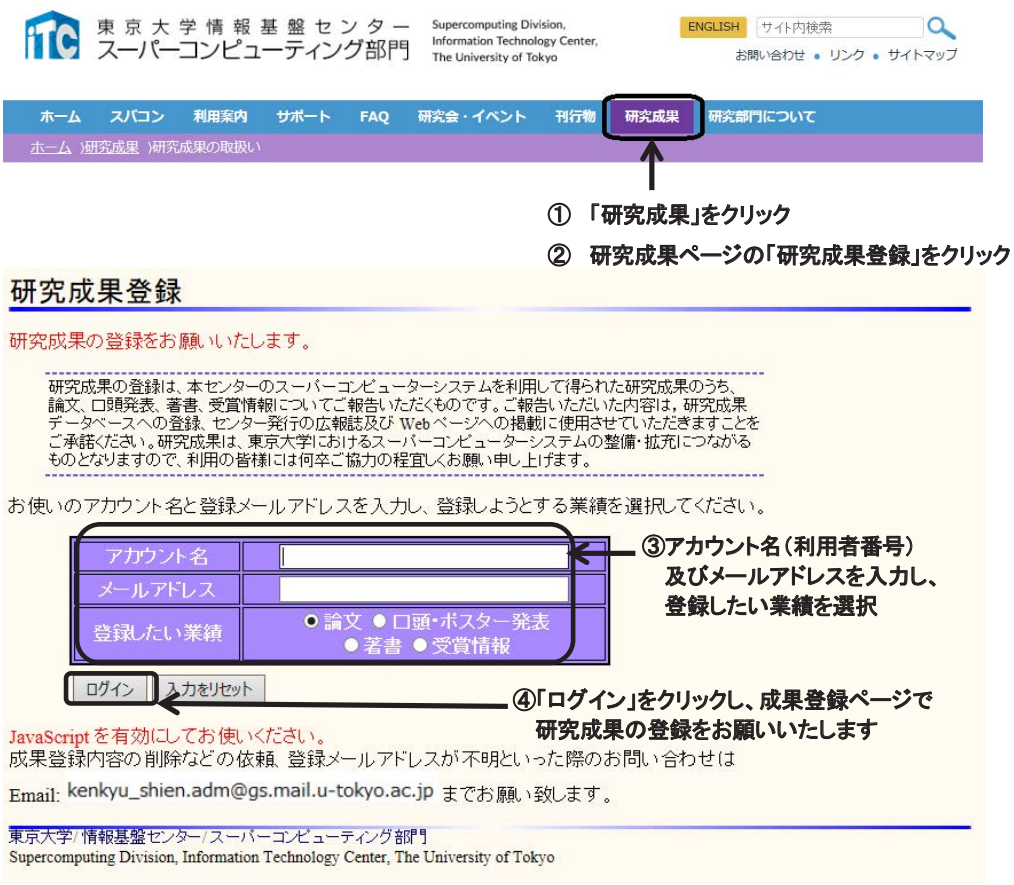
24. [Reedbush-L] 弘瀬拓矢、土岐富士緒、西村栄美、難波大輔、古徳純一: Label-free quality control and identification of human keratinocyte stem cells by deep learning-based automated cell tracking: Stem Cells.

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。

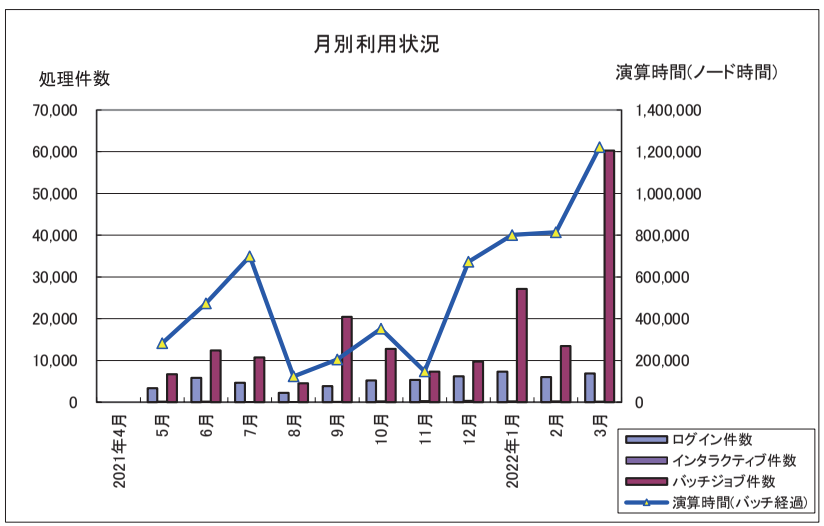


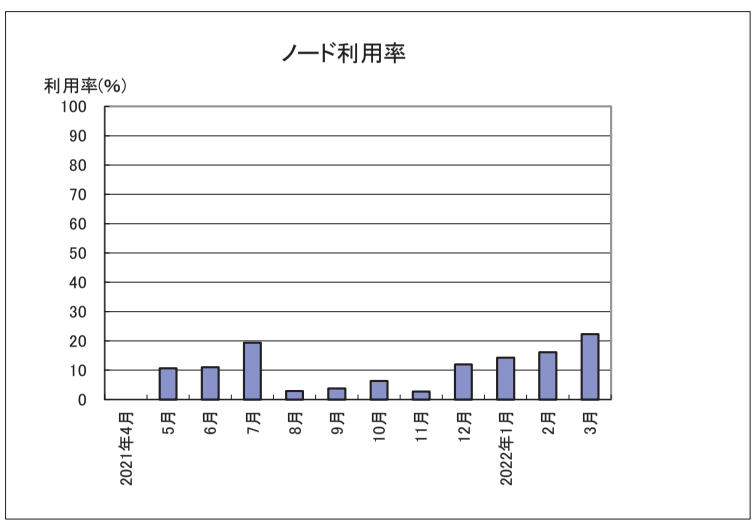
2・3月のジョブ統計

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				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	プリポスト	インタラクティブジョブ	バッチジョブ		/home	/lustre		プリポスト	インタラクティブジョブ	バッチジョブ		
2021年5月	860	206	3,348	54	188	6,718	13,623	148	66,121	697.32	40	35	282,715	822.0	10.7
6月	974	270	5,845	97	186	12,367	23,491	361	189,018	1,398.60	122	36	473,446	842.5	11.0
7月	1,103	229	4,638	66	64	10,730	24,723	438	211,696	1,670.10	74	10	698,405	1,492.2	19.4
8月	939	147	2,212	82	28	4,551	9,866	383	128,337	2,218.48	255	6	123,322	219.4	2.9
9月	1,058	196	3,835	76	150	20,431	19,769	434	246,063	1,944.79	281	40	204,279	289.0	3.8
10月	1,193	311	5,221	256	226	12,759	27,589	580	396,240	836.51	284	85	351,435	480.9	6.3
11月	1,279	256	5,337	76	311	7,340	26,676	774	515,631	3,092.91	86	91	147,549	208.8	2.7
12月	1,286	298	6,189	76	366	9,752	31,952	837	833,732	1,879.83	123	64	672,591	920.1	12.0
2022年1月	1,323	294	7,329	92	220	27,108	33,550	961	979,806	1,926.47	256	60	801,198	1,096.0	14.3
2月	1,293	269	6,015	168	275	13,449	29,553	1,105	1,021,929	2,675.51	496	124	813,968	1,235.3	16.1
3月	1,221	265	6,896	135	187	60,265	31,203	1,131	1,007,327	9,642.52	322	42	1,222,254	1,715.7	22.3
合計			56,865	1,178	2,201	185,470	271,995			27,983	2,339	593	5,791,162		

・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ヶ月の稼動時間
・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



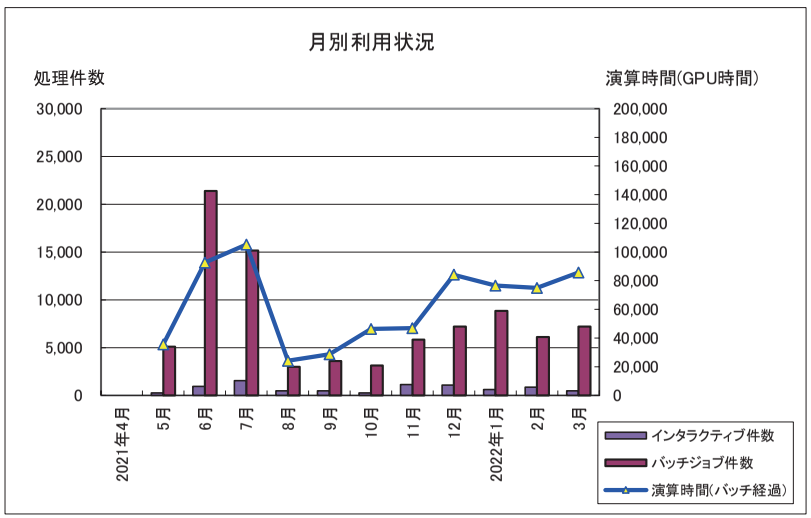


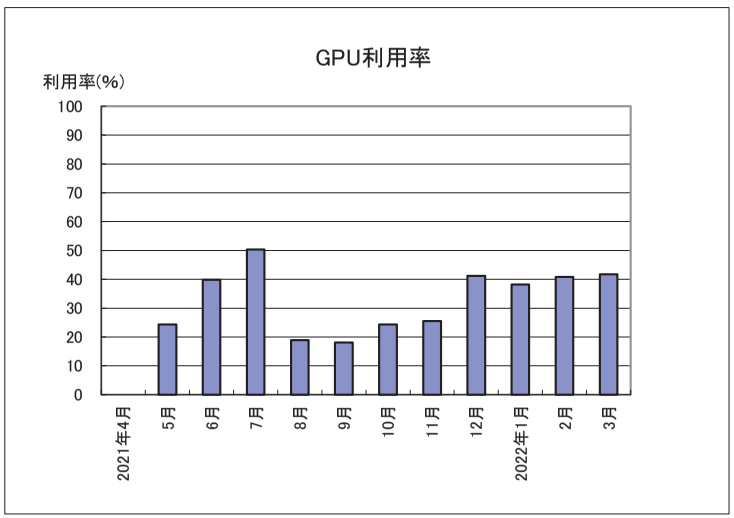
2. Wisteria/BDEC-01スーパーコンピュータシステム（Aquarius）ジョブ処理状況 （RedHat Enterprise Linux 8）

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5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
2月	863	6,116	801	74,982	147.0	40.8
3月	479	7,204	417	85,650	150.1	41.7
合計	8,119	86,585	7,758	701,088		

・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
・登録者数、実利用者数、ログイン件数、接続時間、ファイル使用量、
ログイン（実CPU）はWisteira/BDEC-01(Odyssey) と共通。

・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1GPU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ヶ月の稼動時間
・GPU利用率： サービスGPUに対する利用比率。 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）			平均ノード利用数（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	バッチジョブ		
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	74.2
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
2月	1,229	328	7,404	0	521	70,221	38,572	1,507	1,842,598	2,126.96	0	192	559,162	925.1	67.6
3月	1,245	317	6,767	0	403	48,195	39,693	1,579	1,674,826	3,006.83	0	228	534,752	830.9	60.7
合計			93,978	194	21,976	702,943	497,329			58,743	11	7,402	7,001,304		

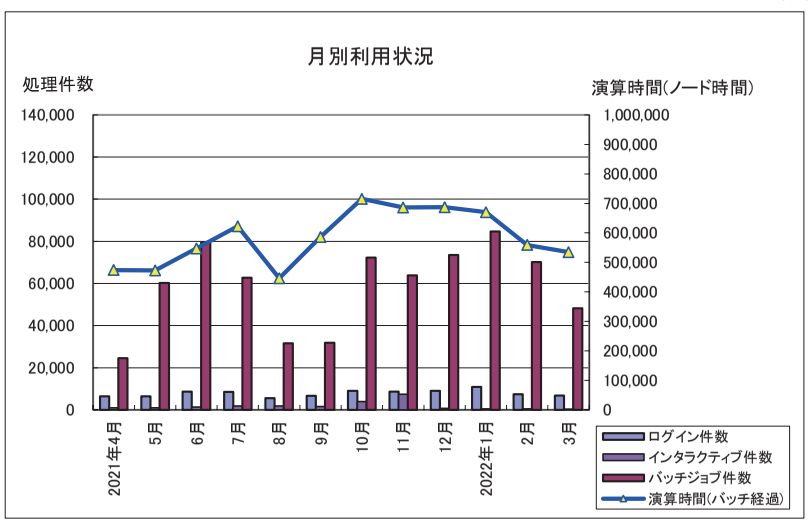
・接続時間： ログイン時間の累計

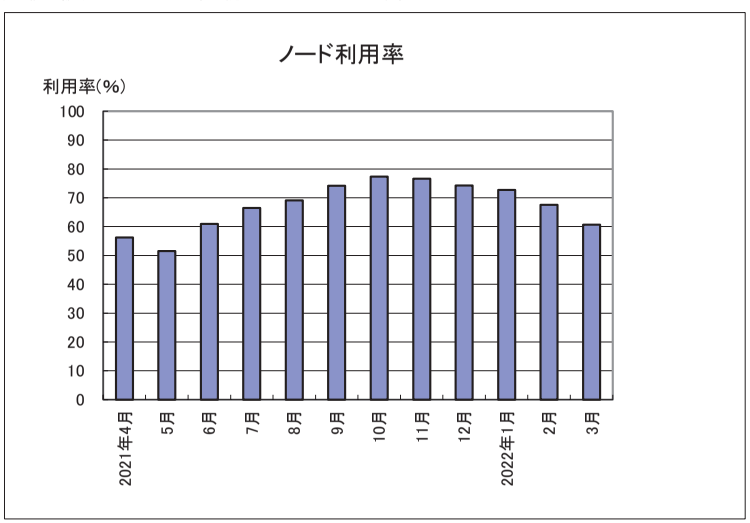
・ログイン（実CPU）： コア時間単位

・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ヶ月の稼動時間

・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

・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。





4. Oakforest-PACS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）

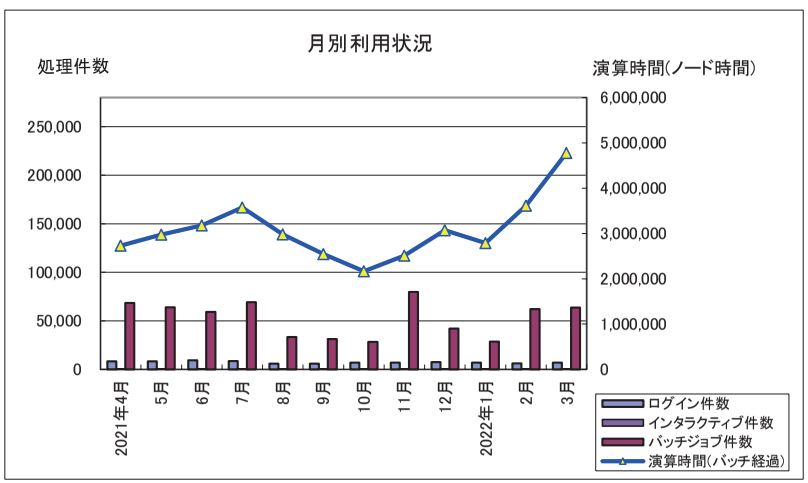
年月	登録者数	実利用者数	処理件数				接続時間[時間]	ファイル使用量 [GiB]		ログイン（実CPU）	演算時間 [ノード時間]（経過時間）			平均ノード利用数（ノード）	ノード利用率（%）
			ログイン	プリポスト	インタラクティブジョブ	バッチジョブ		/home	/work		プリポスト	インタラクティブジョブ	バッチジョブ		
2021年4月	1,690	450	8,322	128	181	68,373	47,950	2,618	6,233,823	3,528.98	310	65	2,731,129	4,116.8	50.2
5月	1,732	347	8,252	192	288	63,829	63,997	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
2月	1,433	318	6,111	151	173	62,095	44,197	2,364	5,238,301	11,174.58	268	67	3,612,361	5,490.5	66.9
3月	1,430	348	6,967	103	41	63,564	51,030	2,336	5,486,343	8,368.03	246	17	4,777,281	6,553.2	79.8
合計			87,625	2,246	2,363	629,076	617,573			90,399	4,364	1,015	36,915,936		

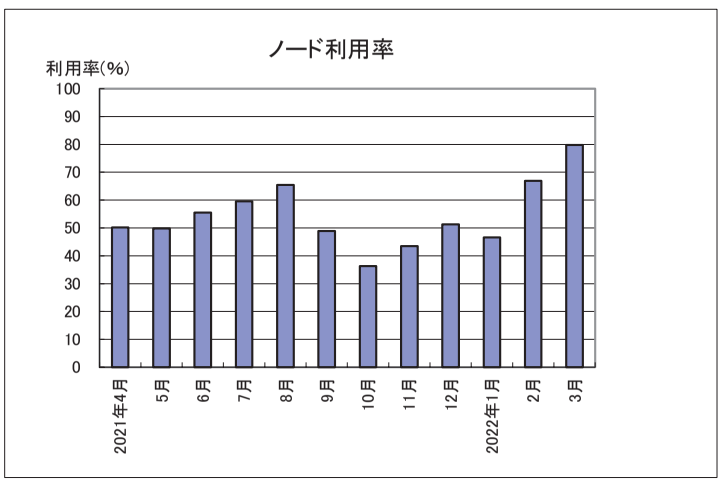
・接続時間： ログイン時間の累計

・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＝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比率。 計算式＝利用ノード時間÷サービスノード時間×100





Wisteria/BDEC-01 利用事例（4）
データ受け渡しライブラリ h3-Open-SYS/WaitIO (2/2)

住元真司（東京大学情報基盤センター）
坂口吉生（富士通株式会社）
松葉浩也* (日立製作所)
*2021 年度客員教授時の成果
中島研吾（東京大学情報基盤センター）

1. はじめに

今回は、前回（2022 年 3 月号）と次回（2022 年 7 月号）との 3 回からなる 2 回目で, 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 h3-Open-SYS/WaitIO［4］について、特に Wisteria/BDEC-01 システムでの実機利用を想定した利用事例について紹介する。

2. WaitIO Hostname Server

前回の実行では WaitIO のプログラムをバッチ実行するには PBID が 0 のホスト名とポート番号をバッチ実行開始後にログファイルから獲得して PBID 1 以降のバッチジョブスクリプトを変更した上で実行する必要があった。バッチ処理は実行時にホストが決まるからである。この状況を改善するため、実行時に決まるホスト名を実行時に中継する WaitIO Hostname Server を開発している。現在、システム連携に向けての改善継続中のため、今後仕様などが変更される可能性があるので注意されたい。今回は現状の動作仕様について説明する。

WaitIO Hostname Server は計算ノードから直接ネットワーク通信が可能なホストで実行可能なサーバであれば実行可能である。Wisteria/BDEC-01 システムではログインノードで実行可能である。実行バイナリとしては、intel CPU(waitio-serv, waitio-serv-intel : ログインノードと Aquarius)用と Arm CPU(waitio-serv-a64fx : Odyssey)用があり実行するノードで使い分ける必要がある。

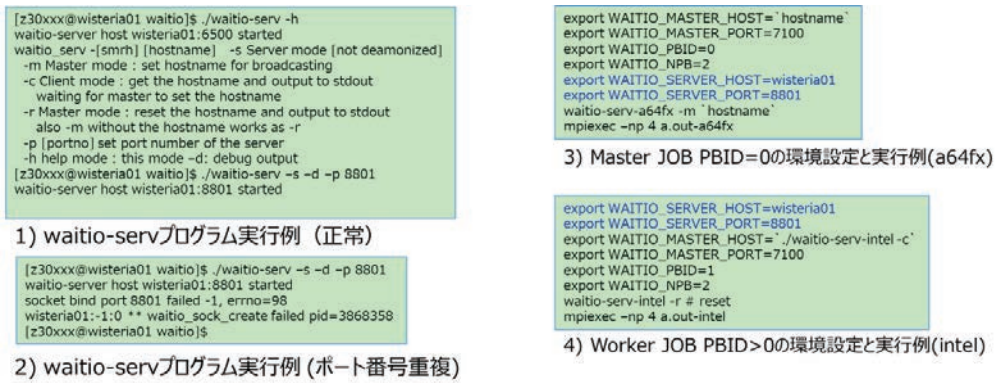


図 1. WaitIO Hostname Server の環境設定例

図 1 に WaitIO Hostname Server の環境設定例を示す。前回に示した実行例に WAITIO_SERVER_HOST と WAITIO_SERVEWR_PORT の環境変数を設定して ログインノードな

どで waitio_serv コマンドをサーバモードで起動（-s オプション、図 1、1)）し、各バッチスクリプトで実行（図 1、3）, 4)）することでバッチジョブ実行時に waitio_serv サーバを介してマスターホスト情報の中継が可能である。なお、waitio-serv コマンドをサーバモードで立ち上げる時、既にポート番号が使われていると Socket の取得に失敗することがある。（図 1 、2)）このような場合は重複した番号と異なるポート番号を再設定して実行する。

3. WaitIO-Socket のアプリケーション適用事例

WaitIO-Socket の利用事例としては、h3-Open-UTIL/MP［5］と観測データ同化による長周期地震動リアルタイム予測［11］で活用が進んでいる。

h3-Open-UTIL/MP は WaitIO-Socket と各 PB で実行される MPI ライブラリを用いて Hybrid な MPI 実行環境を実現し、複数の MPI アプリケーションを融合するカップラーとしてアプリケーション間のデータコンバージョンを含む実行環境を提供している。シミュレーションとデータ同化による学習の融合を実現する。詳しくは次回（2022 年 7 月号）の記事を参照されたい。

観測データ同化による長周期地震動リアルタイム予測［11］において WaitIO は日本各所に設置されている地震観測点からのデータをリアルタイムで転送し Wisteria/BDEC-01 システムでの予測に活用するために開発を進めている。

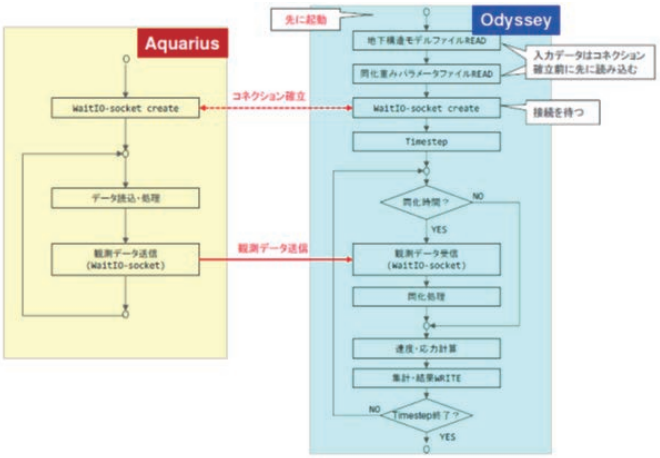


図 2. 観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例

図 2 に観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例を示す。観測済みデータを用いて地震動を解析する場合、通常蓄積されたデータをファイルに格納してこのファイルをアプリケーションに読み込ませて解析処理を実行する。

しかし、長周期地震動リアルタイム予測においては逐一記録される地震データをリアルタイムに解析する必要がある。WaitIO-Socket はこのファイル読み出しを通信に置き換える。

```
program dmy_filter
<省略: 型宣言等>
call mpi_init (ierr)
call mpi_comm_size (MPI_COMM_WORLD, nprocs, ierr)
call mpi_comm_rank (MPI_COMM_WORLD, myrank,  ierr)
call WAITIO_CREATE_UNIVERSE (WAITIO_COMM_UNIVERSE, ierr)

if (myrank==0) then
  open(100,file='./obsfile_list.txt', form='formatted', status='old', iostat=ierr)
  do i=1,300
   <省略: obsデータ読み込み処理>
    print *,"Send obs data ..... "
    call WAITIO_MPI_ISEND (NTMAX1_o, 1,          WAITIO_MPI_INTEGER,  2,1, WAITIO_COMM_UNIVERSE,req(1,1), ierr)
    call WAITIO_MPI_ISEND (DT_o,     1,          WAITIO_MPI_FLOAT,    2,2, WAITIO_COMM_UNIVERSE,req(1,2), ierr)
    call WAITIO_MPI_ISEND (NST_o,    1,          WAITIO_MPI_INTEGER,  2,3, WAITIO_COMM_UNIVERSE,req(1,3), ierr)
    call WAITIO_MPI_ISEND (AT_o,     1,          WAITIO_MPI_FLOAT,    2,4, WAITIO_COMM_UNIVERSE,req(1,4), ierr)
    call WAITIO_MPI_ISEND (T0_o,     1,          WAITIO_MPI_FLOAT,    2,5, WAITIO_COMM_UNIVERSE,req(1,5), ierr)
    call WAITIO_MPI_ISEND (ISO_X_o,  NSMAX,      WAITIO_MPI_INTEGER,  2,6, WAITIO_COMM_UNIVERSE,req(1,6), ierr)
    call WAITIO_MPI_ISEND (ISO_Y_o,  NSMAX,      WAITIO_MPI_INTEGER,  2,7, WAITIO_COMM_UNIVERSE,req(1,7), ierr)
    call WAITIO_MPI_ISEND (ISO_Z_o,  NSMAX,      WAITIO_MPI_INTEGER,  2,8, WAITIO_COMM_UNIVERSE,req(1,8), ierr)
    call WAITIO_MPI_ISEND (ISTX_o,   NST,        WAITIO_MPI_INTEGER,  2,9, WAITIO_COMM_UNIVERSE,req(1,9), ierr)
    call WAITIO_MPI_ISEND (ISTY_o,   NST,        WAITIO_MPI_INTEGER,  2,10,WAITIO_COMM_UNIVERSE,req(1,10),ierr)
    call WAITIO_MPI_ISEND (ISTZ_o,   NST,        WAITIO_MPI_INTEGER,  2,11,WAITIO_COMM_UNIVERSE,req(1,11),ierr)
    call WAITIO_MPI_ISEND (STC_o,    6*NST,      WAITIO_MPI_CHAR,     2,12,WAITIO_COMM_UNIVERSE,req(1,12),ierr)
    call WAITIO_MPI_ISEND (VxAll_obs,NST*NOBS_LEN,WAITIO_MPI_FLOAT,   2,13,WAITIO_COMM_UNIVERSE,req(1,13),ierr)
    call WAITIO_MPI_ISEND (VyAll_obs,NST*NOBS_LEN,WAITIO_MPI_FLOAT,   2,14,WAITIO_COMM_UNIVERSE,req(1,14),ierr)
    call WAITIO_MPI_ISEND (VzAll_obs,NST*NOBS_LEN,WAITIO_MPI_FLOAT,   2,15,WAITIO_COMM_UNIVERSE,req(1,15),ierr)
    call WAITIO_MPI_WAITALL (15,req, status, ierr)
    call sleep(1)
  enddo
  close (100)
endif
call WAITIO_FINALIZE (ierr)
call mpi_finalize (ierr)
end
```

図 3. 観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例：データ供給側

図 3, 図 4 に WaitIO-Socket を用いた長周期地震動リアルタイム予測で用いるデータ転送用のプログラムを示す。通常、システム間の通信は TCP/IP プロトコル上の Socket を使うことになるが、Socket の作成、相手サーバとの通信確立など Socket 通信自体のプログラム知識が必要であるが、WaitIO-Socket を用いることで Socket の知識がなくても MPI プログラムを書くことができれば、サーバ間の通信プログラムを簡易に書くことができる。

図 3 は地震データを格納するサーバにおけるプログラムでサーバ上のファイルを読んで WaitIO MPI Conversion ライブラリを用いてリアルタイム予測を実行するシステムに転送する。

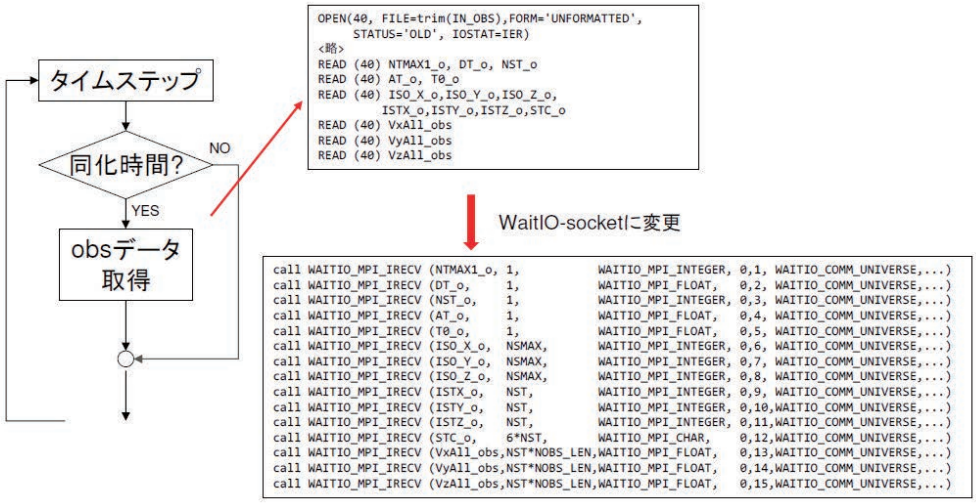


図 4. 観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例：データ活用側

図 4 は元々ファイルから地震データを読んで処理するプログラムであったものを WaitIO MPI Conversion ライブラリを用いて別のサーバからネットワークを経由してデータを獲得している。

このように Socket 通信のプログラム技術がなくても WaitIO Socket を用いて手軽にネットワークを介してデータを転送できることがわかる。

4. Toy プログラムによる実アプリケーションコード実行

本章では実際のアプリケーションプログラムを簡略化した Toy プログラムにより実際の WaitIO-Socket によるアプリケーション実行を試してみる。アプリケーション実行については、6 月より開始予定の Wisteria/BDEC-01 システムでの WaitIO 試行サービス環境下での実行手順を説明する。

ここでは、点源による発熱量計算結果を出力するプログラム(Code-1)のファイル出力を、有限要素法による三次元非定常熱伝導解析するプログラム(Code-2)、並びに Code-2 と Code-1 からのファイル出力を元に有限要素法による三次元非定常熱伝導解析を行い Z=Zmax 面の境界条件へ展開するプログラム(Code-3)の 3 つのプログラムを WaitIO-Socket を用いて書き換えてみる。ソースコード一式は Wisteria/BDEC-01 システム上の /work/share/waitio/src/examples/SYStest-waitio-socket.tgz に置かれているのでそれぞれのアカウントの work ディレクトリに展開して参照されたい。

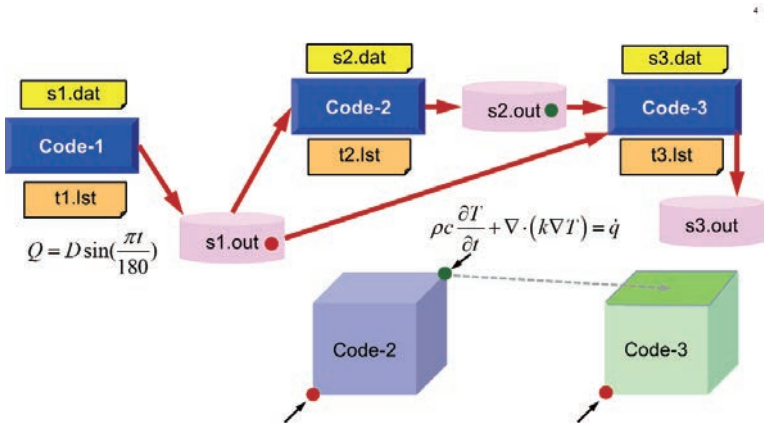


図 4. 有限要素法による三次元非定常熱伝導解析 Toy プログラムの構成

図 4 に本 Toy プログラムの全体像を、表 1 に各プログラムの概要と入出力ファイルについて示す。Code-1 の出力するファイル s1.out は Code-2 と Code-3 から読み込まれる。また、Code-2 の出力する s2.out は Code-3 から読み込まれる。これらのファイルによる入出力を WaitIO-Socket のデータ転送に書き換える。

表 1. 有限要素法による三次元非定常熱伝導解析 Toy プログラムの概要

	Code-1 (Dir:SYStest-waitio-socket/src1)	Code-2 (Dir:SYStest-waitio-socket/src2)	Code-3 (Dir:SYStest-waitio-socket/src3)
概要	発熱量(点源)計算	FEMによる非定常三次元熱伝導方程式, MPI並列化	FEMによる非定常三次元熱伝導方程式, MPI並列化
入力 (Dir:SYStest-waitio-socket/run)	s1.dat	s2.dat s1.out	s3.dat s1.out, s2.out
出力 (Dir:SYStest-waitio-socket/run)	s1.out, t1.lst	s2.out, t2.lst	s3.out, t3.lst

4.1 ソースコードの修正

ファイル出力から WaitIO でのデータ連携への変更は、データ共有しているファイルへの入出力部分のソースコードを WaitIO による送受信のコードによる送受信のソースコードに変更することで実現可能である。

WaitIO におけるソースコードの修正手順を以下に示す。

1. 各アプリケーションプロセス間の接続方式と利用 API の決定

   WaitIO MPI Conversion ライブラリでは 2 種類の簡易初期化関数を準備しています。

   1) 各アプリケーションの MPI プロセスのすべてを用いる waitio_create_universe()関数
   2) 各アプリケーションの MPI プロセスの先頭プロセス(MPI ランク番号 0)だけを使う waitio_create_universe_pbhead()関数

   1), 2)以外の場合は waitio_create_group()関数を用いて個別に作成することが可能です。

   また、実際の通信のデータ形式や通信パターンに応じて WaitIO 関数(waitio_isend, waitio_irecv)、もしくは WaitIO MPI Conversion ライブラリ(waitio_mpi_xxx()関数)を選択可能です。ここでは、アプリケーション間の接続方式として、各アプリケーションの先頭プロセスのみを選択する waitio_create_universe_pbhead()関数を選択し、利用関数として表記法が MPI と類似している WaitIO MPI Conversion ライブラリで記述することにします。
2. 各アプリケーションの PBID へのマッピング

   1 で述べた接続方式と PBID により、WaitIO で用いるプロセスランク番号が異なるため、アプリケーションの結合順序を決めるために各アプリケーションの PBID を決定します。ここでは、Code-1 を PBID=0, Code-2 を PBID=1, Code-3 を PBID=2 と定義します。
3. アプリケーション間のデータ連携コードのプログラミング

   さて、実際のデータ連携コードのプログラミングであるが、今回は、ファイル入出力によるデータ交換を WaitIO で書き換えることになるため、実際のファイル入出力がどのように行われているのかを整理する必要がある。

以下の節では実際のコード修正を元に詳しく述べる。

4.2 アプリケーション間のデータ連携コードのプログラミングの実際

本節では Toy プログラムのデータ連携コードを WaitIO MPI Conversion ライブラリの関数を用い

て書き換える。Toy プログラムである SYStest-waitio-socket ディレクトリ下には各プログラムを格納する src1(Code-1), src2(Code-2), src3(Code-3)の各ディレクトリと実行を行う run ディレクトリから構成される。

```
001   implicit REAL*8(A-H,O-Z)
002   real(kind=8) :: Interval
003   include 'mpif.h'

004   call MPI_Init(ierr)
005   open (11,file='s1.dat',status='unknown')
006   read (11,*) ITERmax, Period, Interval, Val
007   write (*,'(a,i8)')     '##ITERmax ', ITERmax
008   write (*,'(a,1pe16.6)') '##Period  ', Period
009   write (*,'(a,1pe16.6)') '##Interval', Interval
010   write (*,'(a,1pe16.6//)') '##Val     ', Val
011   close (11)

012   open (12,file='s1.out',status='unknown')

013   pi  = 4.d0*datan(1.d0)
014   coef= pi/180.d0
015   time= 0.d0
016   S0time= mpi_wtime ()
017   do
018    S1time= mpi_wtime ()

019    do
020     do iter= 1, ITERmax
021       a= 1.d0
022     enddo
023     E0time= mpi_wtime (ierr)
024     if ((E0time-S1time).gt.Interval) exit
025    enddo

026    ttt= E0time - S0time
027    Source= Val * dsin(ttt*coef)
028 !  rewind (12)
029    write (12,'(2(1pe16.6))') ttt, Source
030   enddo

031   call MPI_Finalize()
032   stop
033   end
```

図 5. Toy プログラムの Original プログラム：Code-1 全体ソースコード(FORTRAN)

File:SYStest-waitio-socket/src1/test1.f

図 5 に、Toy プログラムの内 Code-1(FORTRAN)の Original の全体ソースコードを示す。図 5 中、001-016 行目までは初期化コードであり、004 行で MPI の初期化後、005 行でパラメータファイルを Open し、006 行でパラメータを読み込んでいる。そして、012 行で出力用のファイル s1.out を Open した後、017 行からのループ本体のための変数初期化を行っている。

017-030 行が Code-1 の本体ループである。017-030 行のループ本体では計算結果を 029 行の write 文で変数 ttt と Source をループ毎に書き出している。

今回、図 5 の Code-1 コードを WaitIO 対応のコードにするために、まず、最初に各ファイルのプログラム間の依存関係を把握する必要がある。図 4、表 1 より、Code-1 にて出力される s1.out は、Code-2, Code-3 で読みだされている。このため、Code-1 の s1.out 出力は Code-2, Code-3 に送信する必要がある。表 1 より、Code-2 においては Code-1 からのデータを読み込み、Code-2 へ送信、Code-3 においては、Code-1 と Code-2 からのデータを読み込む必要がある。

```
001    implicit REAL*8(A-H,O-Z)
002    real(kind=8) :: Interval
003    integer :: dest
004    integer(kind=4 ), dimension(:,:), save,allocatable :: sta1
005    integer(kind=4 ), dimension(:,:), save,allocatable :: req1

006    include 'mpif.h'
007    include 'waitio_mpif.h'

008    call MPI_Init(ierr)
009    open (11,file='s1.dat',status='unknown')
010    read (11,*) ITERmax, Period, Interval, Val
011    write (*,'(a,i8/)')     '##ITERmax ', ITERmax
012    write (*,'(a,1pe16.6)') '##Period  ', Period
013    write (*,'(a,1pe16.6)') '##Interval', Interval
014    write (*,'(a,1pe16.6//)') '##Val     ', Val
015    close (11)

016    allocate (sta1(WAITIO_STATUS_SIZE,2*2))
017    allocate (req1(WAITIO_REQUEST_SIZE,2*2))
018    call WAITIO_CREATE_UNIVERSE_PBHEAD (WAITIO_SOLVER_COMM, ierr)

019    open (12,file='s1.out',status='unknown')
```

```
037      write (*,'(2(1pe16.6))') ttt, Source
038      do dest=1, 2
039        call WAITIO_MPI_Isend (ttt, 1,
040    &        WAITIO_MPI_DOUBLE_PRECISION,
041    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,2*(dest-1)+1), ierr)
042        if(ierr.ne.0) goto 100
043        call WAITIO_MPI_Isend (Source, 1,
044    &        WAITIO_MPI_DOUBLE_PRECISION,
045    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,2*(dest-1)+2), ierr)
046        if(ierr.ne.0) goto 100
047      enddo
048      call WAITIO_MPI_Waitall (4, req1, sta1, ierr)
049      if(ierr.ne.0) goto 100
050    enddo
051 100  continue
```

図 6. Toy プログラムの WaitIO プログラム：Code-1 修正部

File:SYStest-waitio-socket/src1/test1_waitio.f

図 6 に、Toy プログラムの WaitIO プログラム Code-1 の修正済み FORTRAN コードを示す。図 6 中、青字のラインが追加したコードである。図 6 にて 038-048 行のループでファイル出力される変数 ttt と Source を WaitIO MPI Conversion ライブラリを用いて Code-2(WatiIO ランク番号= 1)と Code-3(WatiIO ランク番号=2)に送信している。この 2 つの変数の送信には 2 つの WAITIO_MPI_Isend()関数を用い、2 か所に送信するために WaitIO ランク番号 1,2 にループを用いている。それぞれの WAITIO_MPI_Isend()関数に必要な Request 構造体は 005,017 行で 2 次元配列として割り当てている。038-048 行のループでそれぞれの WaitIO ランク番号 1,2 に対して WAITIO_MPI_Isend()関数を 2 回発行した後、048 行の WATIO_MPI_Waitall()関数で相手が受信したことを待ち合わせる。

なお、018 行の WAITIO_CREATE_UNIVERSE_PBHEAD() 関数の引数である WAITIO_SOLVER_COMM 変数は 8 バイトのポインターが格納されるため 8 バイトの変数定義が必要である。本コードでは implicit 定義により 8 バイトの REAL 変数として定義されている。

```
007    integer :: dest
008    integer(kind=kint ), dimension(:,:), save,allocatable :: sta1
009    integer(kind=kint ), dimension(:,:), save,allocatable :: req1

010    allocate (sta1(WAITIO_STATUS_SIZE,1*6))
011    allocate (req1(WAITIO_REQUEST_SIZE,1*6))

!=== SNIP SNIP ===

053!c       read (12,*,end=100) ttt, Source
054         dest=0
055         call WAITIO_MPI_Irecv (ttt, 1,
056    &        WAITIO_MPI_DOUBLE_PRECISION,
057    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
058         call WAITIO_MPI_Irecv (Source, 1,
059    &        WAITIO_MPI_DOUBLE_PRECISION,
060    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)

061         call WAITIO_MPI_Waitall (2, req1, sta1, ierr)
```

```
100      write (22,'(6(1pe16.6))') TIME, ttt, Source, X(ii1),
101    &                           X(ii2), VAL
102      dest= 2
103      call WAITIO_MPI_Isend (TIME, 1,
104    &      WAITIO_MPI_DOUBLE_PRECISION,
105    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
106      call WAITIO_MPI_Isend (ttt, 1,
107    &      WAITIO_MPI_DOUBLE_PRECISION,
108    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)
109      call WAITIO_MPI_Isend (Source, 1,
110    &      WAITIO_MPI_DOUBLE_PRECISION,
111    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,3), ierr)
112      call WAITIO_MPI_Isend (X(ii1), 1,
113    &      WAITIO_MPI_DOUBLE_PRECISION,
114    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,4), ierr)
115      call WAITIO_MPI_Isend (X(ii2), 1,
116    &      WAITIO_MPI_DOUBLE_PRECISION,
117    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,5), ierr)
118      call WAITIO_MPI_Isend (VAL, 1,
119    &      WAITIO_MPI_DOUBLE_PRECISION,
120    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,6), ierr)

121      call WAITIO_MPI_Waitall (6, req1, sta1, ierr)
```

図 7. Toy プログラムの WaitIO プログラム：Code-2 修正部

File:SYStest-waitio-socket/src2/test1_waitio.f

図 7 に、Toy プログラムの WaitIO プログラムの内 Code-2 の修正済み FORTRAN コードを示す。この修正の他初期化コードも Code-1 と同様に追加されている。Code-1 と同様 WaitIO MPI Conversion ライブラリに置き換えている。

図 7 中、054-061 行は Code-1(WatiIO ランク番号=0)からのデータ受信である。コメントアウトされた 053 行を WaitIO MPI Conversion ライブラリに書き換えたものである。また、102-121 行は同じく 100 行のファイル出力を WaitIO MPI Conversion ライブラリに書き換え、Code-3(WatiIO ランク番号=2)への通信に変換したものである。

```
007     integer :: dest
008     integer(kind=kint ), dimension(:,:), save,allocatable :: sta1
009     integer(kind=kint ), dimension(:,:), save,allocatable :: req1

010     allocate (sta1(WAITIO_STATUS_SIZE,1*6))
011     allocate (req1(WAITIO_REQUEST_SIZE,1*6))

!=== SNIP SNIP ===

054IC       read (12,*,end=200) ttx, Source
055         dest=0
056         call WAITIO_MPI_Irecv (ttx, 1,
057     &       WAITIO_MPI_DOUBLE_PRECISION,
058     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
059         call WAITIO_MPI_Irecv (Source, 1,
060     &       WAITIO_MPI_DOUBLE_PRECISION,
061     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)

062         call WAITIO_MPI_Waitall (2, req1, sta1, ierr)
063         if (ttx.ge.TIME) goto 200
```

```
071IC       read (13,*,end=100) ttt, tt1, s0, s1, s2, TBOU
072         dest= 1
073         call WAITIO_MPI_Irecv (ttt, 1,
074     &       WAITIO_MPI_DOUBLE_PRECISION,
075     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
076         call WAITIO_MPI_Irecv (tt1, 1,
077     &       WAITIO_MPI_DOUBLE_PRECISION,
078     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)
079         call WAITIO_MPI_Irecv (s0, 1,
080     &       WAITIO_MPI_DOUBLE_PRECISION,
081     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,3), ierr)
082         call WAITIO_MPI_Irecv (s1, 1,
083     &       WAITIO_MPI_DOUBLE_PRECISION,
084     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,4), ierr)
085         call WAITIO_MPI_Irecv (s2, 1,
086     &       WAITIO_MPI_DOUBLE_PRECISION,
087     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,5), ierr)
088         call WAITIO_MPI_Irecv (TBOU, 1,
089     &       WAITIO_MPI_DOUBLE_PRECISION,
090     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,6), ierr)

091         call WAITIO_MPI_Waitall (6, req1, sta1, ierr)
092         if (ttt.ge.TIME) goto 100
```

図 8. Toy プログラムの WaitIO プログラム：Code-3 修正部

File:SYStest-waitio-socket/src3/test1_waitio.f

図 8 に、Toy プログラムの WaitIO プログラムの内 Code-3 の修正済み FORTRAN コードを示す。この修正の他初期化コードも Code-1, Code-2 と同様に追加されている。Code-1, Code-2 と同様 WaitIO MPI Conversion ライブラリに置き換えている。

図 8 中、Code-2 と同様に 055-062 行は Code-1(WatiIO ランク番号=0)からのデータ受信である。コメントアウトされた 054 行を WaitIO MPI Conversion ライブラリに書き換えたものである。また、072-091 行は 071 行のファイル入力を WaitIO MPI Conversion ライブラリに書き換え、Code-2(WatiIO ランク番号=2)からのデータ受信に変換したものである。

以上のように、ファイル入出力を WaitIO MPI Conversion ライブラリに置き換えるのは機械的に行うことが可能である。

4.3 Wisteria/BDEC-01 における WaitIO 実行環境の概要

本節では 6 月よりサービス開始を予定している WaitIO の試行環境について説明する。

- WaitIO 関連のファイルは Wistria/BDEC-01 の/work/share/waitio 下に配置される。
  Toy プログラム等、実行用のテストサンプルは以下に配置される /work/share/waitio/src/examples/
- module 実行環境として waitio が提供され PATH、LD_LIBRARY_PATH 等の実行に必要な設定が提供される。例えばバッチスクリプト内で module load waitio とするとジョブ実行に必要な環境設定が実行される。

- この他、Wisteria/BDEC-01 の Odyssey と Aquarius 間でのバッチ処理協調実行のための仕組みも準備中である。

これ以降の Toy プログラムのコンパイルと実行の説明では以上の環境下における試行について説明する。

4.4 Toy プログラムのコンパイルと実行

本節では 6 月に Wistria/BDEC-01 での WaitIO サービス開始時の環境を想定したプログラムのコンパイルと実行の概要を説明する。前節で書いた通り WaitIO に関連するファイル群は Wistria/BDEC-01 の/work/share/waitio ディレクトリに置かれている。Toy プログラムのソース並びに実行環境は/work/share/waitio/src/examples/SYStest-waitio-socket.tgz に一式アーカイブされているので各ユーザの/work 下のディレクトリに展開して利用する。図 9 に一連の処理を示す。

Toy プログラムのアーカイブを各ユーザの work ディレクトリに展開するとコード一式は SYStest-waitio-socket ディレクトリ下に展開されるので、このディレクトリ下で make コマンドを実行すると SYStest-waitio-socket 配下の run ディレクトリに Odyssey 用バイナリ(-a64fx)と Aquarius 用バイナリ(-intel)の 3 種類 x 2 CPU タイプ＝計 6 種の WaitIO 用の実行バイナリとオリジナルの実行バイナリ 3 種が生成される。(図 9)

```
0) Extract the Toy program to your /work directory and compile
[z30xxx@wisteria01 sample]$ realpath .
/work/jh21yyyya/z30xxx/sample
[z30xxx@wisteria01 sample]$ tar -zxf /work/share/waitio/src/examples/SYStest-waitio-socket.tgz
[z30xxx@wisteria01 sample]$ cd SYStest-waitio-socket/
[z30xxx@wisteria01 SYStest-waitio-socket]$ make
(cd src1 ; make )
....
make[1]: Leaving directory '/work/01/jh210022a/z30455/sample/SYStest-waitio-socket/src3'
[z30xxx@wisteria01 SYStest-waitio-socket]$
[z30xxx@wisteria01 SYStest-waitio-socket]$ ls -t run/
sol3-a64fx  sol2       sol1    s3-intel.sh  s1-a64fx.sh  s2.dat  OrgResults
sol3-intel  sol2-intel  s3.out  s2-intel.sh  s3-a64fx.sh  s3.sh
sol3        sol1-a64fx  s2.out  s2-a64fx.sh  s1.dat       s2.sh
sol2-a64fx  sol1-intel  s1.out  s1-intel.sh  s3.dat       s1.sh
[z30xxx@wisteria01 SYStest-waitio-socket]$ cd run/
[z30xxx@wisteria01 run]$
```

図 9. WaitIO 版 Toy プログラムのファイル展開とコンパイル

図 10. 11. に Wisteria/BDEC-01 システム(Odyssey+Aquarius)で実行可能なバッチスクリプトを示す。これらのバッチスクリプトは wisteria01 ノード上で 2 章説明の waitio-serv コマンドによるサーバの利用が前提である。これにより、ジョブ実行時のバッチスクリプトの修正は不要である。また、waitio module の利用により環境変数 PATH と LD_LIBRARY_PATH の設定が不要になっている。

図 10. Toy プログラムの WaitIO バッチ実行用スクリプト: a64fx

図 11. Toy プログラムの WaitIO バッチ実行用スクリプト: intel

図 10、図 11 のバッチスクリプトは Code-1+Code-2+Code-3 の組み合わせであれば a64fx 用、intel 用どの組み合わせでも動作可能である。実行時に注意することは Code-1, Code-2 のジョブを Submit した後、Code-2 が実行開始した後に Code-3 を Submit することである。これは、Code-3 のジョブスクリプトの mpiexec 行の前に実行される waitio-serv –r コマンドが登録されたサーバ設定をリセットするためである。このため、リセット後に Code-2 のジョブが実行開始されてもサーバ名を獲得できず実行待ちになる。

図 12, 図 13 に実際の WaitIO 版 Toy プログラムの実行例を示す。図 12 が実際のジョブ実行の様子を示しており、図 13 が一連のジョブ実行に必要な waitio-serv プログラムの実行出力である。

図 12 では、Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)の組み合わせでの実行結果であるが、Code-1, Code-2, Code-3 のそれぞれ任意の a64fx 並びに intel 用のスクリプトを選択可能である。

図 13 の waitio-serv プログラムの出力であるが、Code-1 による Master Host の登録に始まり、Code-3 による Master Host 登録のリセットで一連のジョブ実行が終了する。

```
1) Submit Code-1 and Code-2 Jobs and wait until both jobs running
[z30xxx@wisteria01 run]$ pjsub s1-intel.sh
[INFO] PJM 0000 pjsub Job 335684 submitted.
[z30xxx@wisteria01 run]$ pjsub s2-a64fx.sh
[INFO] PJM 0000 pjsub Job 335685 submitted.
[z30xxx@wisteria01 run]$ pjstat
Wisteria/BDEC-01 scheduled stop time: 2022/04/22(Fri) 09:00:00 (Remain:  1days 18:55:30)

JOB_ID    JOB_NAME  STATUS  PROJECT   RSCGROUP      START_DATE      ELAPSE      TOKEN      NODE  GPU
335684    SYStest-wa RUNNING jhxxxxxxa  debug-a       04/20 14:04:22<  00:00:09      0.1        1   8
335685    SYStest-wa RUNNING jhxxxxxxo  debug-o       04/20 14:04:23<  00:00:08      0.0        1   -

2) Then Submit Code-3 Job and wait until Code-2 and Code-3 finish
[z30xxx@wisteria01 run]$ pjsub s3-a64fx.sh
[INFO] PJM 0000 pjsub Job 335686 submitted.
[z30xxx@wisteria01 run]$ pjstat
Wisteria/BDEC-01 scheduled stop time: 2022/04/22(Fri) 09:00:00 (Remain:  1days 18:55:15)

JOB_ID    JOB_NAME  STATUS  PROJECT   RSCGROUP      START_DATE      ELAPSE      TOKEN      NODE  GPU
335684    SYStest-wa RUNNING jhxxxxxxa  debug-a       04/20 14:04:22<  00:00:24      0.2        1   8
335685    SYStest-wa RUNNING jhxxxxxxo  debug-o       04/20 14:04:23<  00:00:23      0.0        1   -
335686    SYStest-wa RUNNING jhxxxxxxo  debug-o       04/20 14:04:44<  00:00:02      0.0        1   -
[z30xxx@wisteria01 run]$

3) After Code-2 and Code-3 has finished, then kill Code-1 Job
[z30xxx@wisteria01 run]$ pjstat
Wisteria/BDEC-01 scheduled stop time: 2022/04/22(Fri) 09:00:00 (Remain:  1days 18:51:31)

JOB_ID    JOB_NAME  STATUS  PROJECT   RSCGROUP      START_DATE      ELAPSE      TOKEN      NODE  GPU
335684    SYStest-wa RUNNING jhxxxxxxa  debug-a       04/20 14:04:22<  00:04:08      1.7        1   8
[z30xxx@wisteria01 run]$ pjdel 335684
[INFO] PJM 0100 pjdel Accepted job 335684.
[z30xxx@wisteria01 run]$
```

図 12. WaitIO 版 Toy プログラムの実行例: Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)

```
[z30xxx@wisteria01 waitio]$ ./waitio-serv -s -p 8801 -d
waitio-server host wisteria01:8801 started
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.1.6.1 port=7858 fd=5
==== wisteria01:(1722609):waitio_serv_loop:master host has set to wa01-ib0 ====
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.11.101.1 port=3210 fd=5
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.11.101.9 port=30390 fd=5
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.11.101.9 port=30902 fd=5
==== wisteria01:(1722609):waitio_serv_loop:master host  reset! ====
```

図 13. WaitIO 版 Toy プログラムの実行例 waitio-serv 出力: Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)

```
#   時間          s2.out時間     ●発熱量        ●結果          ●の隣結果       ●結果          ●結果
   1.000000E-01  1.000000E-01  1.745347E-03  5.602511E-02  3.225460E-03  0.000000E+00  0.000000E+00
   2.000000E-01  2.000000E-01  1.663485E-02  5.372070E-01  3.184216E-02  0.000000E+00  0.000000E+00
   3.000000E-01  3.000000E-01  1.838060E-02  6.213567E-01  4.488534E-02  1.749074E-64  1.749074E-64
   4.000000E-01  4.000000E-01  2.012594E-02  6.853573E-01  5.309708E-02  1.931968E-35  1.931968E-35
   5.000000E-01  5.000000E-01  2.187122E-02  7.472359E-01  5.987997E-02  7.930161E-32  7.930161E-32

<Snip><Snip><Snip>

   2.920000E+01  2.920000E+01  4.993690E-01  1.739279E+01  1.646121E+00  9.098841E-05  9.098841E-05
   2.930000E+01  2.930000E+01  5.008806E-01  1.744550E+01  1.651154E+00  9.284215E-05  9.284215E-05
   2.940000E+01  2.940000E+01  5.023908E-01  1.749815E+01  1.656181E+00  9.472301E-05  9.472301E-05
   2.950000E+01  2.950000E+01  5.038994E-01  1.755075E+01  1.661204E+00  9.663123E-05  9.663123E-05
   2.960000E+01  2.960000E+01  5.054064E-01  1.760330E+01  1.666221E+00  9.856701E-05  9.856701E-05
   2.970000E+01  2.970000E+01  5.069119E-01  1.765580E+01  1.671233E+00  1.005306E-04  1.005306E-04
   2.980000E+01  2.980000E+01  5.084159E-01  1.770824E+01  1.676241E+00  1.025222E-04  1.025222E-04
   2.990000E+01  2.990000E+01  5.099183E-01  1.776062E+01  1.681243E+00  1.045420E-04  1.045420E-04
   3.000000E+01  3.000000E+01  5.114191E-01  1.781295E+01  1.686240E+00  1.065902E-04  1.065902E-04
```

図 14. WaitIO 版 Toy プログラムの実行例: s3.out ファイル出力: Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)

図 14 に最終出力である s3.out ファイルの出力結果の抜粋を示す。本プログラムでは最後の 2 つの結果が一致することで計算が正しく実行されているかを確認することができる。

5. まとめと今後の予定

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 の概要について、適用事例を含めたプログラミングと実行例を交えて紹介した。本号の別記事では h3-Open-UTIL/MP の概要、次号では 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ので合わせて参考にされたい。

WaitIO-Socket は 6 月より Wisteria でのサービスを開始する予定で、徐々にアプリケーションユーザへの適用事例を増やしていくべくユーザを募集中である。広くユーザに使っていただくため、今年度チュートリアル・講習会の開催を計画している。ご興味のある方は参加いただければ幸いである。

今後は実運用に耐えるべく正常系の処理に加え異常系の処理を拡充すると共にセンター内だけでなくセンター間でのデータ通信にも活用可能なライブラリを拡張していく予定である。

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 利用事例（5）
マルチプログラム連成ライブラリ h3-Open-UTIL/MP (1/2)

荒川隆（CliMTech）
八代尚（国立環境研究所）
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 7 月号）の 2 回に分けて複数のシミュレーションプログラムを連成実行し「計算・データ・学習」融合を推進するためのライブラリ h3-Open-UTIL/MP について紹介する。今回は一般的な単一アーキテクチャ環境で複数プログラムを連成するケースについて説明し，次回では h3-Open-UTIL/MP の特徴的な機能であるアンサンブル連成，Python APP 連成，異機種間連成機能などについて紹介する。なお対象読者としては MPI を用いた並列計算には一通り習熟しているが連成計算については未経験のレベルを想定し，今号では煩瑣をいとわず連成計算の基礎的な概念や実行方法も含めて解説してゆく。なお，このような計算方法は「連成解析」「連成計算」「結合計算」など複数の用語が用いられるが本稿では基本的に「連成計算」の語を用い，一部はコミュニティの慣習に従って結合（例：大気海洋結合）を用いることとする。

2　連成計算の基本概念

本節では連成計算の基本的な概念について解説し，次節で MPI 環境における連成計算の実行方法を説明する。既に MPI を用いた連成計算の経験がある読者は本節および次節は飛ばしても差し支えない。

連成計算の基本概念を説明する前に，ひとつの逸話を紹介したい。司馬遼太郎の「街道をゆく」の一篇[1]に，ある僧侶が鈴木大拙の著作「華厳の研究」の翻訳（原著は英語）を読み華厳経の難解な仏教用語である「相即相入」が平易な言葉として表現されていることで経典の内容が腑に落ちる，という経験をしたことが語られている。この際，鈴木大拙が用いた語が interpenetration で，辞書的には相互浸透あるいは相互貫入と説明されている。すなわち世界のすべては相互に影響し合い，何一つとして独立して存在するものはないという意である。これをシミュレーションに置き換えると，単一の事象を再現するだけでは不完全であり関連する様々な事象との相互作用も含めて考える必要がある，という事になろう。連成計算とはまさにこの世界の構造（の極めて不完全なごく一部）を計算機内で実現するための計算手法である。

改めて述べると，連成計算とは複数のシミュレーションプログラムを複合させ相互作用を伴いながら計算してゆく手法である。計算手法として複数の場をひとつの支配方程式で解く強連成と支配方程式を個別に解いて時間ステップ毎に解を交換しながら計算を進める弱連成に大別される。本稿で述べる h3-Open-UTIL/MP は不特定のシミュレーションモデルを対象として弱連成計算を容易に実現するためのソフトウェアである。このようなソフトウェアを一般にカプラと呼ぶ。現代では連成計算は構造や流体，熱解析など様々な分野で実現されており専用の解析ソフトウェアも多くリリースされているが，ここでは連成計算の先駆的な事例のひとつである気象・気候シミュレーションを事例として連成計算の基本概念について解説する。2021 年ノー

ベル物理学賞受賞者に眞鍋淑郎博士が選ばれたことは，本稿の読者にとっても既知のことであろう。真鍋の代表的な業績が大気海洋結合モデルの開発およびそれを用いた気候変動予測シミュレーションである[2,3]。この計算では大気モデルと海洋モデルが相互に海水面温度や風速などの情報（データ）を交換しながら時間積分を進めるようになっている。現代の最先端の気象・気候シミュレーションモデルでも真鍋が開発した基本的なフレームは変わっておらず，例えば日本の代表的なモデルのひとつである MIROC(Model for Interdisciplinary Research on Climate) は大気モデルと海洋モデルの２つのモデルコンポーネントを中心にしてシステムが構成されている[4]。大気モデルと海洋モデルは異なるバイナリとしてコンパイル・リンクされ，MPI 環境下で同時に実行される（具体的な実行方法については後述）。大きなモデルコンポーネントは大気と海洋の２つであるが他にも陸面や河川などの過程がコンポーネント化されており，これらは大気モデルの一サブルーチンとして大気モデルのプログラムに組み込まれている。実際の計算ではこれらの要素モデルが相互に必要な情報を交換しながら時間積分を進めてゆくことになる。この際，個々のモデルコンポーネントはそれが表現している現象に応じた時空間構造を持つため，情報の交換に際しては構造の差違を吸収するための適切な変換を行う必要がある。MIROC の大気海洋モデルを例にすると，大気モデルは緯度経度に沿った規則的な矩形格子を持つのに対して，海洋モデルは極点を陸上に移動させた変則的な矩形格子を持つ。更に，大気モデルが全球をくまなく覆っているのに対して，海洋モデルは陸域を除いた格子のみが意味のある値を持っている。従って，情報の交換に際してはこのような格子の差違を補正する補間計算が必要となる。時間的にも大気モデルと海洋モデルでは代表的なシミュレーションの時間刻み幅（ΔT）が異なっており，大気側のΔT が海洋に比して短いため，大気から海洋への情報は大気の各時間ステップの平均値が渡されるようになっている。これらの状況を模式的に表したのが図１である。図中，Atm. Step は大気モデルの計算，Land Step は陸面モデル，River Step は河川モデル，Ocean Step は海洋モデルの計算を表す。大気モデルと海洋モデルは個別のバイナリとして並列に実行され，各データ交換ステップで前ステップまでの計算結果が相互に送受信される。一方，大気と陸面や河川では同一時間ステップ内で順番にデータが渡され逐次的に計算が実行される。ここで，T_N，T_{N+1}，T_{N+2} は大気モデルと海洋モデルがデータ交換する大きな時間刻み幅を表している。一方，大気モデル(Atm. Step)の中で青い太線で表されるのが大気モデルの細かい時間刻み幅である。大気モデルから海洋モデルへ送られるデータが時間平均値の場合は，細かい時間刻み幅毎の値で平均値が計算される。

ここでは気候シミュレーションを例に連成計算の基本的な概念について説明したが，現代の並列計算環境下における連成計算の基本的な要素は 1)複数のモデルコンポーネントが情報（データ）を交換しながら計算を進める。2)情報交換に際しては時空間構造の差違を補整する計算が実行される。という２点に集約されると考えられる。一方，情報交換や時空間構造の補正の具体的な方法については分野や用途によって異なる対応が要求される。代表的な例として空間構造の補整（補間計算）を考える。大気モデルと海洋モデルを連成して気候のような長期的な現象を計算する場合には，量の保存が厳密に保たれることが要求される（そうでないと海が干上がったりする）。そのために，補間計算には各格子点が代表する面積に応じて値を比例配分する面積重み法が用いられるが[5]，その際に更に問題を複雑にしているのは海陸分布の存在である。海洋モデルが陸域を除いた格子で意味のある値を持つことは前述の通りであるが，大気モデル側の格子が海洋モデル格子の陸と海の境界をまたぐ場合には海陸の割合も考慮した上で

値を適切に配分する必要がある。一方で厳密性を要求されないような空間補間が許容される事例もあり得る。例えば，地震動から構造物の振動を計算するような場合，構造物からの地震動への影響は無視しうるため情報の伝達方向は一方向だけで良く，物理量の厳密な保存は要求されない。そのため，この例では直方体で構成される地震モデルの格子から単純な線形内挿で構造物モデルの格子点値を求めている[6]。

本稿で解説するカプラ h3-Open-UTIL/MP は特定の分野に特化することなく様々な用途に用いられることを目標としており，上記のような計算方法の差違や更には格子構造の差違に対して柔軟に対応されることが求められ，かつ実現している。そのための仕組みについては第 4 節以降で詳述する。

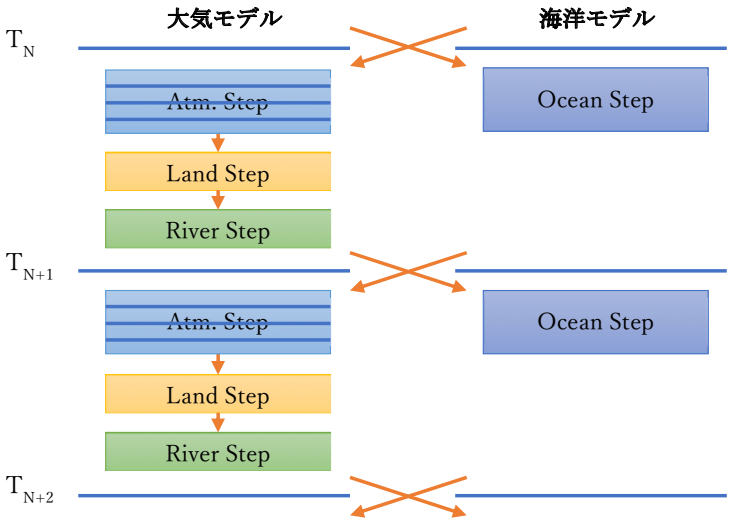


図 1　MIROC における大気モデル・海洋モデル他の連成方法

3　MPI 環境における連成計算

本節では MPI 環境における連成計算の方法について説明する。前提として，連成対象となる個々のモデルコンポーネントは既に MPI 並列化されているものとし，複数のモデルコンポーネントを連成する場合の基本的な概念や手続きについて解説する。

MPI で複数のプログラムを実行する方法には 2 通りの方法がある。ひとつは Master/Worker 方式であり，この場合，マスタとなるプロセスが子プロセスを生成する MPI コマンド(mpi_comm_spawn)をコールする。例えばカプラが独立したプロセスとして最初に起動され，そこから連成対象となるモデルプロセスを生成する場合である。もうひとつは連成対象となるプログラムを一度に起動する方式である。具体的には下の例のように mpiexec（もしくは類似）コマンドに対して複数のプログラムを引数として与えるようにする。

```
mpiexec -np 8 app1.exe : -np 4 app2.exe
```

これら 2 通りの方式では起動されるプログラムの MPI 情報が異なる。簡単なテストプログラム

で具体例を示す。まず Master/Worker 方式のテストプログラムを図 2 に示す。Master 側で MPI サブルーチン mpi_comm_spawn をコールし worker プログラムを起動している。３番目の引数が起動されるプロセス数で，例では５プロセスの worker が起動される。次いでサブルーチン mpi_comm_rank をコールして MPI のデフォルトコミュニケータである MPI_COMM_WORLD を引数として自プロセスのランク番号を取得し出力している。

```
program master
   use mpi
   implicit none
   integer :: my_rank
   integer :: inter_comm
   integer :: ierror

   call mpi_init(ierror)

   call mpi_comm_spawn("./worker", MPI_ARGV_NULL, 5, &
         MPI_INFO_NULL, 0, MPI_COMM_WORLD, &
         inter_comm, MPI_ERRCODES_IGNORE, ierror)

   call mpi_comm_rank(MPI_COMM_WORLD, my_rank, ierror)

   write(*,*) "my_rank = ", my_rank

   call mpi_finalize(ierror)

end program maste
```

```
program worker
   use mpi
   implicit none
   integer :: my_rank
   integer :: ierror

   call mpi_init(ierror)

   call mpi_comm_rank(MPI_COMM_WORLD, my_rank, ierror)

   write(*,*) "my_rank = ", my_rank

   call mpi_finalize(ierror)

end program worker
```

図 2　連成計算テストプログラム（上図：Master，下図：Worker）

このプログラムを mpiexec -n 2 ./master コマンドで実行した場合の結果が図 3 左図である。一方，worker プログラムを worker1 と worker2 として mpiexec -n 2 ./worker1 : -n 5 ./worker2 コマンドで実行した場合の結果が図 3 右図である。結果からわかるように Master/Worker 方式の実行では起動されるプログラム毎に独立した MPI_COMM_WORLD が割り当てられ，その中でランク番号が取得されているのに対し，複数のプログラムを同時に起動する場合はすべてのプロセスで MPI_COMM_WORLD が共有されている。

master_rank =	0
master_rank =	1
worker_rank =	0
worker_rank =	1
worker_rank =	2
worker_rank =	3
worker_rank =	4

worker_rank =	2
worker_rank =	5
worker_rank =	6
worker_rank =	4
worker_rank =	0
worker_rank =	3
worker_rank =	1

図 3 連成計算テスト結果（左図：Master/Worker 方式，右図：同時起動方式）

上の例で示したように Master/Worker 方式の場合はプログラム間でコミュニケータが共有されない。そのためプログラム間の通信を行う際には mpi_comm_spawn の 7 番目の引数で MPI から返されるプログラム間コミュニケータ inter_comm を用いることになる。一方，同時起動方式では MPI_COMM_WORLD が共有されるためプログラム間通信は MPI_COMM_WORLD を用いれば良いが，一方で各プログラム内の通信を混乱なく実行するためには MPI サブルーチン mpi_comm_split を用いてコミュニケータを分割し，プログラム毎に分割されたコミュニケータを用いるようにする必要がある。これらの状況を簡単に示したのが図 4 である。両者の手法には一長一短があるが，Master/Worker 方式の場合，複数の Worker が起動される状況では Worker 間の通信手段が限られる（ないわけではないが相応の手続きを要求される）という問題がある。従って Mater/Worker 方式で複数のモデルコンポーネントを連成する場合には，図に示すように Master を経由して通信を行うのが一般的である。この際にカプラプログラムを Master とすることで補間計算やデータ通信の管理などをモデルとは別のプロセスで処理できるようになる。このことは連成に際しての計算性能向上（実行時間短縮）要因となり得るが，モデルプロセスに比して計算負荷が軽いカプラに専用プロセスを割り当てることは，全体としては計算効率の低下に繋がる懸念がある。一方，同時に起動する場合はカプラに個別のプロセスを割り当てることなく，モデルプロセス同士で直接データ交換することが可能である。この場合カプラはライブラリとしてモデルプロセスにリンクされ実行されることになる。h3-Open-UTIL/MP はモデルプロセスを同時に起動するタイプのカプラである。以下では，この形式での実行におけるデータ通信と補間計算について解説する。図 4 では暗黙の前提としてモデルの各プロセスは相手モデルの各プロセスと直接データ交換を行う Local To Local 通信を行うとして描かれている。相互に複数のプロセスで実行されるモデル間でデータを送受信する方法として，より単純なのは 1)送信側の代表プロセスに送信側各プロセスのデータを集約，2)送信側の代表プロセスから受信側の代表プロセスへ集約したデータを送受信，3)受信側の代表プロセスから受信側各プロセスへデータを分配，という方法である。この際，送信側の代表プロセスもしくは受信側の代表プロセスで全領域のデータを対象として補間計算を行う。この方法は実装が容易だがデータ通信量，プロセスあたりの演算量ともに Local To Local の場合に比して大きくなり，性能上のボトルネックが生じる可能性がある。加えて，大規模並列計算ではメモリ容量の観点から全領域のデータをひとつのプロセスで保持すること自体が不可能な場合もあり得るため，単純な手法では適用できる条件が限られる。一方，Local To Local 通信方法は性能面では優位であるが，適切な格子点データを適切なプロセス間で送受信するためのアルゴリズムが煩雑で実装が難しい場合が(もちろん難易度は双方の格子の形状や対応関係に依存するが)ある。カプラの存在意義のひとつはこの点にあり，本稿で紹介する h3-Open-UTIL/MP においても単純な API の呼び出しだけで Local To Local 通信を実現できるよう

に設計・実装されている。

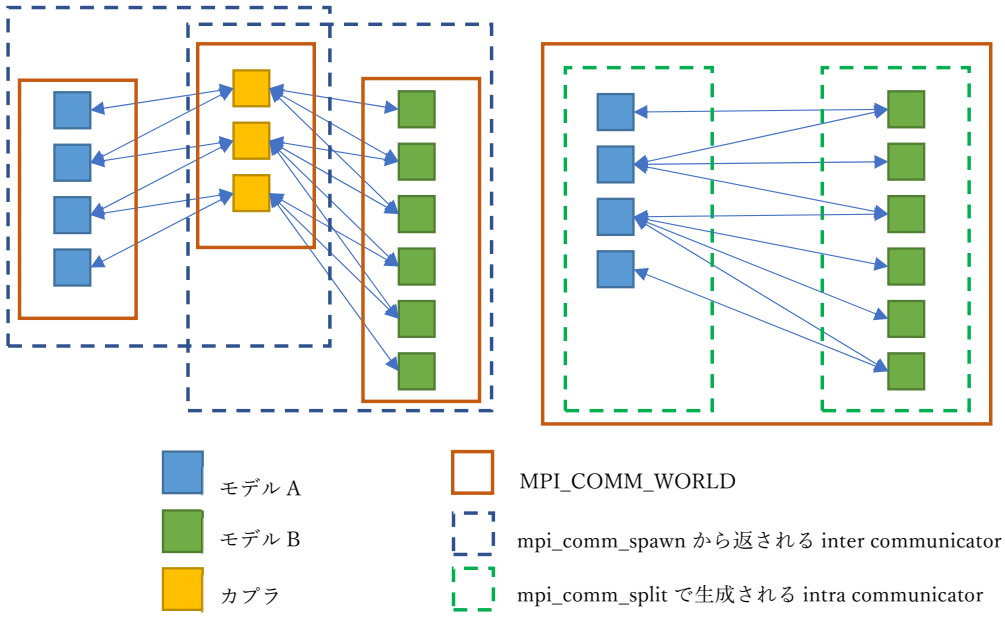


図 4 プログラムの起動方法とコミュニケータの関係（左図：Master/Worker 方式，右図：同時起動方式）

次に格子変換機能について説明する。異なる格子形状のモデル間で情報を交換するには受信側の格子点値を送信側の複数の格子点の値から内挿する必要がある。この内挿計算には前項で述べたように，面積で比例配分する方法や線形内挿する方法など複数の手法がある。しかしながらこれらの手法の違いが影響するのは内挿時の格子点の選択と係数に対してであり，計算アルゴリズムとしては同等である。すなわち受信側のある格子点値 R は送信側の複数(N 個)の格子点値 S(i)と係数 C(i)から次の式のように計算される。

$$R = \sum_{i=1}^{N} S(i) * C(i)$$

すべての受信側格子点に対して送信側の格子点番号とそれに対応する係数を列挙したものをここでは補間テーブルと呼ぶ。ここで各モデルの格子形状が時間的に変わらないとすると，補間テーブルも時間変化しないため事前に計算しておくことが可能である。この補間テーブルおよびモデルの各プロセスが担当する格子の格子点番号の情報が与えられれば，モデルの各プロセスは 1)自身が保持する各格子点値を相手モデルのどのプロセスに送信すれば良いか，2)相手モデルのどのプロセスから何番の格子点値を受信すれば良いか，が決定でき適切な Local To Local のデータ交換と各プロセスでの領域毎の補間計算が可能になる。h3-Open-UTIL/MP においては，これらの情報を入力とすることによって各モデルの格子形状そのものには依存しないで連成計算を実行することを可能としている。

4　h3-Open-UTIL/MP

4.1 h3-Open-UTIL

h3-Open-UTIL はスーパーコンピューティングニュース前号記事[7]で解説された革新的ソフトウェア基盤「h3-Open-BDEC」[8,9]を構成するソフトウェアユニットのひとつである。h3-Open-UTIL ユニットの目的は 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(S+D+L)」融合を容易に実現するための環境を提供することである[10,11,12]。h3-Open-UTIL だけでなく h3-Open-BDEC の各ユニットを有機的に協調させ利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。

4.2 h3-Open-UTIL/MP の構造

h3-Open-UTIL/MP は連成計算を容易に実現するためのソフトウェア(カプラ)である。対象となるのは MPI を用いて領域分割により並列化されたソフトウェア群であり，今のところ Fortran と Python の API が提供されている。h3-Open-UTIL/MP のプログラム構造を図 5 に示す。図は Fortran プログラムと Python プログラムを連成する例である。h3-Open-UITL/MP は最下層に連成ライブラリ Jcup を用いている[13]。Jcup は連成されるモデルコンポーネントの管理や MPI ルーチンコールなどの基本的な機能を提供する。その上に h3-Open-UTIL/MP のプログラムが構築される。h3-Open-UTIL/MP のプログラムは Python の API を除いて Fortran の module 群で構成されており，Fortran プログラムから用いる場合は API のモジュールを use して API サブルーチンをコールする。Python に関連したプログラムについては次号の記事で説明する予定である。

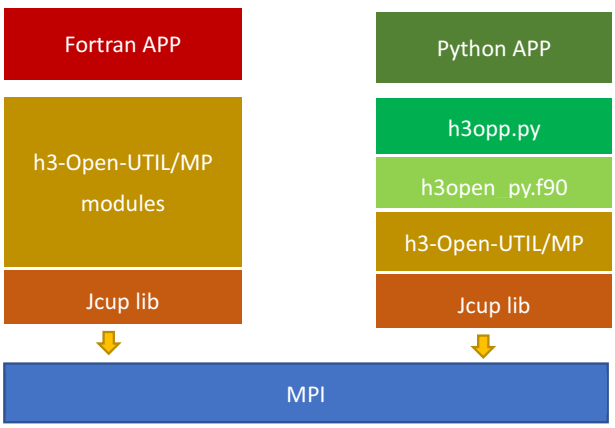


図 5　h3-Open-UTIL/MP のプログラム構造

4.3 h3-Open-UTIL/MP の機能

前項で述べたように連成計算はモデルコンポーネント間で情報(データ)を交換しながら計算を進めることであり，情報の交換に際しては時空間構造の差違を補正する必要がある。h3-Open-UTIL/MP は MPI でデータ並列化されたモデルコンポーネントに対して連成計算を可能とするソフトウェアである。制約条件としては 1)データ交換の時間間隔はデータ毎に一定間隔でなければならない。2)モデルコンポーネントの格子は一意に番号づけられ相対的な位置が時間変化しな

い。の 2 つである。逆に言えば，この要求を満たすモデルや計算条件であればどのようなモデルでも適用可能である。なおデータ交換の時間間隔は一定である必要があるが，その間の積分時間ステップは変化しても構わない。この様子を模式的に示したのが図 6 である。図の例では 3 つのモデルコンポーネントがデータを交換しており，モデル A のデータ交換間隔の間に 3 ステップと 4 ステップの時間ステップを計算している。時間構造の補正として h3-Open-UTIL/MP にはデータの時間平均値を計算する機能がある。平均値計算のアルゴリズムは単純に各時間ステップの ΔT で比例配分する方法であり，式で表すと次のようになる。ここで Dmean は時間平均値，D(t) は時間ステップ t の値， ΔT(t)は時間ステップ t での ΔT，Tex はデータ交換間隔である。先述のようにデータ交換間隔 Tex は一定値でなければならず，またモデルの時間ステップの刻みと一致している，すなわち Tex=Σ ΔT(t)でなければならない。なお平均値を交換するか瞬間値を交換するかは設定で切り替えることができる。設定方法については後述する。

$$Dmean = \sum_{t=1}^{N} D(t) * \Delta T(t)/Tex$$

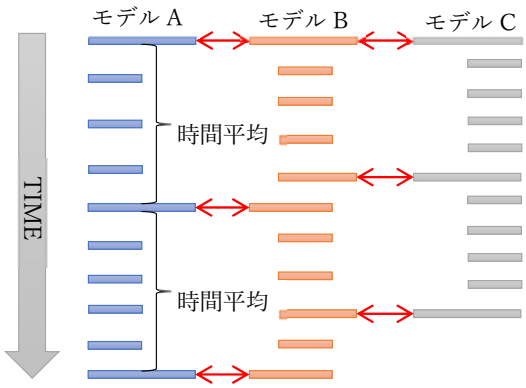


図 6 時間ステップとデータ交換の関係

時間積分ループ中でのデータ交換に関わるサブルーチンコールの様子を図 7 に示す。図で青の四角は時間積分ループの開始と終了，グレーの四角はモデル計算コードを表す。オレンジの四角が h3-Open-UTIL/MP の API ルーチンコールである。時間積分ループ中で連成計算に関わるルーチンコールは h3ou_set_time, h3ou_put_data, h3ou_get_data の 3 種類のみである。h3ou_set_time はモデル時刻をカプラに教えるためのルーチンで，時間積分ループの冒頭付近で呼ばれることが期待されている。h3ou_put_data は相手モデルに送信するデータをカプラに与えるサブルーチン，h3ou_get_data は相手モデルから受信したデータを取得するためのルーチンで，これらのサブルーチンは h3ou_set_time が呼ばれた後であればプログラムの任意の箇所でコールできる。データ送受信や補間計算が実際に行われるのはサブルーチン h3ou_set_time の内部である。モデル A からモデル B へデータが送受信される際のデータの流れを図 8 に示す。図中，青枠の四角は利用者がコールするルーチンを表す。緑枠とオレンジ枠はカプラ内部での動作である。緑とオレンジの 2 種類の枠の相違については次号で解説する。赤枠はカプラ内部で保持されるデータを表す。h3ou_put_data でカプラに与えられたデータはカプラ内部のデータバッファに保持される。

h3ou_set_time がコールされた時点で，当該データがその時刻の送受信対象データかどうかが判定され，送受信データであればバッファから取り出し(get_from_buffer)相手モデルのプロセス順に応じてデータを並び替える（rearrange_send_data)。送信と受信には MPI_Isend, MPI_Irecv が用いられる。モデル B で受信されたデータは再度並び替えられ(rearrange_recv_data)，補間計算サブルーチン interpolate_data に渡される。補間計算で得られたデータはカプラのデータバッファに保持され(put_to_buffer)，API サブルーチン h3ou_get_data がコールされた時点でバッファから取り出され呼び出し側に渡される。

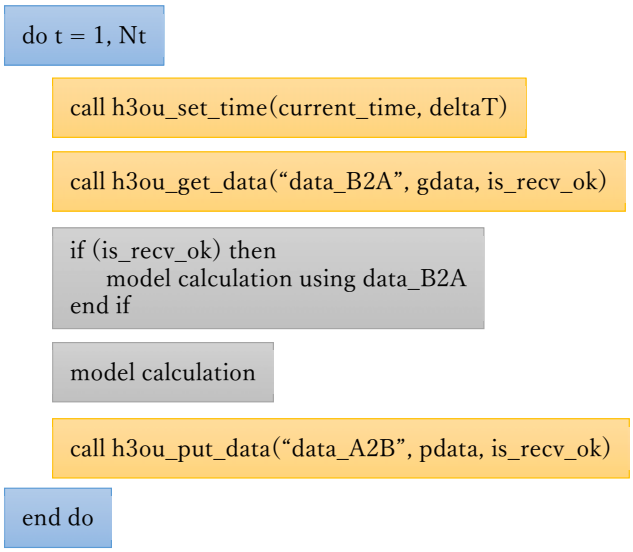


図 7 時間積分ループ中でのカプラ API コールの模式図

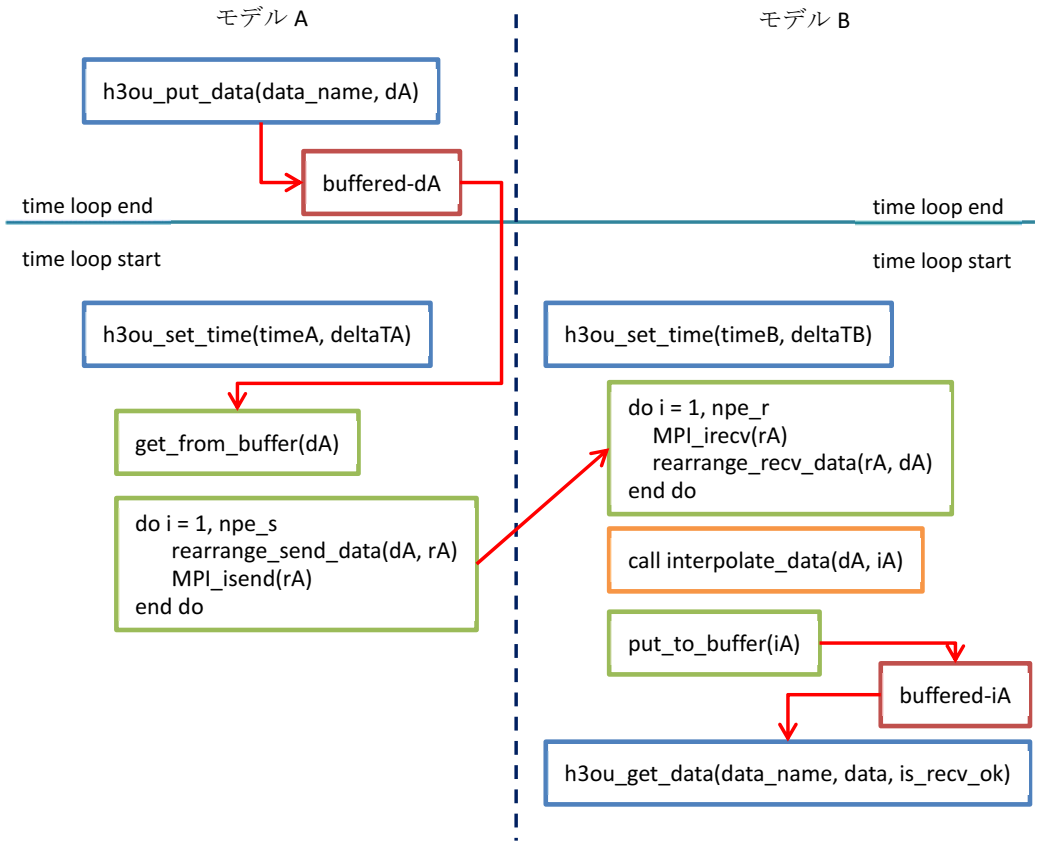


図 8 カプラ内部での送受信データの流れ

（青枠は利用者コールルーチン，緑とオレンジ枠はカプラの内部動作を表す）

ここまで述べたのは h3-Open-UTIL/MP の基本的な機能であり，カプラとして特に新規性のあるものではない。しかしながら h3-Open-UTIL/MP は他のカプラにはない独自の機能を備えている。それらは，

1) アンサンブル連成機能
2) Python アプケーション連成機能
3) 異機種間連成機能

である。これら独自機能については次号の記事で詳細を説明する予定である。以下の節では h3-Open-UTIL/MP の基本的な使用方法について具体例に基づきながら説明してゆく。

5 h3-Open-UTIL/MP の使用方法

この節では具体的な事例に則り h3-Open-UTIL/MP の基本機能の使用方法について説明する。対象としているマシンは Wisteria/BDEC-01 の Odyssey である。サンプルコードおよび実行シェルスクリプトは/work/share/h3-Open-UTIL-MP/src/sample/h3ou/test1（ディレクトリ名は変更になる可能性があります。確定版は次号記事でお知らせいたします）以下の src および run ディレ

クトリにあるので適宜参照されたい。このサンプルでは app1, app2, app3 の 3 つのアプリケーション（モデル）が相互にデータを交換しながら時間積分を進める構成になっている。状況を単純にするため，モデルの格子は同一で積分時間間隔やステップ数も同一としている。

5.1 設定ファイルの準備

h3-Open-UTIL/MP はカプラの動作と交換データの詳細情報を設定するために設定ファイルを用いる。テスト計算プログラムで用いられている設定ファイルを図 9 に示す。セクション h3ou_coupling はカプラ全体の動作を規定するセクションで，今のところ設定可能なのはログファイルの出力レベルである。log_level は”SILENT”,”WISPER”,”LOUD”の３通りが設定可能であり SILENT はログを出力せず LOUD は詳細なログ情報を出力する。LOUD モードは大量のログを出力するためテストやデバッグ時に使用するモードである。debug_mode はログを標準エラー出力に出力するかどうかのフラグである。セクション h3ou_var は２通りの記述形式が定められている。ひとつはデータ交換するモデル名と格子名のセットであり，comp_put, comp_get, grid_put, grid_get のそれぞれに送信モデル名，受信モデル名，送信格子名，受信格子名を与える。この組み合わせに対して以下に記述される交換変数の設定が適用される。h3ou_var セクションのもう一つの記述形式は交換データの設定に関するものである。var_put は送信側データ名，var_get は受信側データ名である。grid_intpl_tag は補間計算時にデータを識別するためのタグで，特に必要ない場合は１を与える。なお，このこのタグを参照する事例については次号で解説する予定である。intvl はデータ交換間隔で単位は秒である。lag は連成計算が並列に行われるか逐次的に行われるかを設定するフラグで，一般的な並列実行の場合は-1 を与える。layer は当該データの鉛直層数である。後述する格子番号設定サブルーチンでも鉛直層数を与えるが，これは格子が取り得る最大の鉛直層数であり，データ毎にはこれを超えない任意の層数を設定可能である。flag はデータを時間平均するかどうかの設定で時間平均する場合は”AVR”を，瞬間値を交換する場合は”SNP”を設定する。

```
# log_level : "SILENT" or "WHISPER" or "LOUD", default = "SILENT"
# debug_mode : .true. or .false., default = .false.
&h3ou_coupling
  log_level = "SILENT"
  debug_mode   = .false.
&end

&h3ou_var   comp_put = "app2",   comp_get = "app1",
            grid_put = "app2_grid", grid_get = "app1_grid", /
&h3ou_var   var_put   = "app2_to_app1",   var_get   = "app2_to_app1", grid_intpl_tag = 1, intvl=720,   lag=-1,
layer=40, flag='AVR' /

&h3ou_var   comp_put = "app1"         ,   comp_get = "app2",
            grid_put = "app1_grid" ,   grid_get = "app2_grid", /
&h3ou_var   var_put   = "app1_to_app2",   var_get   = "app1_to_app2", grid_intpl_tag = 1, intvl=720,   lag=-1,
layer = 40, flag='AVR' /

&h3ou_var   comp_put = "app3"         ,   comp_get = "app1",
            grid_put = "app3_grid" ,   grid_get = "app1_grid", /
&h3ou_var   var_put   = "app3_to_app1"       ,   var_get   = "app3_to_app1"       , grid_intpl_tag = 1, intvl=720,
lag=-1, layer=40, flag='AVR' /

&h3ou_var   comp_put = "app1"         ,   comp_get = "app3",
            grid_put = "app1_grid" ,   grid_get = "app3_grid", /
&h3ou_var   var_put   = "app1_to_app3"       ,   var_get   = "app1_to_app3"       , grid_intpl_tag = 1, intvl=720,
lag=-1,   layer = 40, flag='AVR' /

&h3ou_var   comp_put = "app3"         ,   comp_get = "app2",
            grid_put = "app3_grid" ,   grid_get = "app2_grid", /
&h3ou_var   var_put   = "app3_to_app2"       ,   var_get   = "app3_to_app2"       , grid_intpl_tag = 1, intvl=720,
lag=-1, layer=40, flag='AVR' /

&h3ou_var   comp_put = "app2"         ,   comp_get = "app3",
            grid_put = "app2_grid" ,   grid_get = "app3_grid", /
&h3ou_var   var_put   = "app2_to_app3"       ,   var_get   = "app2_to_app3"       , grid_intpl_tag = 1, intvl=720,
lag=-1,   layer = 40, flag='AVR' /
~
```

図 9 h3-Open-UTIL/MP の設定ファイル例

5.2 API ルーチンコールの概要

h3-Open-UTIL/MP の API を連成対象モデルに組み込む際の手続きは，1)カプラの初期化，2)格子の設定，3)補間テーブルの設定，4)初期時刻の設定，5)第 0 ステップデータの put，6)現在時刻と ΔT の設定，7)データの get，8)データの put，9)連成の終了，の 9 ステップに分けられる。このうち 1)から 4)までが初期設定，6)7)8)が時間積分ループ内の手続きである。これらの手順を図 10 に示す。図では複数回呼び出し可能あるいは設定に従い複数回コールする必要のあるルーチンを水色で，一度だけコールされるルーチンをピンクで示している。なお時間積分ループ中のルーチンコールについては時間ステップ毎に 1 度だけコールするか複数回コール可能かで色分けされている。各ルーチンの引数などについては次節で説明される。初期化部分でコールしている h3ou_get_mpi_parameter は図 4 右図の緑の破線で表されているモデル毎のコミュニケータおよびこのコミュニケータのグループやサイズ，プロセスのランク番号を取得するサブルーチン

である。連成対象プログラムが MPI 並列化されている場合，モデルがこれまで用いているコミュニケータはこのルーチンから得られたコミュニケータに置き換える必要がある。

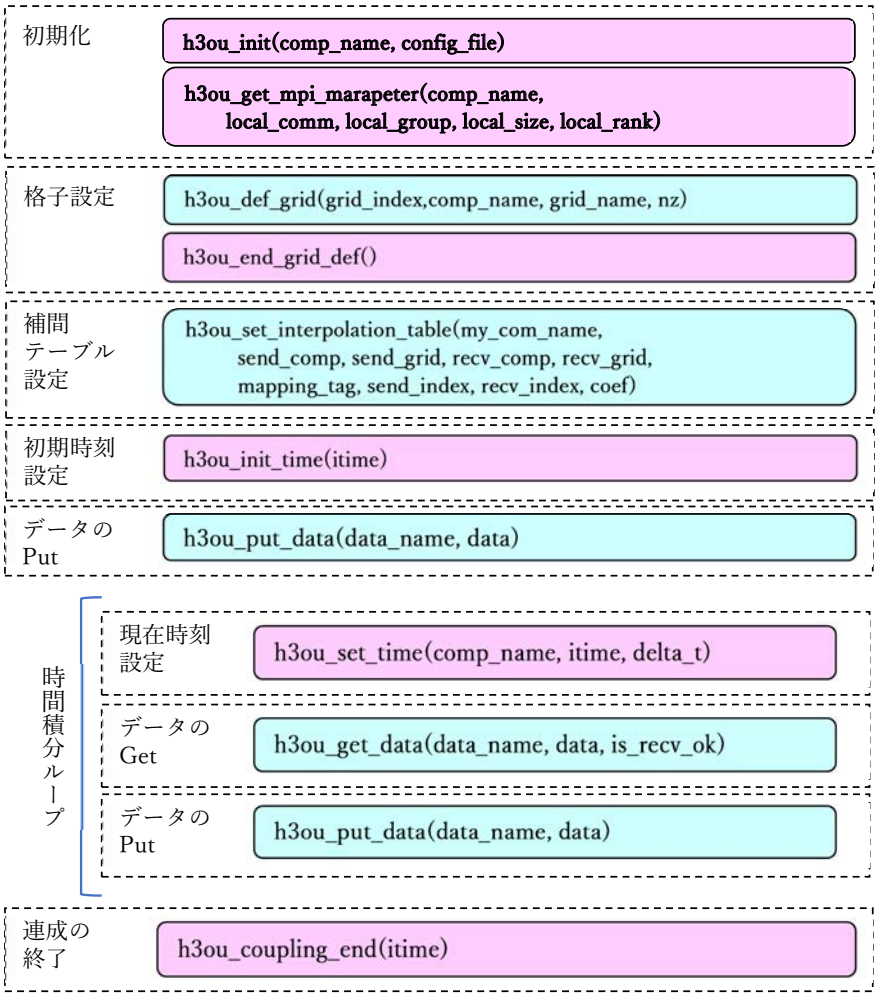


図 10 カプラ API ルーチンコールの概要

サンプルプログラムのメイン部分を図 11 に示す。9 行目のサブルーチン init_common 中で h3ou_init と h3ou_get_mpi_parameter がコールされカプラの初期化と MPI 情報の取得を行っている。次いで 11 行目の cal_and_def_grid でプロセス毎の格子点番号を計算し h3ou_def_grid でカプラに渡した上で，h3ou_end_grid_def をコールしている。13 行目から 24 行目までが補間テーブルの設定に関わるルーチンコールである。ここでは app2→app1，app1→app2，app3→app1，app1 → app3 の順序で補間テーブルを計算し補間テーブルの設定ルーチン h3ou_set_interpolation_table をコールしている。26 行目で h3ou_init_time をコールした後，28 行，29 行めで第 0 ステップのデータをカプラに渡している。時間積分最初のステップの h3ou_set_time で設定ファイルに記述されたすべてのデータが交換されるため，第 0 ステップの h3ou_put_data は自モデルが送信すべきすべてのデータについてコールする必要がある。時間積分ループ内の h3ou_get_data， h3ou_put_data はまとめて記述されているが，実際には必要に応

じてプログラムのどの箇所からでも順不同にコール可能である。最後に 43 行目で h3ou_coupling_end をコールし連成を終了している。

```
 1  program app1
 2    use mpi
 3    use common
 4    implicit none
 5    integer :: int_array(1)
 6    integer :: ierror
 7    integer :: i
 8
 9    call init_common(APP1_NAME, 1)
10
11    call cal_and_def_grid()
12
13    call cal_and_set_map(APP1_NAME, &
14                         APP2_NAME, APP2_GRID, APP2_SIZE, &
15                         APP1_NAME,    APP1_GRID,    APP1_SIZE)
16    call cal_and_set_map(APP1_NAME, &
17                         APP1_NAME,    APP1_GRID,    APP1_SIZE, &
18                         APP2_NAME, APP2_GRID, APP2_SIZE)
19    call cal_and_set_map(APP1_NAME, &
20                         APP3_NAME, APP3_GRID, APP3_SIZE, &
21                         APP1_NAME,    APP1_GRID,    APP1_SIZE)
22    call cal_and_set_map(APP1_NAME, &
23                         APP1_NAME,    APP1_GRID,    APP1_SIZE, &
24                         APP3_NAME, APP3_GRID, APP3_SIZE)
25
26    call init_time(time_array)
27
28    call put_data_2d("app1_to_app2", 1, i)
29    call put_data_2d("app1_to_app3", 1, i)
30
31    do i = 1, APP1_STEP
32      call set_time(APP1_NAME, time_array, APP1_DELTA_T)
33      call get_data_2d("app2_to_app1")
34      call get_data_2d("app3_to_app1")
35
36      call sleep(1)
37
38      call put_data_2d("app1_to_app2", 1, i)
39      call put_data_2d("app1_to_app3", 1, i)
40
41    end do
42
43    call finalize_common()
44
45  end program app1
```

図 11 サンプルプログラムのメイン部分

6　h3-Open-UTIL/MP の API

この節では h3-Open-UTIL/MP の API について一般的な連成に用いられる主立ったものを説明する。

6.1 初期設定 API

表 1 に初期設定に関わる h3-Open-UTIL/MP の API を示す。基本的な連成に用いられるのは h3ou_init と h3ou_get_mpi_parameter の 2 つである。

表 1. 初期設定用 API

ルーチン名	概要
h3ou_init	**カプラの初期化**
h3ou_get_mpi_parameter	**ローカルな MPI 情報を取得する**

これらのルーチンの引数と意味は下のようになる。

(1) カプラ初期化 API

```
subroutine h3ou_init(comp_name, config_file_name, num_of_ensemble)
  character(len=*), intent(IN)   :: comp_name        ! モデルコンポーネント名
  character(len=*), intent(IN)   :: config_file_name ! 設定ファイル名
  integer, intent(IN), optional  :: num_of_ensemble  ! アンサンブル数
```

- カプラの初期化プロセスを実行する。すべてのプロセスが必ず最初に呼ぶ必要がある。カプラ内部でモデルコンポーネント名に従って MPI コミュニケータを生成する。num_of_ensemble は optional 引数であり，次号で説明するアンサンブル結合の場合のみ意味を持つ。

(2) MPI 情報取得 API

```
subroutine h3ou_get_mpi_parameter(comp_name, comm, group, size, rank)
  character(len=*), intent(IN)  :: comp_name  ! モデルコンポーネント名
  integer, intent(OUT)          :: local_comm ! Local なコミュニケータ
  integer, intent(OUT)          :: group      ! Local な MPI グループ
  integer, intent(OUT)          :: size       ! local_comm に所属するプロセス数
  integer, intent(OUT)          :: rank       ! local_comm 内のランク番号
```

- カプラ内部で生成された Local な MPI 情報を取得する。local_comm がモデル内部で使用されるコミュニケータである。group, size, rank はモデルローカルなグループ ID, ローカルなプロセス数, ローカルな自プロセスのランク番号である。

6.2 格子設定 API

表 2 に格子設定用 API を示す。h3ou_def_grid は個々の格子の格子点番号を設定する。h3-Open-UTIL/MP はひとつのモデルコンポーネントに対して複数の格子が設定可能であり，h3ou_def_grid も格子の数に対応して複数回呼び出し可能である。h3ou_end_grid_def は格子設定の終了をカプラに通知する。

表 2. 格子設定用 API

ルーチン名	概要
h3ou_def_grid	**格子点番号の設定**
h3ou_end_grid_def	**格子点番号設定終了**

これらのルーチンの引数と説明は下記の通りである。

(1) 格子設定 API

```
subroutine h3ou_def_grid(grid_index, comp_name, grid_name, num_of_layer)
  integer, intent(IN)           :: grid_index(:)   ! 各プロセスの格子点番号
  character(len=*), intent(IN)  :: comp_name       ! モデルコンポーネント名
```

```
character(len=*), intent(IN) :: grid_name        ! 格子名
integer, intent(IN)          :: num_of_ensemble  ! 鉛直格子数
```

- 格子の格子点番号を設定する。このサブルーチンはデータ交換に関わる格子の数だけ複数回コールしなければならない。grid_index は各プロセスが担当する格子のグローバルな格子点番号である。番号は自然数でなければならない。格子点番号は連続している必要はなく番号に空きがあることは許される。ただし番号の重複は許されない。カプラは番号の重複をチェックしないため注意が必要である。comp_name と grid_name は格子が所属するモデル名と格子名である。nz は補間計算が水平 2 次元で行われる際に，3 次元データの鉛直格子の数を与える。この場合，格子点番号 grid_index は水平格子の分だけを与えれば良い。逆に 3 次元空間で補間計算が必要な場合は 3 次元分の大きさの grid_index を与え nz=1 とすればよい。

(2) 格子設定終了通知 API

```
subroutine h3ou_end_grid_def()
```

- 格子情報設定の終了をカプラに通知する。引数はなく h3ou_def_grid のコールの後に一度だけ呼び出す。このサブルーチンの内部で交換データの設定など複数の初期設定が実行される。

6.3 補間テーブル設定 API

表 3 に補間テーブル設定 API を示す。個々の引数については次に説明する。重要なのは，このサブルーチンがモデルコンポーネント間で各種情報の通信を行っているということである。そのため，送信側モデルと受信側モデルでルーチンコールが正しく対応している必要がある。呼び出しの順序を誤り対応関係が崩れると通信がデッドロックするなどの不具合が生じるため注意しなければならない。

表 3. 補間テーブル設定用 API

ルーチン名	概要
h3ou_set_interpolation_table	**補間テーブルの設定**

このルーチンの引数と説明は下記の通りである。

(1) 補間テーブル設定 API

```
subroutine h3ou_set_interpolation_table(my_name,
                                        send_comp, send_grid,
                                        recv_comp, recv_grid,
                                        mapping_tag,
                                        send_index, recv_index, coef)
  character(len=*), intent(IN)       :: my_name       ! 自モデル名
  character(len=*), intent(IN)       :: send_comp     ! 送信モデル名
  character(len=*), intent(IN)       :: send_grid     ! 送信格子名
  character(len=*), intent(IN)       :: recv_comp     ! 受信モデル名
  character(len=*), intent(IN)       :: recv_grid     ! 受信格子名
  integer, intent(IN)                :: mapping_tag   ! 補間テーブル識別タグ
  integer, optional, intent(IN)      :: send_index(:) ! 送信側格子点番号
  integer, optional, intent(IN)      :: recv_index(:) ! 受信側格子点番号
  real(kind=8), optional, intent(IN) :: coef(:)       ! 補間係数
```

- 補間テーブルを設定する。前述のようにこのサブルーチンは送信側と受信側で正しく

対応している必要がある。6 番目の引数 mapping_tag は同じモデルと格子の組み合わせで複数の補間テーブルを使い分ける際の識別子である。複数の補間テーブルを設定しない場合は 1 を与える。格子の格子点番号を設定する。send_index, recv_index, coef が補間テーブルである。配列の大きさは 3 つの引数で同一でなければならない。これら 3 引数は optional となっており，送信側か受信側のいずれかで与えれば良い。また引数が意味を持つのはルート（0 番）プロセスのみであるためメモリを節約する場合は 0 番以外は大きさ 1 の配列を与えるようにするとよい。

6.4 時刻設定 API

表 4 に時間設定 API を示す。h3ou_init_time は積分の初期時刻を設定する。このサブルーチンは時間積分開始前に一度だけコールされる。h3ou_set_time は時間積分ループ中で毎ステップコールされる（しなければならない）。第 4 節で述べたように h3ou_set_time の中でデータ交換や補間計算など連成に関わる多くの処理がなされる。

表 4. 時間設定 API

ルーチン名	概要
h3ou_init_time	**初期時刻の設定**
h3ou_set_time	**現在時刻とΔT の設定**

これらのルーチンの引数と説明は下記の通りである。

(1) 初期時刻設定 API

```
subroutine h3ou_init_time(time_array)
  integer, intent(IN) :: time_array(6) ! 年月日時分秒の配列
```

- h3ou_init_time は積分初期時刻を設定する。引数 time_array は年月日時分秒を表す大きさ 6 の整数配列である。このサブルーチンは時間積分開始前に 1 回だけコールされる。

(2) 現在時刻と ΔT 設定 API

```
subroutine h3ou_set_time(comp_name, time_array, delta_t)
  character(len=*), intent(IN) :: comp_name     ! モデル名
  integer, intent(IN)          :: time_array(6) ! 現在時刻の配列
  integer, intent(IN)          :: delta_t       ! ΔT
```

- h3ou_set_time は時間積分ループの中で現在時刻と ΔT を設定する。comp_name は当該モデル名，time_array は現在時刻を表す年月日時分秒の配列，delta_t はそのステップの ΔT である。カプラ内部では delta_t の積算時間（秒）でデータ交換判定などを行っている[1]。従ってこのサブルーチンは基本的に（データ交換を行わないステップであっても）毎時間ステップコールしなければならない。

6.5 データ提供取得 API

表 5 にデータ提供取得 API を示す。これらのサブルーチンは時間積分ループ中で h3ou_set_time

[1] カプラ内部での時間管理は delta_t の積算値（秒単位）が用いられており第２引数の年月日時分秒配列はログ出力などでのみ参照されるようになっている。これはユリウス暦とグレゴリウス暦の違いなどカレンダ計算に関する面倒な問題を回避するためである。

がコールされた後なら任意の箇所でコール可能である。また第一ステップで交換するデータをカプラに与えるため，時間積分ループ前に交換される全データについて h3ou_put_data をコールしておく必要がある。また時間平均データについては時間ステップ毎のΔT で比例配分しているため h3ou_put_data については毎ステップコールしなければならない。

表 5. データ提供取得 API

ルーチン名	概要
h3ou_init_time	**初期時刻の設定**
h3ou_set_time	**現在時刻とΔT の設定**

これらのルーチンの引数と説明は下記の通りである。

(1) データ提供 API

```
subroutine h3ou_put_data(data_name, data)
  character(len=*), intent(IN) :: data_name            ! 送信データ名
  real(kind=8), intent(IN)     :: data(:) or data(:,:) ! 送信データ
```

- h3ou_put_data はカプラに対して送信データを与える。引数 data_name は送信データ名である。送信データはデータが鉛直層を持たない場合は一次元配列，鉛直層を持つ場合は 2 次元配列を与える。なお一次元目の配列の大きさは h3ou_def_grid の引数 grid_index の大きさと同じでなければならない。

(2) データ取得 API

```
subroutine h3ou_get_data(data_name, data, is_recv_ok)
  character(len=*), intent(IN) :: data_name            ! 受信データ名
  real(kind=8), intent(INOUT)  :: data(:) or data(:,:) ! 受信データ
  logical, intent(OUT)         :: is_recv_ok           ! データ受信フラグ
```

- h3ou_get_data はカプラから受信データを取得する。送信用サブルーチンと同様，data の配列は鉛直層の有無で 1 次元もしくは 2 次元となる。is_recv_ok は当該時間ステップがデータ交換のタイミングかどうかを返す。データ交換のタイミングでない場合は is_recv_ok は.false.が返り data の値は不定となる。従って受信データを使う場合は必ず is_recv_ok を参照し，真となるタイミングでのみ data の値を参照するようにしなければならない。

6.6 終了処理 API

表 6 に終了処理を示す。このサブルーチンは時間積分ループ終了後，プログラム終了前に一度だけコールされる必要がある。

表 6. 終了処理 API

ルーチン名	概要
h3ou_coupling_end	**連成の終了**

このルーチンの引数と説明は下記の通りである。

(3) 終了処理 API

```
subroutine h3ou_coupling_end(itime, is_call_finalize)
  integer, intent(IN), optional :: itime(:)         ! 積分終了時刻
  logical, intent(IN), optional :: is_call_finalize ! MPI 終了フラグ
```

- h3ou_coupling_end はカプラに対して連成の終了を通知する。引数 itime は積分終了時刻で年月日時分秒を表す大きさ６の整数配列である。is_call_finalize はこのサブルーチン内で MPI の終了サブルーチン MPI_finalize を呼び出すかどうかのフラグである。

7 まとめ

本稿では Wisteria/BDEC-01 利用事例として汎用連成ソフトウェア h3-Open-UTIL/MP の基本機能について解説した。本稿の解説に従って h3-Open-UTIL/MP の API を組み込む事で基本的な連成機能は実現可能である。本文で述べたように h3-Open-UTIL/MP にはアンサンブル連成や Python アプリケーション連成[14], h3-Open-SYS/WaitIO[7]と協調した異機種間連成，など従来のカプラにはない機能が実装されている。これらについては次号で紹介する。

参考文献

[1] 司馬遼太郎, 街道をゆく 24「近江散歩，奈良散歩」, 朝日文庫, 249-260

[2] Manabe, S. and Bryan, K., 1969, Climate Calculations with a Combined Ocean-Atmosphere Model, Journal of the Atmospheric Sciences 26, 786-789, https://doi.org/10.1175/1520-0469(1969)026%3C0786:CCWACO%3E2.0.CO;2

[3] Stouffer, R. J., Manabe, S. and Bryan, K., 1989. Interhemispheric asymmetry in climate response to a gradual increase of atmospheric CO2. Nature 342, 660–662.

[4] Tatebe, H., Ogura, T., Nitta, T., Komuro, Y., Ogochi, K., Takemura, T., Sudo, K., Sekiguchi, M., Abe, M., Saito, F., Chikira, M., Watanabe, S., Mori, M., Hirota, N., Kawatani, Y., Mochizuki, T., Yoshimura, K., Takata, K., O'ishi, R., Yamazaki, D., Suzuki, T., Kurogi, M., Kataoka, T., Watanabe, M., and Kimoto, M.: Description and basic evaluation of simulated mean state, internal variability, and climate sensitivity in MIROC6, Geosci. Model Dev., 12, 2727–2765, https://doi.org/10.5194/gmd-12-2727-2019, 2019.

[5] Jones, P.W., 1999, First- and Second-Order Conservative Remapping Schemes for Grids in Spherical Coordinates, Monthly Weather Review 127, 2204-2210.

[6] Matsumoto, M., Arakawa, T., Kitayama, T., Mori, F., Okuda, H., Furumura, T., Nakajima, K., 2015, Multi-Scale Coupling Simulation of Seismic Waves and Building Vibrations using ppOpen-HPC, Procedia Computer Science 51, 1514-1523, doi: 10.1016/j.procs.2015.05.341

[7] 住元真司, 坂口吉生, 松葉浩也, 中島研吾, 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), スーパーコンピューティングニュース Vol.24 No.2, https://www.cc.u-tokyo.ac.jp/public/VOL24/No2/10_202203Wisteria-1.pdf

[8] h3-Open-BDEC：https://h3-open-bdec.cc.u-tokyo.ac.jp/

[9] 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

[10] 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

[11] 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

[12] 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

[13] Arakawa, T., Inoue, T., Yashiro, H. et al. Coupling library Jcup3: its philosophy and application. Prog Earth Planet Sci 7, 6 (2020). https://doi.org/10.1186/s40645-019-0320-z

[14] 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, https://doi.org/10.1145/3492805.3492809

Wisteria/BDEC-01（Odyssey）における OpenMP によるプログラミング入門（その 2）

中島研吾[(a)]，笠井良浩[(b)]，坂口吉生[(b)]

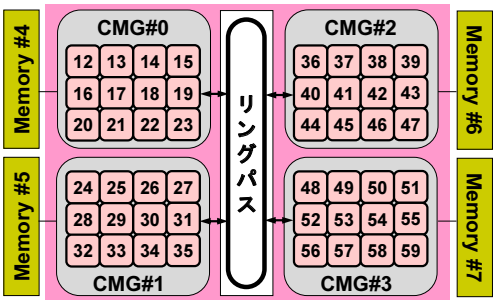
(a)東京大学情報基盤センター，(b)富士通株式会社

本稿では，前号（その 1）に引き続き，Wisteria/BDEC-01（Odyssey）上での最適化について，説明する。プログラム類は Wisteria/BDEC-01 の **`/work/share/ompw/ompw.tar`** から取得できるので，興味ある方は試してみられると良い。

1. Odyssey での実行

（1）A64FX プロセッサ

Odyssey の計算ノードは，A64FX プロセッサであり，図 1 に示すように 48 個のコアから構成されている。12 コアが CMG（Core Memory Group）を構成しており，各 CMG に HBM2 による高速メモリ（8GiB）が搭載されている［1］。いわゆる NUMA アーキテクチャであるため，OpenMP による並列化は各 CMG 単位で実施し，OpenMP/MPI ハイブリッド並列プログラミングでは，ノード当たり 4 つの MPI プロセスを立ち上げる，ことが推奨されているが，本稿では敢えて，1 ノード，48 コアに対して OpenMP 並列化を適用する。図 1 に示すように，メモリ，コアの番号は各 CMG において，CMG#0（メモリ：#4，コア：#12-#23），CMG#1（メモリ：#5，コア：#24-#35），CMG#2（メモリ：#6，コア：#36-#47），CMG#3（メモリ：#7，コア：#48-#59）



プロセッサ名	A64FX
プロセッサ数 (コア数)	1 (48+アシスタントコア2 or 4)
周波数	2.2 GHz
理論演算性能	3.3792 TFLOPS
メモリ容量	32 GiB
メモリ帯域幅	1,024 GB/s
L1 Cache	64 KiB/core (Inst/Data)
L2 Cache	8 MiB/CMG

図 1　A64FX プロセッサの構成と諸元［1］

（2）C コンパイラ

Odyssey では C コンパイラとして「trad モード」,「clang モード」の 2 種類がある（表 1 参照）。デフォルトは trad モードである。使用する機能や性能によって使いわける必要があるが，本稿のケースの場合，問題規模が比較的小さい場合は「clang モード」の方が高速であったため，上記のサンプルプログラムでは基本的に「clang モード」を使用している。後述するように，最適化を施し，問題規模を大きくした場合には，同等，もしくは「trad モード」の方が高速である場合もある。

表 1　C コンパイラ：2 種類のモード

trad モード （-Nnoclang オプション） （デフォルト）	• 「京」および PRIMEHPC FX100 以前のシステム向け富士通コンパイラをベースとする。 • trad モードは，従来の富士通コンパイラとの互換を重視する場合に適している。 • サポートしている仕様は, C89/C99/C11, OpenMP 3.1/OpenMP 4.5（一部） • オプション省略時（デフォルト）は，-Nnoclang オプション適用
clang モード （-Nclang オプション）	• オープンソースソフトウェアである Clang/LLVM コンパイラをベースとする。 • clang モードは，最新言語仕様を使用したプログラムや，オープンソースソフトウェアを翻訳する場合に適している。 • サポートしている仕様は，C89/C99/C11， OpenMP 4.5/OpenMP 5.0（一部)

（3）コンパイル・実行

まず，図 1 の CMG＃0 のみ 12 コアを使ったケースを実行してみよう。詳細は［2］をご覧いただくとして，ここでは簡単に概要について紹介する。

```
>$ cp /work/share/ompw/ompw.tar .
>$ tar xvf ompw.tar
>$ cd ompw            ! (このディレクトリを以下<$O-ompw>と呼ぶ)
>$ module load fj     ! (ログインしたら必ずこれをタイプする)
>$ make -f makec      ! (Fortran であれば make -f makef)
>$ cd run

"INPUT.DAT", "c12.sh", "f12.sh"を修正する

>$ pjsub c12.sh       ! (Fortran であれば psub f12.sh)
```

図 2　ファイルのコピー，コンパイル，実行［2］

C 言語の場合，**make -f makc-org** とすると（2）で述べた「trad」モードになる。コンパイルオプションは，下記のようになっている。:

- **makec：-Kfast,openmp -Nclang -msve-vector-bits=512**
- **makec-org：-Kfast,openmp**

-msve-vector-bits=512 は，SIMD 長を 512 に固定するオプションである。Fortran では SIMD 長は 512 に固定されているが，clang では可変長 SIMD がデフォルトとなっている。SIMD 長を 512 に固定することによって，PCG 法の計算の大部分を占める行列ベクトル積部が高速化されたため，本オプションを採用することとしている。

図 3 は実行制御ファイル**<$O-ompw>/run/INPUT.DAT** の例である。ここではまず，メッシュ数=2,097,152（=128^3）の場合を実施する。

図 4 はジョブスクリプト**<$O-ompw>/run/c12.sh** の例である。Fortran 用の**<$O-ompw>/run/f12.sh** も **solc0** が **solf0** となっている以外は同様である。NUMA アーキテクチャにおいて，できるだけローカルなメモリを使用して効率良く計算を実行するために［3］,

numactl -l を使用する。また使用する CMG を指定するために**-C** でコア番号，**-m** でメモリ番号（図 1）を指定することもできるが，結果的に計算時間への影響はほとんどない［2］。環境変数 **XOS_MMM_L_PAGING_POLICY** については複数 CMG 対して OpenMP を適用する場合には「**demand**」を指定することが推奨されている。詳細は［2］を参照されたい。

```
128 128 128                     NX NY NZ        !X,Y,Z 方向のメッシュ数
1.00e-0  1.00e-00 1.00e-00      DX/DY/DZ        !各メッシュの 3 辺の長さ
1.0e-08                         EPSICCG !収束判定値（10-8 を使用）
```

図 3　メッシュ数=128^3 の場合の実行制御ファイル**<$O-ompw>/run/INPUT.DAT** の例［2］

```
#!/bin/sh
#PJM -N "c12"                           !ジョブ名称（省略可）
#PJM -L rscgrp=debug-o          !実行キュー名（Resource Group）
#PJM -L node=1                          !ノード数（原則＝1）
#PJM --omp thread=12                    !スレッド数（1-48）
#PJM -L elapse=00:15:00                 !実行時間
#PJM -g gxYZ                            !グループ名（財布）
#PJM -j
#PJM -e err                             !エラー出力ファイル
#PJM -o c12.lst                         !標準出力ファイル

module load fj
export OMP_NUM_THREADS=12               !スレッド数（--omp thread=XX と同じ数）
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl -l ./solc0
numactl -C 12-23 -m 4 ./solc0
```

図 4　12 コア・1-CMG を使用する場合のジョブスクリプト**<$O-ompw>/run/c12.sh** の例［2］

また，**solver_PCG.c** のインタフェースが，［4］のコードと比べて若干変更となっている。図 5 が変更を施した冒頭部で，配列 W へのアクセス効率向上のために，配列 W に対して連続領域を確保するように変更した。また係数行列に関連したポインタに restrict 型修飾子を適用することによって，最適化の促進を図っている。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <omp.h>
#include "solver_PCG.h"

extern int
solve_PCG(int N, int *restrict indexLU, int *restrict itemLU,
         double *restrict D, double *restrict B, double *restrict X,
         double *restrict AMAT, double EPS, int *restrict ITR, int *restrict IER)
{
       double VAL, BNRM2, WVAL, SW, RHO, BETA, RHO1, C1, DNRM2, ALPHA, ERR;
       double Stime, Etime;
       int i, j, ic, ip, L, ip1, N3;

 double (*restrict W)[N] = (double (*)[N])malloc(4*sizeof(double[N]));
       if(W == NULL) {
             fprintf(stderr, "Error: %s¥n", strerror(errno));
       return -1;
       }
```

図 5　**solver_PCG.c** の変更部分，係数行列に関連したポインタに restrict 型修飾子の適用と，配列 W に対して連続領域の確保

（4）実行例

表 2 は，図 3 に示す NX=NY=NZ=128 の場合にコア数（スレッド数）を 1 から 12 まで変化させた場合の PCG 法ソルバーの計算時間，並列化効率の値を示す。計算を 5 回実行して，最速時間を採用している（以下同様に測定している）。コア数（スレッド数）を増加させると，計算粒度（Granularity）の低下，メモリの実行性能が飽和するため，効率は低下するが，12 コア使用時に 75%程度の並列化効率が達成されている。

表 2　PCG 法ソルバーの計算時間（NX=NY=NZ=128）（1～12 コア）（Fortran）

Thread #	sec	Speed-up	Parallel Efficiency (%)
1	50.27	1.00	100.00
2	25.24	1.99	99.60
4	12.98	3.87	96.86
6	9.24	5.44	90.73
8	7.27	6.92	86.50
12	5.27	9.54	79.49

表 3　PCG 法ソルバーの計算時間（NX=NY=NZ=128）（12~48 コア）（Fortran）

Thread #	sec	Speed-up	Parallel Efficiency (%)
12	5.27	12.00	100.00
24	2.78	22.72	94.68
36	1.95	32.49	90.24
48	1.60	39.54	82.38

表 3 は CMG の数を増やして，最大 48 コアまで使用した場合である。12 コア（1-CMG）の場合の計算性能を 12.0 としてある。48 コア使用時の並列化効率は，12 コアの場合を基準とすると 80%程度となっている。スレッド数を変化させるためには，図 4 のジョブスクリプトの中の「**`#PJM --omp thread=XY`**」の「**`XY`**」の値を変化させれば良いが，**`<$O-ompw>/run/`**には，各スレッド数に対して，**`cXY.sh, fXY.sh`**（`XY=01,02,04,06,08,12,24,36,48`）が用意されている。

2. 複数 CMG での性能向上：First Touch Data Placement

表 3 に示すように，48 コア，すなわち 4-CMG を使用した場合の並列化効率は 12 コア，1-CMG の場合を基準とすると 80%程度である。これは決して悪い数字とは言えないが，更なる高速化を試みよう。

NUMA アーキテクチャでは，プログラムにおいて変数や配列を宣言した時点では，物理的メモリ上に記憶領域は確保されず，ある変数を最初にアクセスしたコア（の属する CMG）のローカルメモリ上に，その変数の記憶領域（ページ）が確保される［3］。

これを **First Touch Data Placement（First Touch）**［3］と呼び，配列の初期化手順により得られる性能が大幅に変化する場合があるため，注意が必要である。例えばある配列を初期化する場合，特に指定しなければ 0 番の CMG で初期化が行われるため，記憶領域は 0 番 CMG のローカルメモリ上に確保される。したがって，他の CMG でこの配列のデータをアクセスする場合には，必ず 0 番 CMG のメモリにアクセスする必要があるため，高い性能を得ることは困難である。

配列の初期化を，実際の計算の手順にしたがって OpenMP を使って並列に実施すれば，実際に

計算を担当する CMG のメモリにその配列の担当部分の記憶領域が確保され，より効率的に計算を実施することができる。1-CMG しか使用しない場合はこのような配慮は不要であり，例えば OpenMP/MPI ハイブリッドで 1-CMG 当りに 1 つの MPI プロセスを使用する場合も同様である。

1.で使用したプログラム類は「**<$O-ompw>/src-c0**（実行形式は **solc0**）」，「**<$O-ompw>/src-f0**（同 **solf0**）」であるが，ここで述べた「First Touch」を適用したプログラム類は，「**<$O-ompw>/src-c1**（同 **solc1**）」，「**<$O-ompw>/src-f1**（同 **solf1**）」にある。プログラムの変更箇所は，係数行列を生成する **poi_gen.c, poi_gen.f** である。変更は配列の初期化の部分のみである。

図 6，図 7 に **src-c0, src-f0** との相違点を示す。**src-c1, src-f1** では配列初期化部分が計算実行時と同様に OpenMP によって並列化されている部分のみが異なっている。表 4 に First Touch 適用の効果を示す。48 コアで約 10%の性能改善が得られている。実行にあたっては，**<$O-ompw>/run/**に，**c1_XY.sh, f1_XY.sh**（XY=12,24,36,48）が用意されているので，それらを適宜編集して使用すると良い。

表 4　PCG 法ソルバー計算時間（NX=NY=NZ=128）（12~48 コア）（Fortran），First Touch の効果

	Thread #	sec	Speed-up	Parallel Efficiency (%)
src-f0	12	5.27	12.00	100.00
	24	2.78	22.72	94.68
	36	1.95	32.49	90.24
	48	1.60	39.54	82.38
src-f1	12	5.26	12.00	100.00
	24	2.70	23.39	97.45
	36	1.86	33.83	93.96
	48	1.44	43.77	91.18

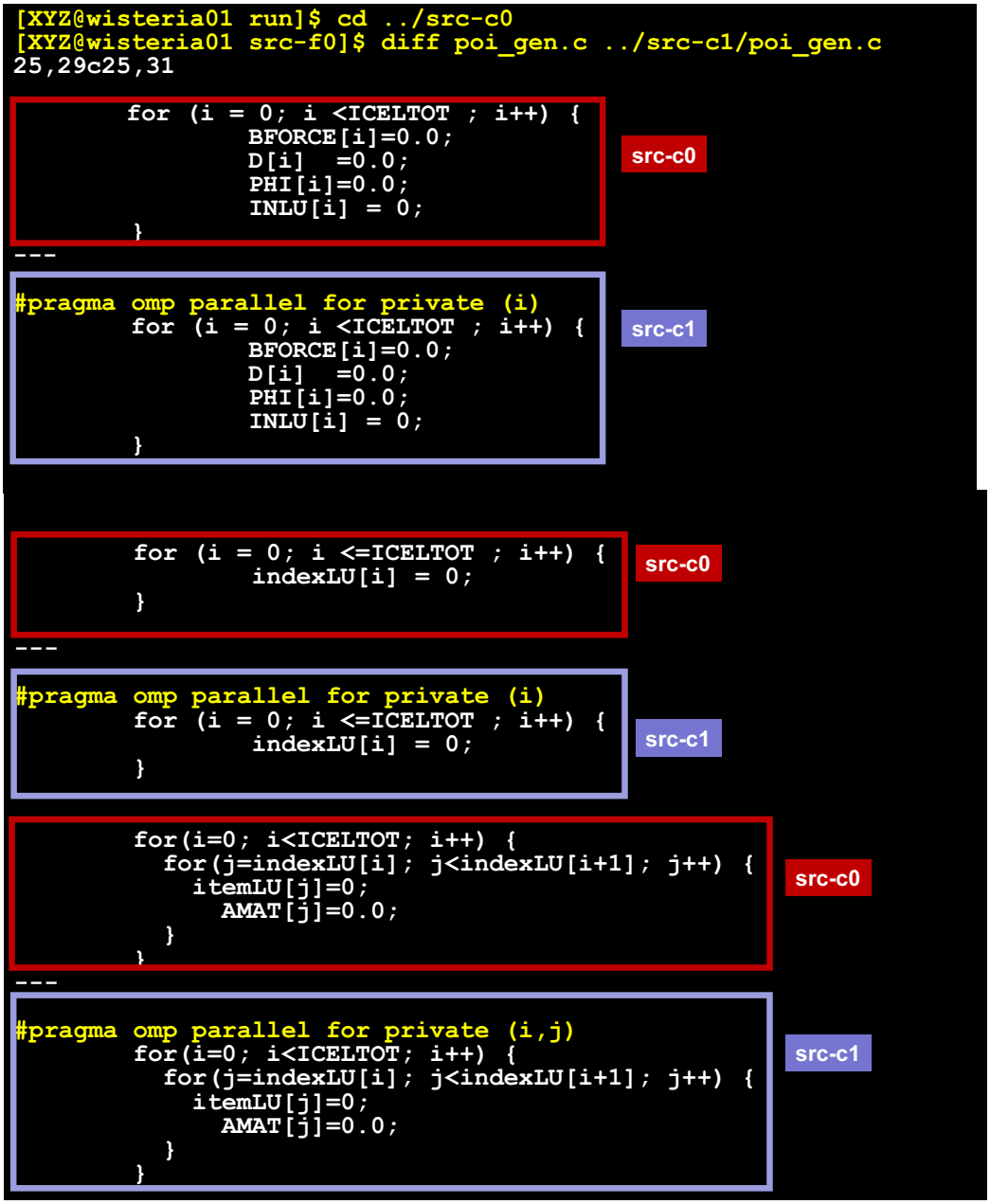


図 6　First Touch を適用するためのプログラム変更（**<$O-ompw>/src-c0/poi_gen.c, <$O-ompw>/src-c1/poi_gen.c**）

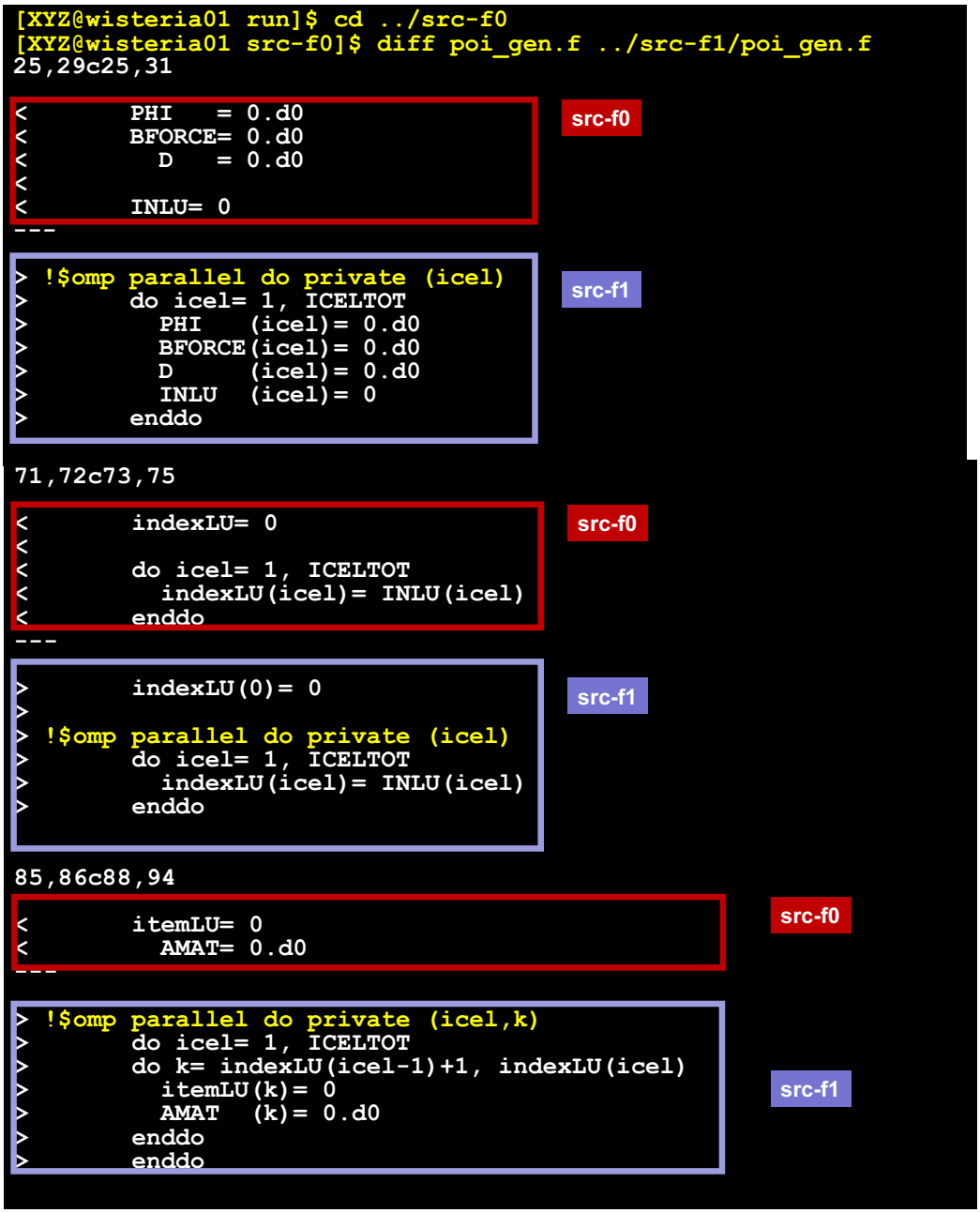


図 7　First Touch を適用するためのプログラム変更（**<$O-ompw>/src-f0/poi_gen.f, <$O-ompw>/src-f1/poi_gen.f**）

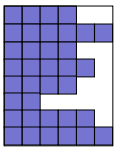
3. 疎行列格納形式の効果

図 8 は，CG 法で疎行列ベクトル積 $A\,p = q$ の計算を実行している部分で，**<$O-ompw>/src-c0,src-c1** 及び**<$O-ompw>/src-f0,src-f1** では，前号記事でも述べたようにこのように CRS（Compressed Row Storage）形式［5］と呼ばれる疎行列格納形式を採用している：

(a)

CRS（Compressed Row Storage）

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



(b)

CRS（Compressed Row Storage）

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

図 8　CRS 形式（Compressed Row Storage）［5］による疎行列格納方法，疎行列ベクトル積実装，（a）C 言語（**<$O-ompw>/src-c0,src-c1**）（b）Fortran（**<$O-ompw>/src-f0,src-f1**）

疎行列ベクトル積の特徴は，右辺に現れる **W[P][itemLU[j]], W(itemLU(k),P)** が間接参照を含むため，memory-bound なプロセスとなっていることである。CRS 形式はメモリ，計算量の削減には効果的であるが，メモリアクセス効率は必ずしも良くない。Ellpack-Itpack（ELL）形式［5］は各行における非零非対角成分数は最大非零非対角成分数に固定する方法で（図 9），実際に非零非対角成分が存在しない部分は係数=0 として計算する。計算量，必要記憶容量ともに CRS 形式と比較してやや増加するが，高いメモリアクセス効率が得られるため，計算時間としては大幅に短縮できる場合がある［6］。非ゼロ非対角成分が存在しない列に対しては，ダミーの列番号を参照し，係数行列は 0 として扱う（図 9）。

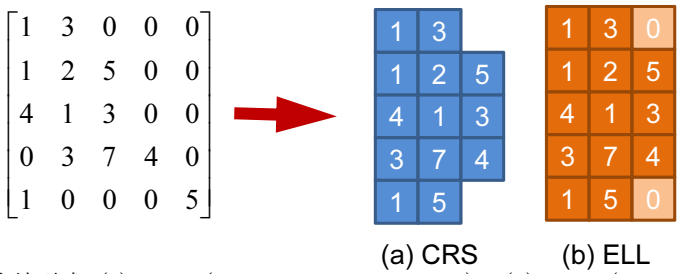


図 9　疎行列の格納形式（a）CRS（Compressed Row Storage），（b）ELL（Ellpack-Itpack）［5］

本稿で扱うアプリケーションは，疎行列の非ゼロ非対角成分の数が最大 6 であるため，ELL 形式を容易に適用することができる。非ゼロ非対角成分数が行によって大幅に変動する場合は，よりフレキシブルな Sliced-ELL 形式［7］が使用される場合もある［6］。

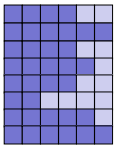
図 10 は，図 8 に示す疎行列ベクトル積を ELL 形式で記述したものである。C 言語の場合は，図 11 に示すような実装が考えられるが，A64FX では非常に計算速度が遅いため，図 10（a）の

ような実装を採用している。Intel Xeon プロセッサでは，図 10（a）と図 11 の実装は同じ効率が得られる。ダミー列番号の設定方法等の実装の詳細については，プログラム本体（**<$O-ompw>/src-c2**（実行形式は **solc2**），**<$O-ompw>/src-c2**（同 **solf2**））及び講習会資料［2］を参照されたい。

(a)

ELL (ELLPACK/ITPACK)

```
for(i=0; i<N; i++) {
      VAL = D[i] * W[P][i];
      for(j=0; j<6; j++) {
            VAL += AMAT[6*i+j] * W[P][itemLU[6*i+j]];
      }
      W[Q][i] = VAL;}
```



(b)

ELL (ELLPACK/ITPACK)

```
do i= 1, N
  W(i,Q)= D(i)*W(i,P)
  do j= 1, 6
    W(i,Q)= W(i,Q) + AMAT(j,i)*W(itemLU(j,i),P)
  enddo
enddo
```

図 10　ELL 形式（Ellpack-Itpack）［5］による疎行列格納方法，疎行列ベクトル積実装，（a）C 言語（**<$O-ompw>/src-c2**）（b）Fortran（**<$O-ompw>/src-f2**）

ELL (ELLPACK/ITPACK)

```
for(i=0; i<N; i++) {
      VAL = D[i] * W[P][i];
      for(j=0; j<6; j++) {
            VAL += AMAT[i][j] * W[P][itemLU[i][j]];
      }
      W[Q][i] = VAL;}
```

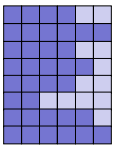


図 11　ELL 形式（Ellpack-Itpack）［5］による疎行列格納方法，疎行列ベクトル積実装，図 8（a）の拡張としては自然な実装であるが，本稿ではこの方法は採用していない

4. ループオーバーヘッド削減の効果

OpenMP の実行モデルは，fork-join モデル［8］と呼ばれるもので，通常はマスタースレッドによるシリアル実行であるが，OpenMP 指示文（directive）によりマルチスレッドが生成し，並列実行を実施するものである。

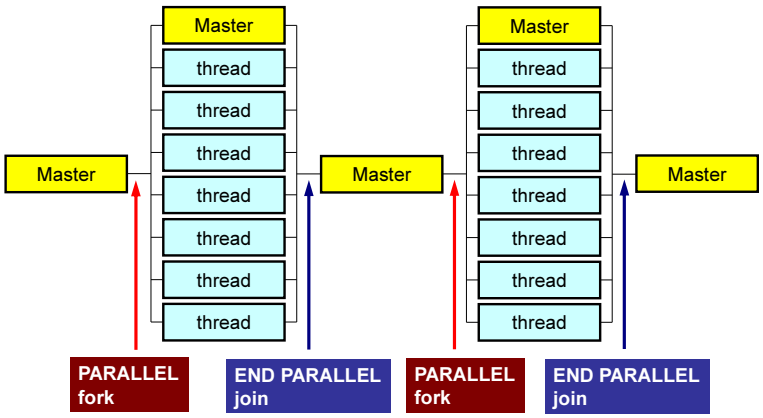


図 12　OpenMP の実行モデル：fork-join モデル［8］

共役勾配法の各ステートメントのほとんどは各ベクトルに対するループであり，これまで紹介したプログラムでは，各ループに OpenMP 指示文を挿入している。図 13 は CG 法の後半部分の実装例（CRS 形式）であり，各ループに「**#pragma omp parallel for**」，「**!$omp parallel do**」が挿入されている。「**#pragma omp parallel**」，「**!$omp parallel**」により，図 12 に示すような fork-join が各ステートメントごとに生じるため，これがオーバーヘッドとなる可能性がある。本章で紹介する新たな実装（**<$O-ompw>/src-c3**（実行形式は **solc3**），**<$O-ompw>/src-c3**（同 **solc3**））は ELL 法による実装（**<$O-ompw>/src-c2**，**<$O-ompw>/src-c2**）に対して，「**#pragma omp parallel**」，「**!$omp parallel**」の呼び出しを各反復で一回とし，各ループに対しては「**#pragma omp for**」，「**!$omp do**」を適用するものである（図 14）。

（a）

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

C1 = 0.0;
#pragma omp parallel for private (i) reduction(+:C1)
  for(i=0; i<N; i++) {
      C1 += W[P][i] * W[Q][i];
  }
  ALPHA = RHO / C1;

#pragma omp parallel for private (i)
  for(i=0; i<N; i++) {
      X[i]   += ALPHA * W[P][i];
      W[R][i] -= ALPHA * W[Q][i];
  }


DNRM2 = 0.0;
#pragma omp parallel for private (i) reduction(+:DNRM2)
  for(i=0; i<N; i++) {
      DNRM2 += W[R][i]*W[R][i];
  }

  ERR = sqrt(DNRM2/BNRM2);
...
```

（b）

```
!$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(itemLU(k),P)
        enddo
        W(i,Q)= VAL
      enddo

      C1= 0.d0
!$omp parallel do private(i) reduction(+:C1)
      do i= 1, N
        C1= C1 + W(i,P)*W(i,Q)
      enddo

      ALPHA= RHO / C1

!$omp parallel do private(i)
      do i= 1, N
        X(i)  = X(i)   + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo

      DNRM2= 0.d0
!$omp parallel do private(i) reduction(+:DNRM2)
      do i= 1, N
        DNRM2= DNRM2 + W(i,R)**2
      enddo

      ERR = dsqrt(DNRM2/BNRM2)...
```

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(i)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

図 13　CG 法の後半部分の実装（CRS 形式）（a）C 言語（**<$O-ompw>/src-c0,src-c1**），（b）Fortran（**<$O-ompw>/src-f0,src-f1**）

5. 計算結果（First-Touch，行列格納形式，ループオーバーヘッド削減の効果）

3.，4.に示した各実装例については，**c2_48.sh**，**c3_48.sh**，**f2_48.sh**，**f3_48.sh** を使用して実行することができる。これらは，全て 48 コアを使用した場合であるが，ジョブスクリプト内でコア数を変更して実行することも可能である。表 5 に 48 コアを使用した場合の PCG 法の計算時間を示す。First-Touch の効果は C 言語でより大きい。ループオーバーヘッド削減の効果は Frotran では 5%程度，C 言語（clang）ではむしろ遅くなっているが，C 言語（trad）では大きな効果が認められ，sol-c3 のレベルでは clang と trad の差はほとんど無くなっている。

表 5　PCG 法ソルバーの計算時間（sec.）（NX=NY=NZ=128）（48 コア）

	Fortran	C（clang）	C（trad）
初期設定（sol-c0, sol-f0）	1.671	1.564	2.354
＋First-Touch（sol-c1, sol-f1）	1.480	1.122	1.720
＋ELL（sol-c2, sol-f2）	0.747	0.809	1.127
＋omp-parallel 削減（sol-c3, sol-f3）	0.707	0.834	0.854

(a)

```
      ITR= N
      Stime= omp_get_wtime()

      do L= 1, ITR
!$omp parallel private(i,k,VAL)

!$omp do
      do i= 1, N
        W(i,Z)= W(i,R)*W(i,DD)
      enddo

      RHO= 0.d0
!$omp do reduction(+:RHO)
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      if ( L.eq.1 ) then
!$omp do
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
       else
        BETA= RHO / RHO1
!$omp do
         do i= 1, N
           W(i,P)= W(i,Z) + BETA*W(i,P)
         enddo
      endif

!$omp do
      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

      C1= 0.d0
!$omp do reduction(+:C1)
      do i= 1, N
        C1= C1 + W(i,P)*W(i,Q)
      enddo

      ALPHA= RHO / C1

!$omp do
      do i= 1, N
        X(i)  = X(i)   + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo

      DNRM2= 0.d0
!$omp do reduction(+:DNRM2)
      do i= 1, N
        DNRM2= DNRM2 + W(i,R)**2
      enddo

!$omp end parallel
      ERR = dsqrt(DNRM2/BNRM2)...
```

(b)

```
  *ITR = N;
  Stime = omp_get_wtime();

  for(L=0; L<(*ITR); L++) {
#pragma omp parallel private (i,j,VAL){

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

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

if(L == 0) {
#pragma omp for
    for(i=0; i<N; i++) {
      W[P][i] = W[Z][i];
    }
    } else {
     BETA = RHO / RHO1;
#pragma omp for
    for(i=0; i<N; i++) {
      W[P][i] = W[Z][i] + BETA * W[P][i];
    }
}

#pragma omp for
  for(i=0; i<N; i++) {
      VAL = D[i] * W[P][i];
      for(j=0; j<6; j++) {
          VAL += AMAT[6*i+j] *
W[P][itemLU[6*i+j]];
      }
      W[Q][i] = VAL;
  }

  C1 = 0.0;
#pragma omp for reduction(+:C1)
  for(i=0; i<N; i++) {
      C1 += W[P][i] * W[Q][i];
  }
  ALPHA = RHO / C1;

#pragma omp for
  for(i=0; i<N; i++) {
      X[i]    += ALPHA * W[P][i];
      W[R][i] -= ALPHA * W[Q][i];
  }

  DNRM2 = 0.0;
#pragma omp for reduction(+:DNRM2)
  for(i=0; i<N; i++) {
      DNRM2 += W[R][i]*W[R][i];
  }

}

ERR = sqrt(DNRM2/BNRM2);
```

図 14　ループオーバーヘッドを削減した実装例（a）C 言語（**<$O-ompw>/src-c3**），（b）Fortran（**<$O-ompw>/src-f3**）

今回は，Odyssey 上での実行方法，First Touch Data Placement（First Touch）・疎行列格納形式・ループオーバーヘッド削減の効果について述べた。次回（恐らく最終回）は更なる最適化，詳細プロフィラによる性能測定法について紹介する。

（第 3 回（7 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] OpenMP によるマルチコア・メニィ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/

[3] Mattson, T.G., Sanders, B.A., Massingill, B.L., Patterns for Parallel Programming, Software Patterns Series (SPS), Addison-Wesley, 2005

[4] 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

[5] Saad, Y.: Iterative Methods for Sparse Linear Systems Second Edition, SIAM, 2003

[6] Nakajima, K., Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method, Proceedings of the 20th IEEE International Conference for Parallel and Distributed Systems (ICPADS 2014) 25-32, Hsin-Chu, Taiwan, 2014

[7] Monakov, A., Lokhmotov, A., Avetisyan, A., Automatically tuning sparse matrix-vector multiplication for GPU architectures, Lecture Notes in Computer Science 5952, 112-125, 2010

[8] OpenMP ARB（Architecture Review Board）：https://www.openmp.org/

分子動力学計算によるアミロイド凝集様態の理論的解析

大 滝 大 樹

長崎大学　生命医科学域（医学系）　分子標的医学研究センター

１．はじめに

アルツハイマー病・パーキンソン病（PD）・プリオン病など，神経変性疾患の患者の神経細胞にはアミロイドの異常蓄積が見られる。アミロイドがβシート構造を多く有することは知られているが，その構造不均一性などから通常のタンパク質に用いられる構造決定手法の多くが適用できず，詳細な立体構造の決定は極めて難しい。そのためアミロイドの詳細な分子構造情報は少なく，変異により凝集性が大きく変化する原因など，アミロイドの分子構造と凝集性との関係は不明な点が多い。

このような中，近年の実験技術の進歩によりアミロイドの三次構造が報告されるようになった。最近ではPDと関わりがあるとされるα-シヌクレイン（αSyn）のアミロイドの構造が次々に報告されている。αSynはパーキンソン病，レビー小体型認知症，多系統萎縮症の特徴となるレビー小体の主要な構成タンパク質である。αSynは正常型では細胞質内において特定の構造を持たない状態とαヘリックスが豊富な構造とで平衡状態で存在していると考えられている[1]。また，*in vitro*ではαヘリックス構造が豊富な四量体で存在するという報告もある[2]。一方，レビー小体中に異常蓄積しているαSynはアミロイド構造をとり，株多様性などプリオン様の性質を示すことが知られている。プリオン病の原因となるプリオンタンパク質の構造についてはクライオ電子顕微鏡で2021年に初めてアミロイドの構造が報告された[3]ものの，未だに分かっていない部分が多い[1]。

アミロイドは一般的に疎水性相互作用を豊富に持つことが知られている。そこで本研究ではアミロイドの疎水性領域に着目し，①固体NMRにより提案されたαSynアミロイド構造を用いた分子動力学（MD）シミュレーションを行い，②これまでの研究および①で得られた知見に基づきプリオンタンパク質の部分アミロイドのモデルを作成しMDシミュレーションを行った。疎水性残基への変異を導入した変異型についても計算を行い，疎水性相互作用がアミロイドの構造安定性に及ぼす影響を調べた。

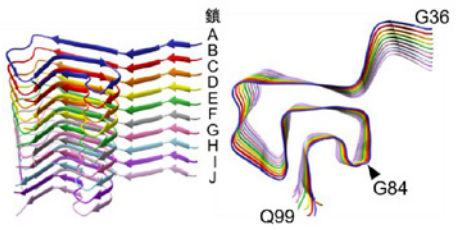


図 1：α-Synアミロイド

PDB：2N0Aの第36-99残基部分の構造。A-Jの10本の鎖が積層している。

２．計算方法

２．１．αSynのMDシミュレーション

αSynアミロイドの計算の初期構造には固体NMRにより得られた構造（PDB: 2N0A）[4]を用いた。構造が不安定な1-35，100-140残基を取り除

[1] 本研究を開始したのはクライオ電子顕微鏡による構造が報告されるより前である。報告された構造と本研究で作成したモデルアミロイドの間には細かい部分で違いは見られるが，モデル作成時に仮定したparallel-in-registerβシート構造が実験でも観測されるなど，共通点も見られる。

き，N 末端と C 末端をそれぞれアセチル化，N-メチルアミド化することでキャップした。図 1 に計算に用いたαSyn アミロイドの構造を示す。

変異型のモデルは SCWRL4[5]により作成した。周囲に水分子，Na イオン，Cl イオンを配置し，150 mM の NaCl 水溶液を模した系を作成した。

MD 計算は GROMACS[6]を用いて Reedbush-U/H 上で行った。力場は，タンパク質には AMBER ff99SB-ILDN[7]，水分子には TIP3P[8]を用いた。温度は Velocity-rescaling 法[9]により 310 K に制御し，圧力は Berendsen 法[10]により 1 bar に制御した。MD 計算の時間刻みは 2 fs とした。最小化計算および平衡化計算ののち，400 ns のプロダクション計算を行った。各変異型について独立した MD シミュレーションを 3 回（野生型は 5 回）ずつ行った。

2．2．プリオンのモデルアミロイドの MD シミュレーション

プリオンのアミノ酸配列から疎水性残基が豊富な部分配列（第 107-143 残基）を取り出し，疎水性相互作用を豊富に持つ（疎水性コア）モデルアミロイドを作成した。まず，αSyn の結果から得られた知見などに基づき Swiss PDB Viewer[11]を用いて一層分のコンフォメーションを作成した。これを UCSF Chimera[12]を用いて 10 層分積層した後，Modeller[13]を用いてアミロイド構造の refinement を行った。変異型モデルは SCWRL4[5]により作成した。得られたモデルアミロイドのタンパク質の N 末端と C 末端をそれぞれアセチル化，N-メチルアミド化することでキャップした。

MD 計算は，最小化計算および平衡化計算の後 50 ns のプロダクション計算を行った。各変異型について独立した MD シミュレーションを 10 回ずつ行った。その他の条件はαSyn と同様である。

2．3．解析

それぞれの MD 計算で 10 ps ごとにスナップショットを取得し，αSyn は 100-400 ns，プリオンのモデルアミロイドは 40-50 ns のトラジェクトリを用いて解析を行った。二次構造解析には DSSP[14,15]を用いて，βシート化傾向を評価した。疎水性相互作用解析には PyInteraph[16]を用いた。疎水性アミノ酸残基の側鎖の重心間距離が 5 Å 以内だった場合に疎水性相互作用をカウントし[17]，MD 計算のトラジェクトリ中に占める割合をスコアとして比較した。分子の図は UCSF

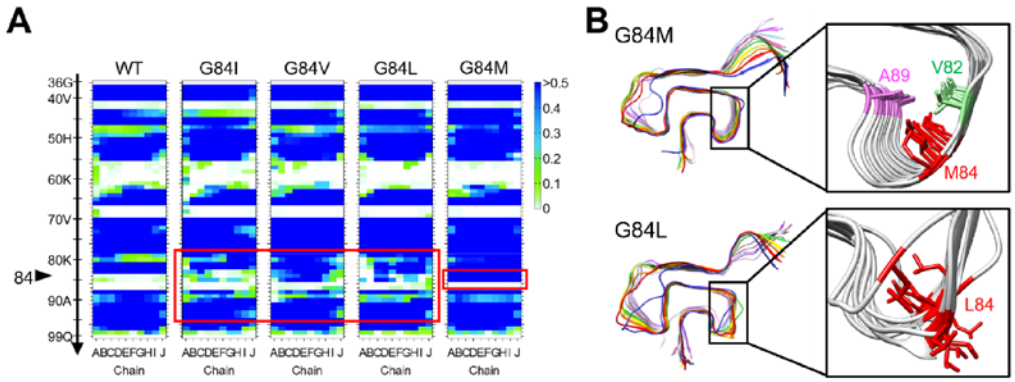


図 2：αSyn における MD 計算の結果

(A)βシート化傾向。(B)G84M および G84L 変異型の最後のスナップショットと第 84 残基近傍の U 字ループ構造における疎水性残基の側鎖の配向。

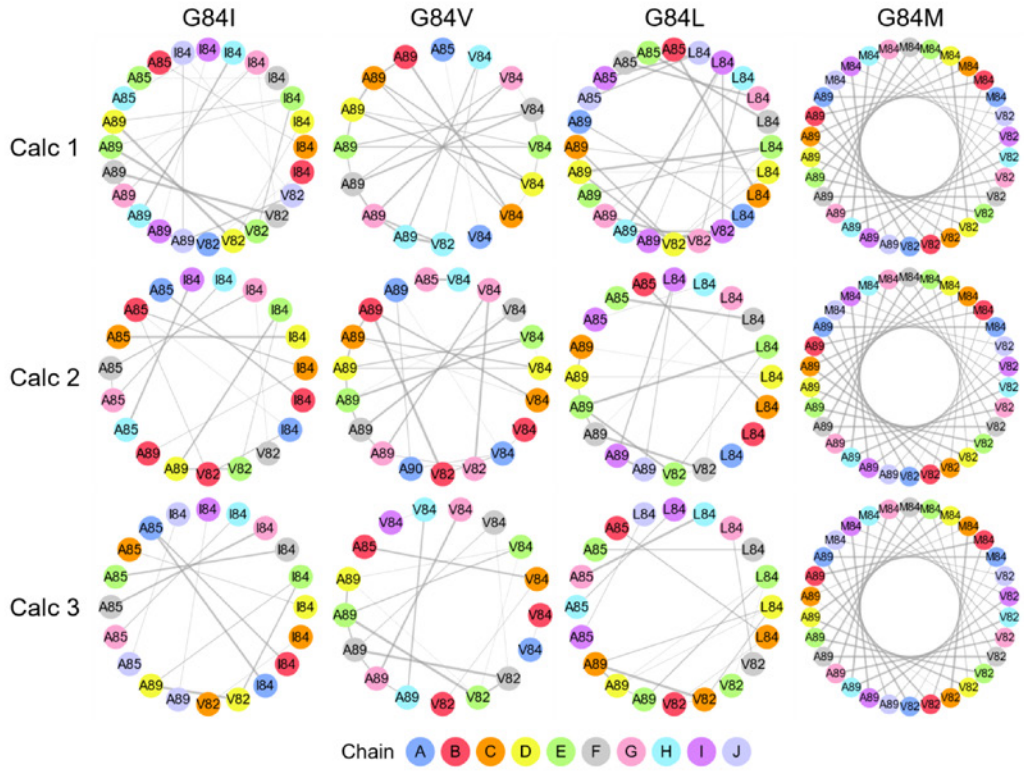


図 3：αSyn における疎水性相互作用

各点（ノード）がアミノ酸残基を表し，疎水性相互作用がある残基間を線で結んでいる。線が太いほど疎水性相互作用が強い。

Chimera[12]，疎水性相互作用のダイアグラムは Cytoscape[18]を用いて作成した。

3．結果

3．1．αSyn

本研究ではαSyn アミロイドの U 字ループ構造（第 84-87 残基）に注目した。このような U 字型の構造はαSyn 以外のアミロイドにも多く見られるため，本研究により得られた知見が他のアミロイドにも適用できる可能性がある。そこで，第 84 残基の Gly を疎水性残基（Ile，Met，Leu など）に変更したモデルを作成し MD 計算を行った。

図 2A に野生型（WT），G84I，G84V，G84L，G84M 変異型の計算から得られたβシート化傾向を示す。第 84 残基近辺を見ると G84M 変異体のβシート化傾向が特に大きく，構造が安定化していることが分かる。一方で G84I はβシート化傾向が小さく U 字ループ構造が不安定化していることが分かる。MD 計算終了時における G84M，G84L 変異型の U 字ループ部分のスナップショットを図 2B に示す。これを見ても，G84M では M84 が A89 や V82 と相互作用して U 字ループ構造が安定化していることが分かる。

図 3 には G84I，G84V，G84L，G84M 変異型の疎水性相互作用ダイアグラムを示す。どの変異型も第 84 残基と V82・A89 との疎水性相互作用が見られるが，G84M 変異型が一番強く，かつ再現性良く疎水性相互作用を形成していることが分かる。この理由は Met の側鎖が他の疎水性残基に比べて長く，周辺の疎水性残基と相互作用しやすいためと考えられる。僅かな長さの差が部分構造の安定性に大きく影響することが示唆される。

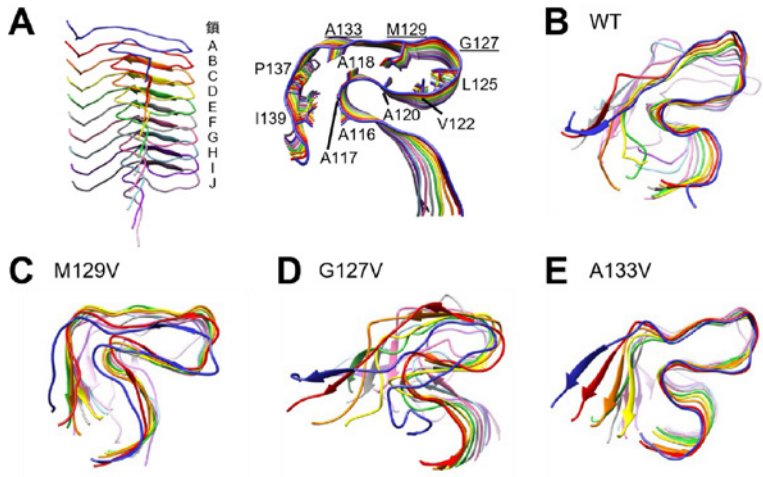


図 4：プリオンのモデルアミロイド

(A)作成したモデルアミロイドの構造。変異を導入したアミノ酸残基には下線を付している。(B)-(E)は MD シミュレーション終了時のスナップショット。(B)WT，(C)M129V，(D)G127V，(E)A133V。

3．2．プリオンのモデルアミロイド

本研究で作成したモデルアミロイドの構造を図 4A に示す。変異はこれまでの実験研究などの報告に基づき，G127V，M129V，I138M，A133V を考えた。

図 4B-E に 50 ns の MD 計算における最後のスナップショットを示す。野生型（WT）に比べて，M129V 変異型は全体的にアミロイド構造を保っているが（図 4C），G127V 変異型ではアミロイド構造が大きく崩れている（図 4D)。M129V 変異は疎水性コアを大きく安定化することが分かっているが，本モデルの計算結果でも安定化されており，実験事実と矛盾しない。また，ヒトのプリオンタンパクのアミロイドはクロイツフェルト・ヤコブ病（CJD）と関係しているが，G127V 変異は CJD に感染し難いことが分かっている。本計算結果より，G127V 変異型はアミロイドの構造の保持が難しく，蓄積が困難になることで CJD に感染し難くなる可能性が示唆される。

図 5 はアミノ酸残基間の疎水性相互作用のスコアを比較したものである。WT と M129V 変異型で U 字ループ部分（第 120-130 残基，図 4A 参照）の相互作用を比較すると，WT では V122-M129 のスコアが A120-M129 や V122-L125 に比べて大きいのに対し，M129V 変異型では 3 つのスコアが同等（〜400）になっており，変異により相互作用のバランスが変化していることが分かる。さらに，M129V 変異型では変異箇所のスタック方向の相互作用（図 5，M/V129-M/V129）のスコアが高くなっていることからアミロイドが安定化されたことが見て取れる。また，A115-V121 および A118-P137 では M129V 変異によりスコアが減少しているのに対し，A113-V121，A117-A133，A117-I139，I139-I139 では（有意性はないが）スコアが増加している。このように，変異の影響は変異箇所近辺だけでなく，より広範な領域に影響を与えながら疎水性相互作用のバランスを変化させていることが分かる。また，G127V 変異における構造の不安定性は V122-V122，V122-M129，L125-L125，L130-L130 のスコアの大幅な減少からも確認できる。

A133V 変異型の MD 計算における最後のスナップショット（図 4E）は WT に比べて構造が安定化しているように見える。これは Ala から Val へ変わったことにより A118-V133 の疎水性相互作用が大きくなり U 字ループの「閉じ口」の部分（図 5 右下図を参照）が安定化したこと，また，

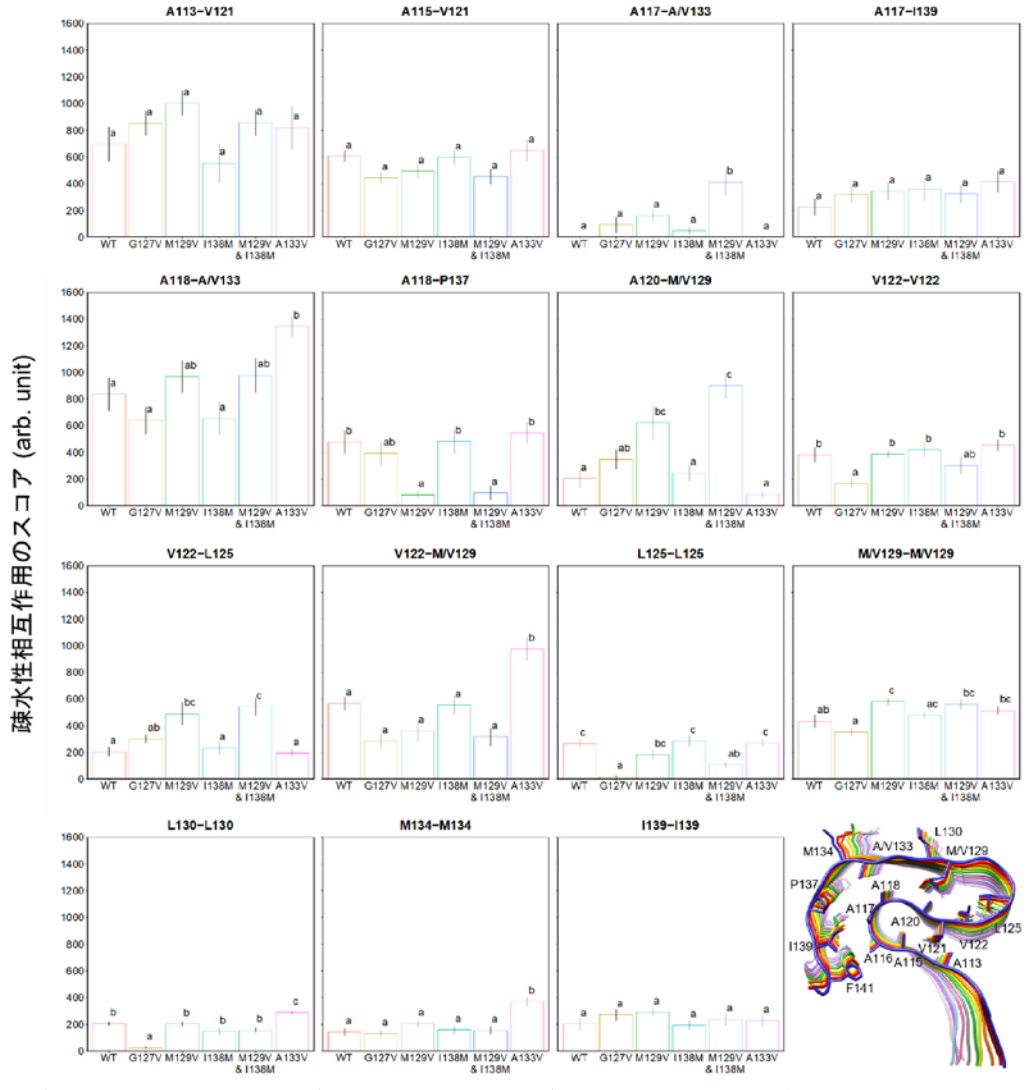


図 5：疎水性相互作用のスコア

エラーバーは標準誤差を示す。変異型間のスコアの差の有意性については Tukey 法（$\alpha = 0.05$）で比較した。同じアルファベットが付記されている棒グラフ間には有意差がない。

それに伴い M134-M134，L130-L130 のスタック方向の相互作用や V122-M129 の相互作用が増加したことが原因と考えられる（図 5）。図 6 にシミュレーション中における A118-A/V133 の α 炭素（C_α）間の距離の変化を示す。これを見ても，A133V 変異型では距離が 7 Å 程度に保たれており，WT に比べ安定化していることが分かる。

I138M 変異型および M129V&I138M 変異型についても，疎水性相互作用の変化やそれに伴う構造変化について知見を得ることができた。特徴的なのは，A120-M/V129 や A117-A133 の疎水性相互作用に見られるように，M129V 変異により強くなった相互作用が I138M 変異を導入することでさらに強くなる点である（図 5）。これは第 128-133 残基の β シートがシフト（図 4A では左方向）することで U 字ループ部分の相互作用が入れ替わることに起因しており，M129V と I138M が相性の良い変異であることを示唆している。詳細な議論は文献[19]を参照されたい。

4．まとめと今後の展望

本研究では αSyn アミロイドに変異を導入したモデルを幾つか作成し，MD 計算を行った。U 字型のループ構造に着目し，疎水性コアの疎水性相互作用が αSyn アミロイドの構造安定性に大きく影響することを示した。また，これらの知見に基づき，プリオンタンパク質の一部の配列を用いたモデルアミロイドを作成して MD 計算を行った。実験などで報告されている特徴的な変異型についても計算を行い，我々のモデルが実験で得られた結果と大きく矛盾しないことを確認した。さらに，疎水性相互作用に注目し，疎水性残基の変化が疎水性コア（ひいてはアミロイドの構造）の安定性に大きく影響することが分かった。今後は，近年，数多く報告されているアミロイドの構造（プリオンのアミロイドを含む）などと比較することで，疎水性相互作用とアミロイドの構造安定性について議論を深めていきたいと考えている。

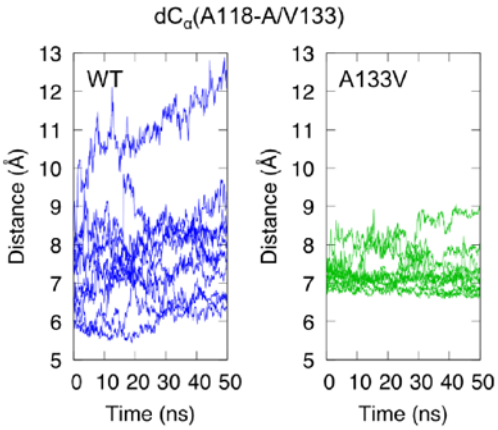


図 6：A118-A/V133 の C_α 間の距離

謝　辞

本研究は『2019 年度（前期・後期）東京大学情報基盤センター「若手・女性利用者推薦」』および『学際大規模共同利用・共同研究拠点（JHPCN）萌芽型共同研究課題』における採択課題「分子動力学計算によるアミロイド凝集様態の理論的解析」（拠点課題 ID：EX19301）によって行われました。本研究における数値計算は東京大学情報基盤センターの Reedbush-U/H 上で行われました。また，本研究は科学研究費助成事業（若手研究：19K16058）の支援を受けています。この場を借りて感謝申し上げます。

参　考　文　献

[1] A. Pineda, J. Burré, “Modulating membrane binding of α-synuclein as a therapeutic strategy” Proc. Natl. Acad. Sci. USA 114, 1223-1225 (2017)

[2] T. Bartels, J. G. Choi, D. J. Selkoe, “α-synuclein occurs physiologically as a helically folded tetramer that resists aggregation” Nature 477, 107-110 (2011)

[3] A. Kraus, F. Hoyt, C. L. Schwartz, B. Hansen, E. Artikis, A. G. Hughson, G. J. Raymond, B. Race, G. S. Baron, B. Caughey, “High-resolution structure and strain comparison of infectious mammalian prions” Mol. Cell 81, 1-12 (2021).

[4] M. D. Tuttle, G. Comellas, A. J. Nieuwkoop, D. J. Covell, D. A. Berthold, K. D. Kloepper, J. M. Courtney, J. K. Kim, A. M. Barclay, A. Kendall, W. Wan, G. Stubbs, C. D. Schwieters, V. M. Y. Lee, J. M. George, C. M. Rienstra, “Solid-state NMR structure of a pathogenic fibril of full-length human α-synuclein” Nat. Struct Mol. Biol. 23, 409-415 (2016)

[5] G. G. Krivov, M. V. Shapovalov, R. L. Dunbrack, “Improved prediction of protein side-chain conformations with SCWRL4” Proteins: Structure, Function, and

Bioinformatics 77, 778-795 (2009)
[6] 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, E. Lindahl, “GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit” Bioinformatics 29, 845-854 (2013)
[7] K. Lindorff-Larsen, S. Piana, K. Palmo, P. Maragakis, J. L. Klepeis, R. O. Dorr, D. E. Shaw, “Improved side-chain torsion potentials for the AMBER ff99SB protein force field” PROTEINS: Struct. Funct. Gen. 78, 1950-1958 (2010)
[8] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, M. L. Klein, “Comparison of simple potential functions for simulating liquid water” J. Chem. Phys. 79, 926-935 (1983)
[9] G. Bussi, D. Donadio, M. Parrinello, “Canonical sampling through velocity rescaling” J. Chem. Phys. 126, 014101 (2007)
[10] H. J. C. Berendsen, J. P. M. Postma, A. DiNola, J. R. Haak, “Molecular dynamics with coupling to an external bath” J. Chem. Phys. 81, 3684-3690 (1984)
[11] N. Guex, M. C. Peitsch, “SWISS-MODEL and the Swiss-Pdb Viewer: An environment for comparative protein modeling” Electrophoresis 18, 2714-2723 (1997)
[12] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, T. E. Ferrin, “UCSF Chimera-a visualization system for exploratory research and analysis” J. Comput. Chem. 25, 1605-1612 (2004)
[13] A. Šali, T. L. Blundell, “Comparative protein modelling by satisfaction of spatial restraints” J. Mol. Biol. 234, 779-815 (1993)
[14] W. Kabsch, C. Sander, “Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features” Biopolymers 22, 2577-2637 (1983)
[15] W. G. Touw, C. Baakman, J. Black, T. A. H. te Beek, E. Krieger, R. P. Joosten, G. Vriend, “A series of PDB related databases for everyday needs” Nucleic Acids Research 43, D364-D368 (2015)
[16] M. Tiberti, G. Invernizzi, M. Lambrughi, Y. Inbar, G. Schreiber, E. Papaleo, “Py-Interaph: A framework for the analysis of interaction networks in structural ensembles of proteins” J. Chem. Inf. Model. 54, 1537-1551 (2014)
[17] J. Salamanca Viloria, M. F. Allega, M. Lambrughi, E. Papaleo, “An optimal distance cutoff for contact-based Protein Structure Networks using side-chain centers of mass” Sci. Rep. 7, 2838 (2017)
[18] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, T. Ideker, “Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks” Genome Research 13, 2498-2504 (2003)
[19] **H. Otaki**, Y. Taguchi, N. Nishida, “Conformation-dependent influences of hydrophobic amino acids in two in-register parallel β-sheet amyloids, an α-synuclein amyloid and a local structural model of PrP^{Sc}” bioRxiv: 10.1101/758938

FDPS を用いた大規模自重崩壊モデルの計算速度に関する研究

平 田　紗 椰

神戸大学大学院 工学研究科 修士課程学生

１．はじめに

斜面崩壊発生後に迅速に救出作業の意思決定を支援する目的で数値シミュレーションを行うためには，計算精度，計算規模，計算速度が必要である。

本研究では，野中(2020) の **SPH 法**を用いた土塊の自重崩壊シミュレーションに対し，大規模計算に備えて粒子間距離を小さくし粒子数を増やして大規模計算した結果の計算速度を検証する。大規模計算を行うためにスーパーコンピュータ Oakforest-PACS を用いた。

２．SPH（Smoothed Particle Hydrodynamics）法を用いた弾塑性解析

（1）土の支配方程式

連続体力学に基づく固体の各支配方程式と **SPH 法**で使用する離散式を記載する。上付き文字 *I*，*J* はそれぞれ中心粒子 *I*，周辺粒子 *J* の物理量であることを表している。

質量保存則は，

$$\frac{D\rho}{Dt} = -\rho\frac{\partial v_i}{\partial x_i} \tag{1}$$

である。ここでρは密度，v_iは速度ベクトルの i 方向成分である。

式(1)を離散化すると，密度の変化率は

$$\frac{D\rho}{Dt} = \sum_{J=1}^{N} m^J\left(v_i^I - v_i^J\right)\frac{\partial W^{IJ}}{\partial x_j^I} \tag{2}$$

で求められる。mは質量，Wはカーネル関数である。

運動量保存則から導かれる運動方程式は，

$$\frac{Dv_i}{Dt} = \frac{1}{\rho}\frac{\partial \sigma_{ij}}{\partial x_j} + g_i \tag{3}$$

となる。σ_{ij}は応力テンソル，g_iは外力ベクトルでここでは重力を表す。

式(3)を離散化すると，

$$a_i^I = \frac{\partial v_i^I}{\partial t} = \sum_{J=1}^{N} m^J\left(\frac{\sigma_{ij}^I}{(\rho^I)^2} + \frac{\sigma_{ij}^J}{(\rho^J)^2}\right)\frac{\partial W^{IJ}}{\partial x_j^I} + g_i \tag{4}$$

となる。ここで，a_i^Iは加速度ベクトルである。

運動方程式の応力項にさらに人工粘性項と人工応力項を加える。人工粘性項は **SPH 法**で問題となっている数値振動を抑えるためである。

（2）弾塑性構成式

土が外力を受けて変形するときの応力－ひずみ関係について，本研究では弾塑性モデルを適用

する。降伏関数を F とすると，降伏が起こったときの条件式は$F = 0$となり塑性変形を，$F < 0$のときは弾性変形を起こすと考える。全ひずみは弾性ひずみと塑性ひずみに分解することができ，

$$\varepsilon_{ij} = \varepsilon_{ij}^{E} + \varepsilon_{ij}^{P} \tag{5}$$

がなりたつ。式(5)を増分で考えると，全ひずみ速度テンソル$\dot{\varepsilon}_{ij}$は弾性ひずみ速度テンソル$\dot{\varepsilon}_{ij}^{E}$と塑性ひずみ速度テンソル$\dot{\varepsilon}_{ij}^{P}$の和となるので，変形すると，

$$\dot{\varepsilon}_{ij}^{E} = \dot{\varepsilon}_{ij} - \dot{\varepsilon}_{ij}^{P} \tag{6}$$

となる。弾性域では，塑性ひずみ速度$\dot{\varepsilon}_{ij}^{P} = 0$であるため，全ひずみ速度がそのまま弾性ひずみ速度になる。

応力速度テンソル$\dot{\sigma}_{ij}$は四階のテンソルである弾性マトリックスD_{ijkl}^{E}を用いると，一般化したフックの法則より，

$$\begin{aligned}\dot{\sigma}_{ij} &= D_{ijkl}^{E}\dot{\varepsilon}_{kl}^{E} \\ &= \frac{\nu E}{(1+\nu)(1-2\nu)}\dot{\varepsilon}_{kk}\delta_{ij} + \frac{E}{1+\nu}\dot{\varepsilon}_{ij} \\ &= \left(K - \frac{2}{3}G\right)\dot{\varepsilon}_{kk}\delta_{ij} + 2G\dot{\varepsilon}_{ij} \\ &= 2G\dot{e}_{ij} + K\dot{\varepsilon}_{kk}\delta_{ij}\end{aligned} \tag{7}$$

と求められる。偏差ひずみ速度は$\dot{e}_{ij} = \dot{\varepsilon}_{ij} - \frac{1}{3}\dot{\varepsilon}_{kk}\delta_{ij}$である。

$$\delta_{ij} = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases} \tag{8}$$

の性質を持つ。また，νはポアソン比，Eはヤング率を表し，これら 2 つの定数と剛性率G，体積弾性率Kの間には

$$G = \frac{E}{2(1+\nu)} \quad \text{and} \quad K = \frac{E}{3(1-2\nu)} \tag{9}$$

という関係がある。$\dot{\varepsilon}_{kk}$は総和規約を適用する。

塑性域では，応力は変化しないので$F = 0$となる。式(6)を式(7)に代入すると，

$$\begin{aligned}\dot{\sigma}_{ij} &= D_{ijkl}^{E}\dot{\varepsilon}_{kl}^{E} \\ &= D_{ijkl}^{E}(\dot{\varepsilon}_{kl} - \dot{\varepsilon}_{kl}^{P})\end{aligned} \tag{10}$$

となる。ここで，塑性ひずみ速度テンソル$\dot{\varepsilon}_{ij}^{P}$は

$$\dot{\varepsilon}_{ij}^{P} = \dot{\lambda}\frac{\partial g_p}{\partial \sigma_{ij}} \tag{11}$$

で求められる。 $\dot{\lambda}$は塑性乗数λの変化率で非負のスカラー，g_pは塑性ポテンシャル関数である。

(3) Drucker-Prager モデル

降伏基準fの定義については様々なモデルが提案されている。本研究では Bui *et al.* (2008)を参照して，**Drucker-Prager モデル**を用いている。

Drucker-Prager モデルでは応力速度テンソル$\dot{\sigma}_{ij}$を

$$\dot{\sigma}_{ij} = 2G\dot{e}_{ij} + K\dot{\varepsilon}_{kk}\delta_{ij} - \dot{\lambda}\left(9K\sin\psi\,\delta_{ij} + \frac{G}{\sqrt{J_2}}s_{ij}\right) \tag{12}$$

で求めている。式中の塑性乗数の変化率$\dot{\lambda}$はスカラーで，

$$\dot{\lambda} = \frac{3\alpha_\phi K\dot{\varepsilon}_{kk} + \frac{G}{\sqrt{J_2}}s_{ij}\dot{\varepsilon}_{ij}}{27\alpha_\phi K\sin\psi + G} \tag{13}$$

である。

降伏関数Fは

$$F(I_1, J_2) = \sqrt{J_2} + \alpha_\phi I_1 - k_c \tag{14}$$

で表される。ここで，I_1とJ_2は応力の不変量というスカラー量で，それぞれコーシー応力テンソルの一次の不変量，偏差応力テンソルの 2 次の不変量といい，

$$I_1 = \sigma_{kk} = \sigma_{xx} + \sigma_{yy} + \sigma_{zz} \quad \text{and} \quad J_2 = \frac{1}{2}s_{ij}s_{ij} \tag{15}$$

で示される。s_{ij}は偏差応力で，応力テンソルから平均応力テンソルを引いた，

$$s_{ij} = \sigma_{ij} - \frac{1}{dim}\sigma_{kk}\delta_{ij} \tag{16}$$

で求められる。dimは次元数である。式(14)におけるα_ϕとk_cは材料定数である粘着力と内部摩擦角から求められる定数で，3 次元では，

$$\alpha_\phi = \frac{2\sin\phi}{\sqrt{3}(3-\sin\phi)} \quad \text{and} \quad k_c = \frac{6c\cos\phi}{\sqrt{3}(3-\sin\phi)} \tag{17}$$

を用いる。また，非関連流れ則を用いるので，塑性ポテンシャルgは，

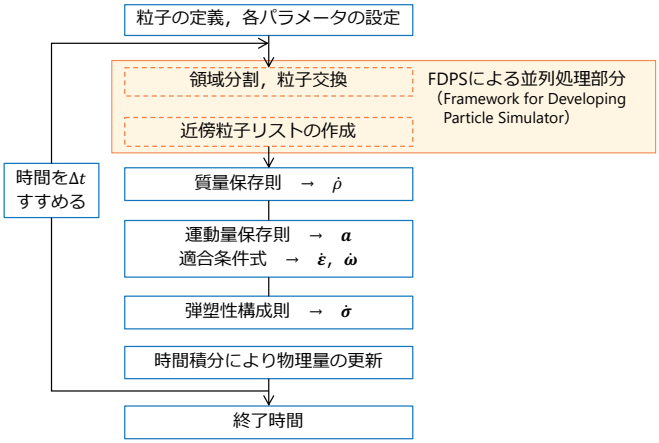
$$g = \sqrt{J_2} + 3I_1\sin\psi \tag{18}$$

で表し，ψはダイレタンシー角で，本研究では$\psi = 0$である。非関連流れ則において，ダイレタンシー角が 0° の条件は材料が塑性非圧縮であることを表す。

（4）数値計算の流れ

数値計算を実施するために，全粒子の初期配置や各パラメータの設定を行う。次に，並列計算を行う際に各プロセスが計算を担当する領域を決めて必要な粒子情報を通信分配する領域分割，影響半径内にある粒子を探索し相互作用計算を行う粒子のリストを作成する近傍粒子リストの作成は，いずれも **FDPS** が自動で並列処理を行う。**FDPS** については牧野(2017)，野村ら(2020)，行方ら(2020)などを参考にされたい。第 1 図にシミュレーションのフローチャートを示す。

FDPS に導入する物理過程では，影響半径内の粒子間で質量保存則より密度の変化率，運動方程式より加速度ベクトル，適合条件式よりひずみ速度テンソルの計算の順に相互作用計算を行う。さらに，得られたひずみ速度テンソルを用いて弾塑性構成式から応力速度テンソルを求める。各時間ステップの最後に，求めた各物理量の増分に対して時間積分を行うことで，密度，速度ベクトル，位置ベクトル，ひずみテンソル，応力テンソルの更新を行う。これらを時間ステップが終了条件に達するまで繰り返し計算を行う。



第 1 図： 数値計算のフロー

３．土の自重崩壊シミュレーションのスケーリング検証

プロセス数に対する粒子数を一定にし，粒子数の増加と計算時間の関係を検証した。

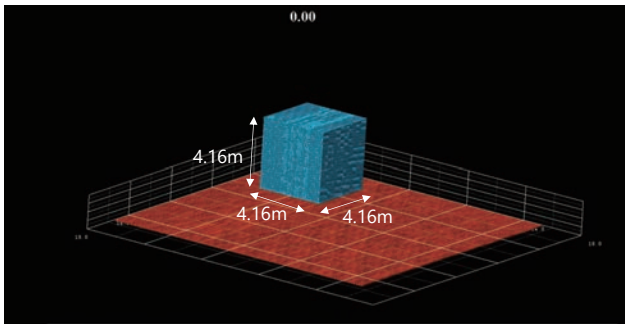
シミュレーションパラメータに第 1 表の値を用いて，プロセスあたりの粒子数を一定にし，粒子数の増加と計算時間の関係を検証した。32 プロセス，327,232 粒子から 2,048 プロセス，11,984,896 粒子のシミュレーション結果を第 2 表に示し，その初期状態を第 2 図に，1 秒後の状態を第 3 図に示す。第 2 図と第 3 図で，青色で示されている粒子は移動できる土粒子，茶色で示されている粒子は移動できない壁粒子である。

第 1 表： シミュレーションパラメータ

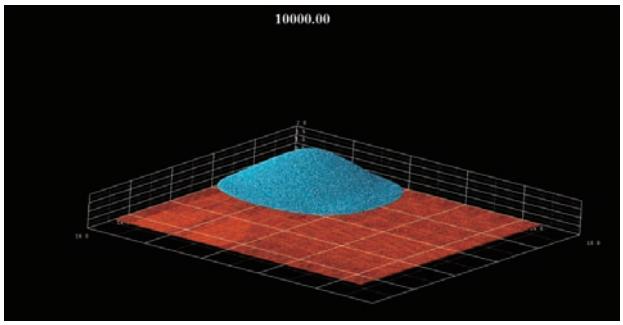
初期粒子間距離 Δd [m]	0.04
時間刻み Δt [s]	1.0×10^{-4}
ステップ数	10,000
密度 ρ[kgm^{-3}]	1850.0
ヤング率 E[kPa]	1800.0
ポアソン比 ν	0.3
粘着力 c[kPa]	0
内部摩擦角 ϕ[°]	25.0
ダイレタンシー角 ψ[°]	0.0

第 2 表： 粒子数と計算時間

プロセス数	32	256	2,048
土粒子数	140,608	1,124,864	8,998,912
土粒子数/プロセス数	4,394	4,394	4,394
壁粒子数	186,624	746,496	2,985,984
全粒子数	327,232	1,871,360	11,984,896
土粒子数/全粒子数	0.43	0.60	0.75
計算時間(sec)	6,689	9,806	29,742



第 2 図： 初期状態



第 3 図： 10,000 ステップ後の変形状態

本問題は Weak scaling であるため，計算時間が増加しないことが理想である。1 ノード 32 プロセスと比較して，8 ノード 256 プロセスでの計算時間は 1.6 倍，64 ノード 2,048 プロセスでの計算時間は 4.5 倍になった。計算時間が増加した要因としては，プロセスあたりの土粒子数を同じにして計算したので，プロセス数が増えるにつれ全粒子に対する土粒子の割合が増加している。そのため，移動している土粒子で領域間の粒子交換や近傍粒子リストの作成の計算時間が長くなったと考えた。

4. まとめ

解像度を上げるため粒子間距離を小さくした大規模シミュレーションをスーパーコンピュータ Oakforest-PACS によって実行した。64 ノード 2048 プロセスまでの並列計算を行い，Weak Scaling を計測したところプロセス数を 8 倍に増加したときに計算時間が 1.6 倍，プロセス数を

64 倍に増加したときに計算時間 4.5 倍となった。Scaling の際に動く粒子数をそろえて scaling ことにより，動かない粒子の割合が減ったことが計算時間の増加の原因と考えた.

謝辞 : 本研究を実施するための計算機資源として東京大学情報基盤センター，2020 年度若手・女性利用(女性研究者)枠を活用しました。また，JSPS 科研費 21H01436 の助成を受けました。ここに記して謝意を表します。

参 考 文 献

Bui, Ha H., Fukagawa, R., Sako, K. and Ohno, S., “Lagrangian meshfree particles method (SPH) for large deformation and failure flows of geomaterial using elastic-plastic soil constitutive model”, *International Journal for Numerical and Analytical Methods in Geomechanic*s, **Vol. 32**, pp. 1537-1570, 2008

行方大輔，坂本亮，中村孝史，木村耕行，似鳥啓吾，野村昴太郎，坪内美幸，牧野淳一郎『粒子法シミュレーションコード開発のためのフレームワーク(FDPS)の開発』, 情報処理学会研究報告, **Vol.2018-HPC-167，No. 22**，2018

野中沙樹『SPH シミュレーションによる斜面崩壊の定量的評価に関する研究』，63p.，神戸大学大学院工学研究科修士論文， 2020

野村昴太郎，沼田龍介，八柳祐一，行方大輔，岩澤全規，牧野淳一郎『超並列粒子法シミュレーションプログラム自動生成ツールの紹介』，*Journal of Plasma Fusion Research*, **Vol. 96，No. 2**，pp. 57–63，2020

牧野淳一郎『連載「FDPS 入門(1)」』, 分子シミュレーション研究会会誌 “アンサンブル”, **Vol. 19, No. 2**，2017

Exploring Communication-Computation Overlap in Parallel Iterative Solvers on Manycore CPUs using Asynchronous Progress Control

堀越将司 [a)]，Balazs Gerofi [b)]，石川裕 [c)]，中島研吾 [b,d)]

(a) インテル，(b) 理化学研究所，(c) 国立情報学研究所，(d) 東京大学情報基盤センター

本稿では，2019 年 5 月，2021 年 10 月に実施された大規模 HPC チャレンジの結果を報告する。本大規模 HPC チャレンジの成果をまとめた論文「Masashi Horikoshi, Balazs Gerofi, Yutaka Ishikawa and Kengo Nakajima, Exploring Communication-Computation Overlap in Parallel Iterative Solvers on Manycore CPUs using Asynchronous Progress Control」は，2022 年 1 月にオンラインで開催された HPC Asia 2022 でのワークショップ IXPUG（Intel eXtreme Performance Users Group）[1] に採択され，Proceedings（ACM Digital Library）に掲載された。本稿は出版元である ACM の許可を受け次ページ以降に上記論文の全文を掲載するものである。

1) https://www.ixpug.org/events/ixpug-hpc-asia-2022

Exploring Communication-Computation Overlap in Parallel Iterative Solvers on Manycore CPUs using Asynchronous Progress Control

Masashi Horikoshi
Accelerated Computing Systems and Graphics (AXG) Group, Intel Corporation
Japan
Masashi.Horikoshi@intel.com

Balazs Gerofi
RIKEN Center for Computational Science (R-CCS)
Japan
bgerofi@riken.jp

Yutaka Ishikawa
National Institute of Informatics
Japan
yutaka_ishikawa@nii.ac.jp

Kengo Nakajima
Information Technology Center, The University of Tokyo/ RIKEN Center for Computational Science (R-CCS)
Japan
nakajima@cc.u-tokyo.ac.jp

ABSTRACT

Preconditioned parallel solvers based on the Krylov iterative method are widely used in scientific and engineering applications. Communication overhead is a critical issue when executing these solvers on large-scale massively parallel supercomputers. In this work, we investigate communication-computation overlapping by *asynchronous progress control* to various types of preconditioned conjugate gradient methods for parallel finite-element applications. Performance of the developed method is evaluated using up to 4,096 nodes of the Oakforest-PACS system at JCAHPC, equipped with Intel® Xeon Phi™ Manycore Processors. We show that the performance of the iterative solver can be improved by up to 38% on 4,096 nodes. We apply the IHK/McKernel lightweight multi-kernel operating system and find that it can provide 20% and 6% improvements on 2,048 and 4,096 node respectively. Furthermore, we investigate the effects of asynchronous communication progression on the IHK/McKernel and find that it can provide an 2-3% improvement from 128 node to 1,024 node.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; **Network measurement**; • **Applied computing** → **Earth and atmospheric sciences**; **Engineering**.

KEYWORDS

iterative solver, MPI, non-blocking communication, asynchronous progress threads, lightweight kernel

HPCAsia 2022 Workshop, January 11–14, 2022, Virtual Event, Japan

ACM ISBN 978-1-4503-9564-9/22/01…$15.00
https://doi.org/10.1145/3503470.3503474

ACM Reference Format:
Masashi Horikoshi, Balazs Gerofi, Yutaka Ishikawa, and Kengo Nakajima. 2022. Exploring Communication-Computation Overlap in Parallel Iterative Solvers on Manycore CPUs using Asynchronous Progress Control. In *International Conference on High Performance Computing in Asia-Pacific Region Workshops (HPCAsia 2022 Workshop), January 11–14, 2022, Virtual Event, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3503470.3503474

1 INTRODUCTION

Preconditioned parallel solvers based on the Krylov iterative method are widely used in scientific and engineering applications. Communication overhead is a critical issue when executing these solvers on large-scale massively parallel supercomputers. In the present work, we introduced communication-computation overlapping by *asynchronous progress control* [1], which is supported by Intel® MPI Library from version 2019, to various types of preconditioned Krylov iterative methods, such as *pipelined* conjugate gradient method [2]. We focus on optimization of global collective communications for dot products in parallel Krylov iterative solvers. Target linear equations are derived from GeoFEM/Cube [3], which is a parallel finite-element application on massively parallel supercomputers. Performance of the developed method was evaluated using up to 4,096 nodes of the Oakforest-PACS (OFP) system at JCAHPC with Intel Xeon Phi Manycore Processors[4], and the performance of the iterative solver was evaluated.

In the previous works [5][6], we examined the impact of the IHK/McKernel [10], lightweight multi-kernel OS developed at RIKEN R-CCS, which provides efficient and scalable execution environment on large-scale systems by reducing OS noise and communication overhead. Improvement of the performance of parallel multigrid solvers was 10-20% at 2,048 nodes of OFP. In the present work, effects of IHK/McKernal were also evaluated for Asynchronous Progress Control.

The rest of this paper is organized as follows. Section refsect:appl provides an overview of the target application and algorithms. Section 3 provides a brief summary of the target hardware system.

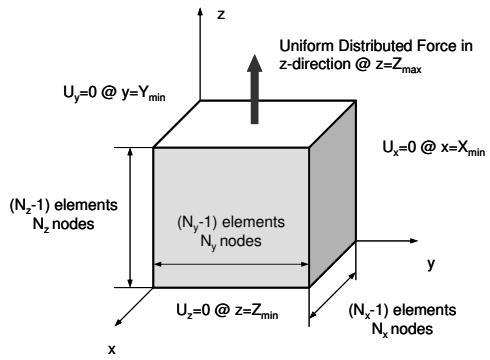


Figure 1: Simple cube geometry for 3D linear elasticity problems in GeoFEM/Cube [3]

Section 4 overviews Intel's Asynchronous Progress Control. Section 5 presents overview of IHK/McKernel, and special configuration of threads for Intel's Asynchronous Progress Control. Section 6 describes numerical experiments and results. Finally, Section 8 concludes the paper and offers future perspective.

2 TARGET APPLICATION AND ALGORITHMS

2.1 GeoFEM/Cube

GeoFEM/Cube [3] is the target application, and it solves 3D linear elasticity problems in simple cube geometries using a parallel FEM. Tri-linear hexahedral elements are used for the discretization. Material properties are defined as homogeneous, where Poisson's ratio is set to 0.30 for all elements and Young's modulus is 1.00. The boundary conditions are described in Fig 1. Large-scale linear equations with sparse matrices derived from the application are solved by preconditioned Krylov iterative methods . The original GeoFEM/Cube adopts the incomplete Cholesky conjugate gradient (ICCG) method [11] with preconditioning by incomplete Cholesky factorization without fill-ins (IC(0)) [11]. The additive Schwarz domain decomposition (ASDD) for overlapped regions [12] is introduced for stabilization of parallel IC(0) with block-Jacobi-type localization. The GeoFEM/Cube application is parallelized by domain decomposition using the Message-Passing Interface (MPI) [3]. In the OpenMP/MPI hybrid parallel programming model, multithreading by OpenMP is applied to each partitioned domain. GeoFEM/Cube is written in Fortran with MPI and OpenMP.

In GeoFEM/Cube, the entire model is divided into domains, and each domain is assigned to an MPI process, as shown in Fig.2. The local data structure in GeoFEM/Cube is node-based with overlapping elements, and as such is appropriate for the preconditioned iterative solvers used in GeoFEM/Cube (Fig.3).

GeoFEM/Cube generates distributed local meshes and coefficient matrices in a parallel manner using MPI. In GeoFEM/Cube, each MPI process has $(Nx/Px) \times (Ny/Py) \times (Nz/Pz)$ vertices, while the total number of vertices in each direction is (Nx, Ny, Nz) and the number of partitions in each direction is (Px, Py, Pz).

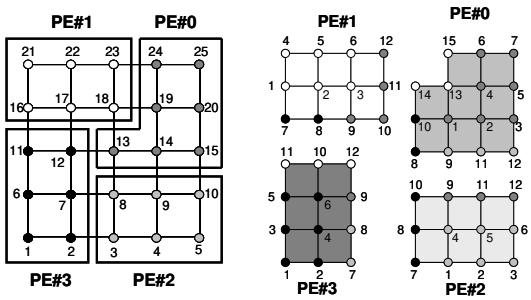


Figure 2: Node-based partitioning in GeoFEM/Cube [3]

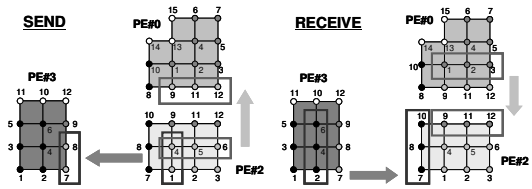


Figure 3: Communications among MPI processes [3]

2.2 Pipelined CG

Algorithm 1 shows the preconditioned conjugate gradient method (PCG) [11], which includes SpMV, dot products, the DAXPY computations, and preconditioning.

Algorithm 1 Preconditioned Conjugate Gradient Method (PCG) [11]

1: $r^{(0)} = b - Ax^{(0)}$; $u^{(0)} = M^{-1}r^{(0)}$; $p^{(0)} = u^{(0)}$; $\rho_0 = (r^{(0)}, u^{(0)})$
2: **for** $k = 0, 1, \ldots,$ until convergence **do:**
3: $q^{(k)} = Ap^{(k)}$
4: $\alpha_k = \rho_k / (p^{(k)}, q^{(k)})$;
5: $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$
6: $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k)} \Rightarrow check\ convergence: |r^{(k+1)}|$
7: $u^{(k+1)} = M^{-1} r^{(k+1)}$
8: $\rho_{k+1} = (r^{(k+1)}, u^{(k+1)})$
9: $\beta_{k+1} = \rho_{k+1} / \rho_k$
10: $p^{(k+1)} = u^{(k+1)} + \beta_{k+1} p^{(k)}$
11: **end for**

In computations on parallel computers with distributed memory, SpMV (Sparse Matrix-Vector Multiplication), dot products, and preconditioning may require communication. Although there are various types of communication patterns in preconditioning, SpMV relies upon point-to-point communication with neighbors and dot products rely upon global collective communication [3][11].

If the code is implemented by MPI, `MPI_Isend` and `MPI_Irecv` with `MPI_Wait/Waitall` are used in SpMV, and `MPI_Allreduce` is

used in dot products. The overhead for such communications is a critical issue on massively parallel supercomputers with more than 10^4 MPI processes.

In recent years, many algorithms and methods for avoiding and reducing communication overhead have been proposed. Methods based on the matrix powers kernel [13] reduce the number of global communications at each iteration by extending the halo region in Figs. 2 and 3. They generally require complicated data structures and are not suitable for general preconditioning methods, such as incomplete LU (ILU).

In [2], several methods for conjugate gradient methods, which reduce overhead of global collective communications for dot products, are introduced. The sequence of Krylov iterations is changed using recurrence relations without changing the original algorithm. Because the order of computation is changed, rounding errors are propagated differently. Therefore, convergence may be affected in ill-conditioned problems. Authors applied computing with single precision to such algorithms, and results show that original CG is the most robust [14]. This does not happen for the cases with double precision for ill-conditioned problems in the present work.

Algorithm 2 shows PCG using Chronopouos/Gear CG method [15], where 2 dot products are combined into a single reduction. Thus, overhead for calling `MPI_Allreduce` may be reduced.

Algorithm 2 Preconditioned Chronopoulos/Gear Conjugate Gradient Method [15]

1: $r^{(0)} = b - Ax^{(0)}$; $u^{(0)} = M^{-1}r^{(0)}$; $p^{(0)} = u^{(0)}$; $q^{(0)} = Ap^{(0)}$; $\gamma_0 = (r^{(0)}$
2: **for** $k = 0, 1, \ldots,$ until convergence **do:**
3: $p^{(k+1)} = u^{(k)} + \beta_k p^{(k)}$
4: $q^{(k+1)} = w^{(k)} + \beta_k q^{(k)}$
5: $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k+1)}$
6: $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k+1)} \Rightarrow check\ convergence: |r^{(k+1)}|$
7: $u^{(k+1)} = M^{-1}r^{(k+1)}$
8: $w^{(k+1)} = Au^{(k+1)}$
9: $\gamma_{k+1} = (r^{(k+1)}, u^{(k+1)})$
10: $\delta_{k+1} = (w^{(k+1)}, u^{(k+1)})$
11: $\beta_{k+1} = \gamma_{k+1}/\gamma_k$
12: $\alpha_{k+1} = \gamma_{k+1}/(\delta_{k+1} - \beta_{k+1}\gamma_{k+1}/\alpha_k)$
13: **end for**

The pipelined method [2] reduces communications overhead by overlapping global collective communications and computations.

Algorithm 3 shows Preconditioned Pipeined CG method [2]. This method is derived from Algorithm 2.

In pipelined CG [2], collective communication for dot products can be overlapped with heavier computations, such as SpMV and preconditioning. In the original CG algorithm (Algorithm1), dot products ($\alpha_k = \rho_k/(p^{(k)}, q^{(k)})$ in line-4, and $\rho_{k+1} = (r^{(k+1)}, u^{(k+1)})$ in line-8) are used in the next lines (line-5 and line-9), while they are used after in SpMV and preconditioning in the pipelined algorithm. $\delta_k = (p^{(k)}, q^{(k)})$ is defined in line-3, and it it used in line-5, therefore computing of δ_k can be overlapped with $z^{(k)} = M^{-1}q^{(k)}$ in line-4. $\gamma_{k+1} = (r^{(k+1)}, u^{(k+1)})$ is defined in line-9, and it is used in line-11 after $w^{(k+1)} = Au^{(k+1)}$ in line-10.

Asynchronous collective communication (e.g., `MPI_Iallreduce`) supported in MPI-3 is effective for such procedures.

In [16], the authors implemented pipelined CG to the original GeoFEM/Cube with ICCG and ASDD using up to 12,288 cores (384 nodes) of the Reedbush-U system [17] with Intel® Xeon® E5-2695v4 (code name: Broadwell-EP) CPUs using Intel® MPI 2017. Fig. 4 compares the original and pipelined CG method for strong scaling. Pipelined CG provides much better scalability than the original CG. Pipelined CG is effective in the very limited case of strong scaling, where the problem size per MPI process is very small.

Algorithm 3 Preconditioned Pipelined Conjugate Gradient Method [2]

1: $r^{(0)} = b - Ax^{(0)}$; $u^{(0)} = M^{-1}r^{(0)}$; $p^{(0)} = u^{(0)}$; $q^{(k)} = Ap^{(k)}$; $w^{(0)} = Au^{(0)}$; $\alpha_0 = (r^{(0)}, u^{(0)})/(w^{(0)}, u^{(0)})$; $\beta_0 = 0$; $\gamma_0 = (r^{(0)}, u^{(0)})$
2: **for** $k = 0, 1, \ldots,$ until convergence **do:**
3: $\delta_k = (p^{(k)}, q^{(k)})$
4: $z^{(k)} = M^{-1}q^{(k)}$
5: $\alpha_k = \gamma_k/\delta_k$
6: $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$
7: $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k)} \Rightarrow check\ convergence: |r^{(k+1)}|$
8: $u^{(k+1)} = u^{(k)} - \alpha_k z^{(k)}$
9: $\gamma_{k+1} = (r^{(k+1)}, u^{(k+1)})$
10: $w^{(k+1)} = Au^{(k+1)}$
11: $\beta_{k+1} = \gamma_{k+1}/\gamma_k$
12: $p^{(k+1)} = u^{(k+1)} + \beta_{k+1}p^{(k)}$
13: $q^{(k+1)} = w^{(k+1)} + \beta_{k+1}q^{(k)}$
14: **end for**

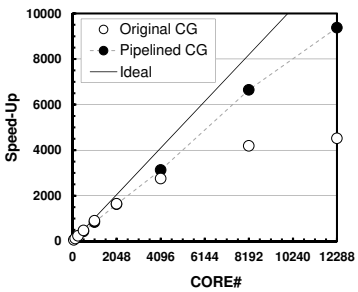


Figure 4: Effects of pipelined CG [16] for GeoFEM/Cube with ICCG and ASDD using up to 12,288 cores (384 nodes) of Reedbush-U [16] and strong scaling; total problem size: 28,311,552 DOF

Finally, Algorithm 4 shows Preconditioned Gropp's CG method [2][18]. While this method is similar to Algorithm 3, computations are smaller than those of Algorithm 3. Dot products in line-3 and line-4 (γ_k, δ_k) are overlapped with preconditioning in line-5 and SpMV in line-6.

In the present work, following four algorithms for the conjugate gradient method are evaluated. Each method includes one *SpMV*

(Sparse Matrix Vector Multiply), one preconditioning and three dot products for each iteration. Number of *DAXPY*'s (constant times a vector plus a vector (axpy) in double-precision) in each method as follows. 8 DAXPY's in a iteration of Pipelined CG (Alg.3), while it is 3 in the original CG (Alg.1):

- Algorithm 1: Original CG (3 DAXPY's in one iteration) [11]
- Algorithm 2: Chronopoulos/Gear CG (4 DAXPY's) [15]
- Algorithm 3: Pipelined CG (8 DAXPY's) [2]
- Algorithm 4: Gropp's CG (5 DAXPY's) [2][18]

Algorithm 4 Preconditioned Gropp's Conjugate Gradient Method [2][18]

$r^{(0)} = b - Ax^{(0)}$; $u^{(0)} = M^{-1}r^{(0)}$; $w^{(0)} = Au^{(0)}$; $z^{(0)} = q^{(0)} = s^{(0)} = p^{(0)} = 0$
for $k = 0, 1, \ldots,$ until convergence **do:**
  $\gamma_k = (r^{(k)}, u^{(k)})$
  $\delta_k = (w^{(k)}, u^{(k)})$
  $m^{(k)} = M^{-1}w^{(k)}$
  $n^{(k)} = Am^{(k)}$
  if $k > 0$ **then**
    $\beta_k = \gamma_k/\gamma_{k-1}$; $\alpha_k = \gamma_k/(\delta_k - \beta_k\gamma_k/\alpha_{k-1})$
  else
    $\beta_0 = 0$; $\alpha_k = \gamma_k/\delta_k$
  end if
  $z^{(k+1)} = n^{(k)} + \beta_k z^{(k)}$
  $s^{(k+1)} = m^{(k)} + \beta_k s^{(k)}$
  $q^{(k+1)} = w^{(k)} + \beta_k q^{(k)}$
  $p^{(k+1)} = u^{(k)} + \beta_k p^{(k)}$
  $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$
  $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k)} \Rightarrow$ *check convergence* : $|r^{(k+1)}|$
  $u^{(k+1)} = u^{(k)} - \alpha_k s^{(k)}$
  $w^{(k+1)} = w^{(k)} - \alpha_k z^{(k)}$
end for

3 TARGET HARDWARE (OAKFOREST-PACS, OFP)

The Oakforest-PACS system (OFP) [4] is the premiere supercomputer system at the *Joint Center for Advanced High-Performance Computing (JCAHPC)* [4] , which was established by the University of Tokyo and University of Tsukuba. The system consists of 8,208 nodes of Intel Xeon Phi 7250 (code name: Knights Landing, or *KNL*), and *Intel® Omni-Path Architecture* (Intel® OPA) provides a 100 Gbps interconnection in a fat-tree topology with full bisection bandwidth. Each Xeon Phi 7250 node is built using 68 modified Atom® (code name: Silvermont) cores running at 1.4 GHz, and the memory unit consists of 96 GB of DDR4 RAM and 16 GB of stacked 3D MCDRAM, which can be utilized as an L3 cache or high-bandwidth memory. Each core has two 512-bit vector units and supports AVX-512 SIMD instructions. Each core can host four threads (*i.e.,* 272 overall logical CPUs on the entire chip) and is equipped with 2 512-bit floating-point vector ALU. The total theoretical computational performance is 25 PFLOPS, and the system achieved 13.55 PFLOPS on the HPL benchmark. In the present work, only MCDRAM was used for memory in the flat/quadrant mode on the OFP. Intel's compiler and MPI library (2018) were used. Table 1 summarizes the specifications of each node of OFP.

Table 1: Summary of the performance of single node of the Oakforest-PACS(OFP)

Architecture	Intel Xeon Phi 7250 (Knights Landing)
Frequency(GHz)	1.40
Core # /CPU (socket) (Maximum Effective Thread #)	68 (272)
Peak Performance (GFLOPS)	3,046.4
Memory Size (GB)	MCDRAM:16,DDR4:96
Memory Bandwidth (GB/sec)	MCDRAM:490 [20], DDR4:84
Compiler & MPI Library	Intel® Parallel Studio 2019 XE, Intel MPI 2019

4 ASYNCHRONOUS PROGRESS CONTROL

4.1 Overview

MPI Non-blocking communication in application thread (e.g. for `MPI_Isend`) is expected to run asynchronously. To do that, progress thread is required if MPI communication is not completely offloaded. But turning on the progress thread (e.g. for `MPI_test`) requires `MPI_THREAD_MULTIPLE`. But in most case, MPI progress thread is not practically used due to poor MPI performance due to serialization around one queue for all threads, thread switching overhead, and process-wide MPI objects management overhead. Because those restrictions are closely related with MPI 3.1 standard (defined as "threads are not separately addressable: a rank in a send or receive call identifies a process, not a thread"). Those history and practices are well summarized in [21]. To overcome such restrictions, Intel MPI 2019 developed new asynchronous progress design, which is based on MPICH asynchronous design. From Intel MPI version 2019, it supports the new asynchronous progress threads that users to manage communication in parallel with application's computation to achieve better communication and computation overlapping. Asynchronous progress control on the Intel MPI has a full support for MPI point-to-point operations, blocking collectives, and a partial support for non-blocking collectives (`MPI_Ibcast`, `MPI_Ireduce`, and `MPI_Iallreduce`).

Setting the `I_MPI_ASYNC_PROGRESS_PIN` environment variable on the Intel MPI allows to control a pinning of the asynchronous progress threads to logical processor cores. By using this feature [1], we can provide separate dedicated cores for asynchronous progress threads.

4.2 Core Configuration

By default, Intel MPI allocates cpu core resource to the asynchronous progress threads from last logical core. For the 68 cores Intel

[1]To enable asynchronous progress control on the Intel MPI, environment variable I_MPI_ASYNC_PROGRESS=on, after loading release_mt environment, is required.

Xeon Phi processor, it has 272 logical cores so the default pinning values are from 271 to 264 in the case of 8 MPI processes and 8 asynchronous progress threads, for example. It depends on application but there are several ways to allocate logical cores for asynchronous progress threads on the many core processor. For simplicity on the GeoFEM application, we fixed the number of MPI process as 8 and used 8 OpenMP threads per MPI process. Total 64 logical core by using same number of physical core for better GeoFEM performance (1 hardware thread per physical core used).

We exclude first two physical and last two physical cores of the Xeon Phi from GeoFEM computation by using environment variable `I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=0,1,68,69,136,137,204,205` in order to avoid first core (logical cores 0,68,136, and 204) which is affected by OS noise and jitters. Logical cores from 2 to 65 are used for GeoFEM computation. Fig. 5 illustrates such situation as experimental setup. We have tested 5 different mapping of 8 asynchronous progress threads (1 progress thread per MPI process) in addition to the case with asynchronous progress control. First method (Fig. 6) is using default setup of Intel MPI which uses from logical core 268 to 271. Second method (Fig. 7) is using first two physical cores (logical core 0,1,68,69,136,137,204 and 205). Since first two cores are not used by computation, we can allocate 8 logical cores by 2 free physical cores.Third method (Fig. 8) is using last two physical cores (66,67,134,135,202,203,270 and 271). This method is basically same as second method but it is not affected by OS noise and jitters. Forth method (Fig. 9) is using first two and last two cores (0,1,68,69,66,67,134 and 135). This method is using all 4 free physical core to provide 8 asynchronous progress threads. Last method (Fig. 10) is using same physical core as MPI process (138,146,154,162,170,178,186 and 192). This method does not use any additional physical core for asynchronous progress threads and use other hardware thread (e.g. hardware thread #2) on the same physical core of MPI process. For example, logical core 2 (hardware thread #0 on the physical core 2) is used for MPI process 0, and logical core 138 (hardware thread #2 on the physical core 2) is used one asynchronous progress thread. Since using same physical core for MPI communication and computation, this method is expected to have better cache hit and to reduce internal data transfer during overlapping. Table 2 is the list of core numbers for `I_MPI_ASYNC_PROGRESS_PIN`.

Table 2: Summary of mapping method of asynchronous progress thread

Mapping method	`I_MPI_ASYNC_PROGRESS_PIN=`
#0. Without asynchronous thread	N/A (`I_MPI_ASYNC_PROGRESS=off`)
#1. Intel MPI default	(271,270,269,268,267,266,265,264)
#2. First 2 physical cores	0,1,68,69,136,137,204,205
#3. Last 2 physical cores	66,67,134,135,202,203,270,271
#4. First 2 and last 2 cores	0,1,68,69,66,67,134,135
#5. Using other hardware thread on same core	138,146,154,162,170,178,186,192

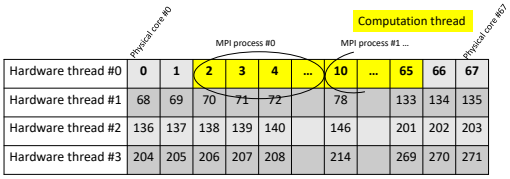


Figure 5: Figure of core mapping method #0 on Xeon Phi processor

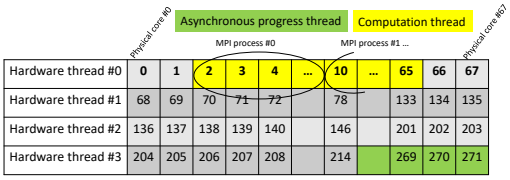


Figure 6: Figure of core mapping method #1 on Xeon Phi processor

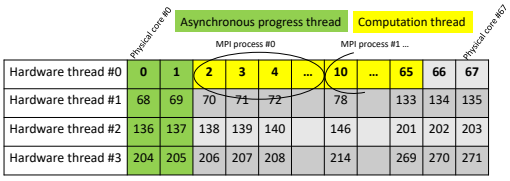


Figure 7: Figure of core mapping method #2 on Xeon Phi processor

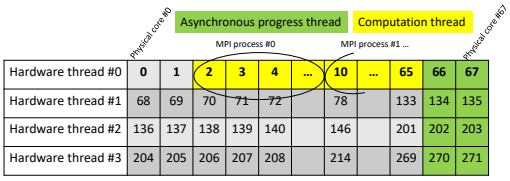


Figure 8: Figure of core mapping method #3 on Xeon Phi processor

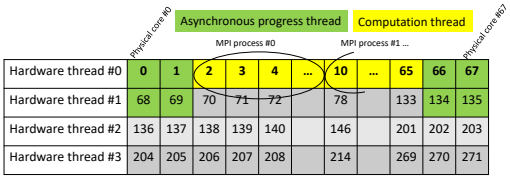
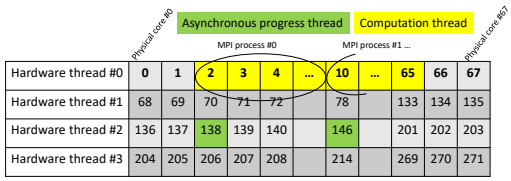


Figure 9: Figure of core mapping method #4 on Xeon Phi processor



Hardware thread #0	**0**	**1**	**2**	**3**	**4**	**...**	**10**	**...**	**65**	**66**	**67**
Hardware thread #1	68	69	70	71	72		78		133	134	135
Hardware thread #2	136	137	138	139	140		146		201	202	203
Hardware thread #3	204	205	206	207	208		214		269	270	271

Figure 10: Figure of core mapping method #5 on Xeon Phi processor

4.3 Experiments of various mapping methods on 512 node

We evaluated which mapping method is best for GeoFEM by using smaller test case on smaller cluster size. We set a problem size of GeoFEM as (256, 128, 128) for 512 node test as smaller proxy of full (512, 256, 256) size, which is equivalent to 100,663,296 DOFs. We investigate all method from #0 to #5 on the 512 node by using best performing number during 5 executions for each algorithm. Figure 11 shows relative performance of algorithm 4:Gropp's CG which is using `MPI_Iallreduce`. Except mapping method #2, all methods of asynchronous progress control enabled shows better performance than method #0. Method #4 is best performing core configuration with 37% improvement. While method #1 and #5 need only free logical cores, method #3 and #4 need more hardware resource as free physical cores. Method #4 needs 2 more physical cores than Method #3. It is reasonable results that more hard ware resource gives more performance improvement. We are not sure why method #2 showed poor performance. One possible reason is OS noise and jitters. Message size of GeoFEM's `MPI_Iallreduce` is 16 byte. It is well known such small size global synchronized collective communications are affected by OS noise and jitters [22].

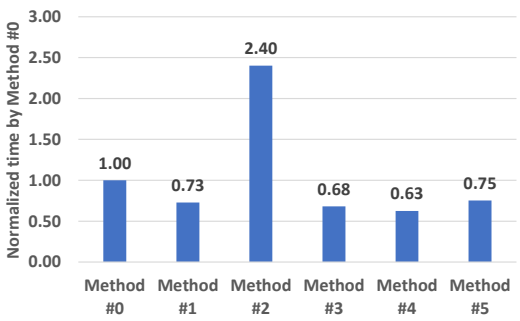


Figure 11: Result of 512 node experiment. Method #0 to #5 shows relative performance. Normalized by Method #0 (without asynchronous progress control).

5 IHK/MCKERNEL

This section gives a general overview of IHK/McKernel and describes developments targeted for asynchronous communication threads on Oakforest-PACS.

5.1 Overview

The IHK/McKernel multi-kernel operating system is comprised of two main components. Interface for Heterogeneous Kernels (IHK) [7], a low-level software infrastructure, provides capabilities for partitioning resources in a many-core environment (*e.g.*, CPU cores and physical memory) and it enables management of lightweight kernels. McKernel is a lightweight co-kernel developed on top of IHK. An overview of the multi-kernel architecture is depicted in Fig. 12.

IHK is capable of allocating and releasing host resources dynamically and no reboot of the host machine is required when altering configuration. It is implemented as a collection of Linux kernel modules without any modifications to the Linux kernel itself, which enables straightforward deployment of the multi-kernel stack on a wide range of Linux distributions. Besides resource and LWK management, IHK also facilitates an Inter-kernel Communication (IKC) layer, which is used for implementing system call delegation (discussed below).

McKernel has been developed from scratch and while it is designed explicitly for high-performance computing workloads it retains a Linux compatible application binary interface (ABI) so that it can execute unmodified Linux binaries. There is no need for recompiling applications or for any McKernel specific libraries. McKernel implements only a small set of performance sensitive system calls and the rest of the OS services are delegated to Linux. Specifically, McKernel implements memory management, it supports processes and multi-threading, it has a simple round-robin co-operative (tick-less) scheduler, and it supports standard POSIX signaling. It also implements inter-process memory mappings and it offers interfaces for accessing hardware performance counters.

For each OS process executed on McKernel there is a process running on Linux, which we call the *proxy-process*. The proxy process' main role is to assist system call offloading. Essentially, it provides the execution context on behalf of the application so that offloaded system calls can be invoked in Linux. For more information on system call offloading, refer to [8]. The proxy process also provides means for Linux to maintain various state information that would have to be otherwise kept track of in the co-kernel. McKernel for instance has no notion of file descriptors, but it simply returns the number it receives from the proxy process during the execution of an `open()` system call. The actual set of open files (*i.e.*, file descriptor table, file positions, etc.) are managed by the Linux kernel. Relying on the proxy process, McKernel provides transparent access to Linux device drivers not only in the form of offloaded system calls (*e.g.*, through `write()` or `ioctl()`), but also via direct device mappings. Details of the device mapping mechanism has been described elsewhere [9].

5.2 Support for Asynchronous Progress

As shown in Fig. 12, IHK partitions CPU cores into Linux and lightweight kernel (LWK) domains. Only the LWK CPUs are exposed to

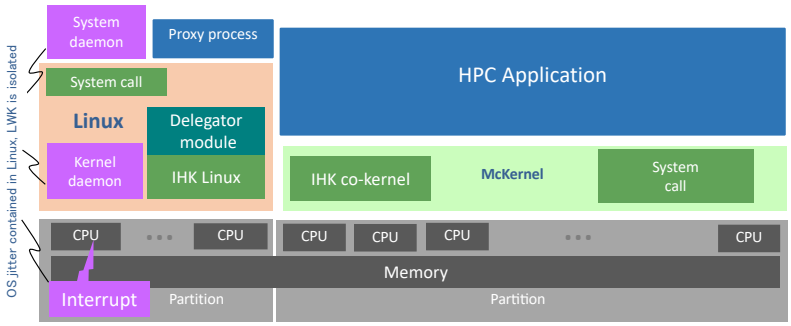


Figure 12: Architectural overview of IHK/McKernel

application running on McKernel, *e.g.*, on the Xeon Phi processor used in this study only 256 logical CPU cores from the overall 272 are visible in McKernel. In addition, McKernel renumbers CPUs and provides an automatic process pinning mechanism that integrates with MPI to ease the burden on users for dealing with CPU numbering related issues.

With respect to asynchronous progress threads, the main challenge that had to be addressed in McKernel was the enablement of additional CPU cores in LWK without directly exposing them to computation threads of the application. For this purpose McKernel's internal CPU management code as well as the process pinning mechanism has been enhanced for asynchronous progress awareness. Specifically, CPU cores dedicated to auxiliary threads are kept transparent from application code and MPI progress threads are pinned automatically to those CPUs. We use the second tile of the Xeon Phi and pin one progress thread per logical CPU on each core.

6 NUMERICAL EXPERIMENTS

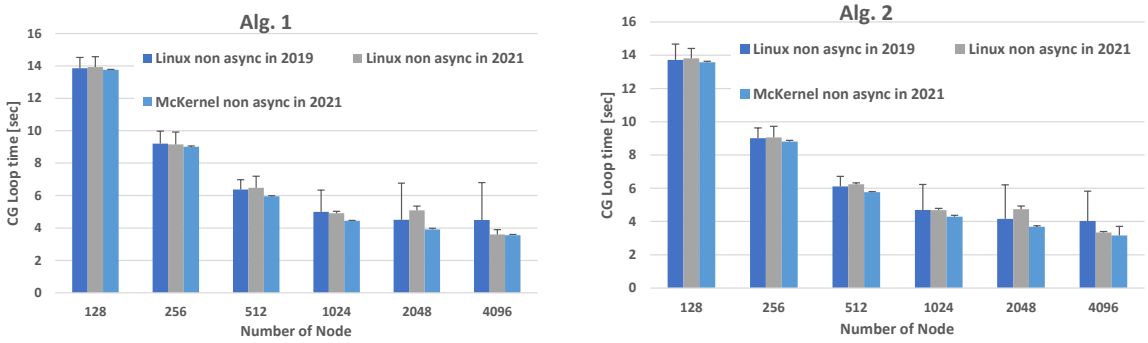
6.1 Performance results on Linux (measured in 2019)

Since method #4 is the best core mapping so we measured real large problem size up to 4096 node by using asynchronous progress thread with method #4 in addition to method #0 as a baseline. First measurement was done on May 2019 by using Intel tools version 2019 Update 1. We fixed the problem size as (512, 256, 256), which has 100,663,296 DOFs, and performed strong scaling measurements. Fig. 13a-14b show bar graphs as best performing numbers during 5 executions on each run, which have also error bars as worst performing numbers of 5 runs. Detailed performance numbers are also listed on the Table 7 in the appendix as a reference.

Since Alg. 1-4 do not have global asynchronous MPI calls (`MPI_Iallreduce`), but using `MPI_Allreduce`, it is not related with asynchronous progress setups, essentially. Fig. 13a-14b have measurement results for those algorithms marked as "Linux non async in 2019". We can see performance improvement (or decreasing CG Loop time) with growing the number of node from 128 to 2,048 node. From 2,048 to 4,096 node, we are seeing saturation of performance or worse performance by poor scaling. These performance saturations are derived by time increase of `MPI_Allreduce` on the large number of node executions. For example, by profiling on 2048 node, total time of `MPI_Allreduce` calls for Alg. 1 is 59% of total CG Loop time while it is 15% on 128 node. It is what we would like to hide or overlap into computation by using asynchronous progress threads.

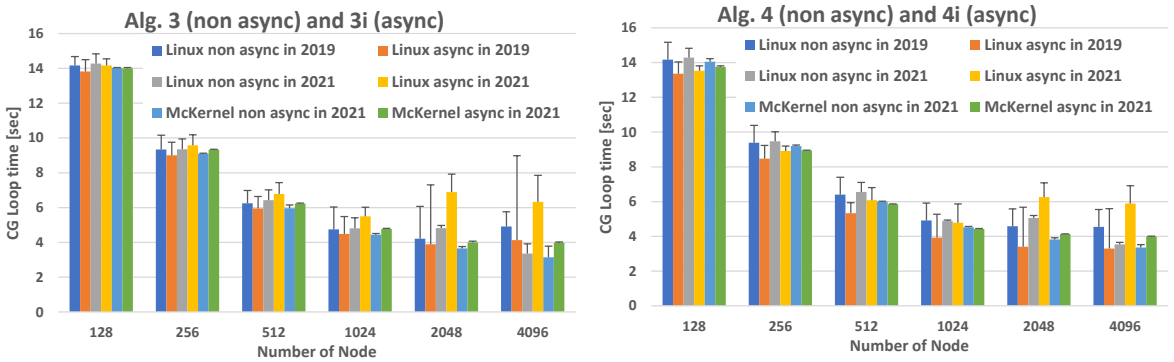
Performance of the original CG (Alg.1) saturates, as number of nodes is more than 2,048. On the contrast, Chronopoulos/Gear algorithm (Alg.2) keeps scalability up to 4,096 nodes, and much better than Alg.1, because communication overhead by `MPI_Allreduce` is reduced. Pipelined CG (Alg.3) and Gropp's CG (Alg.4) are generally slower than Alg.1, because the amount of computations per each iteration is larger than Alg.1. Alg.3i and Alg.4i with `MPI_Iallreduce` are also slower, but both of them are much faster than Alg.1, and competitive with Alg.2. The reason of this improvement is not clear, while `MPI_Iallreduce` may reduce the communication overhead at 4,096 nodes.

Effects of asynchronous progress threads are significant with more than 512 nodes, although performance of Alg.3i is not scaled at 4,096 nodes. Generally, the results show Alg.3i and Alg.4i provide much better scalability than Alg.1 by combination of `MPI_Iallreduce` and asynchronous progress threads. Fig. 14a-14b have the result of asynchronous progress threads for the Alg. 3i and 4i, which have asynchronous global MPI calls as `MPI_Iallreduce`. We can compare performance "Linux no async in 2019" and "Linux async in 2019" on Fig. 14a and 14b, respectively. We are seeing significant performance improvement on Alg. 4i while Alg. 3i was expected to have similar performance improvement by overlapping, but result of Alg. 3i showed smaller performance improvement than Alg. 4i. It is not clear why it shows such behavior. Table 3 show summarized performance improvement results which have comparison of original non asynchronous algorithm (Alg. 3 and 4) and asynchronous versions with asynchronous progress control (Alg. 3i and 4i). We successfully got up to 19% and 38% improvements for Alg. 3 to 3i and Alg. 4 to 4i, respectively, by overlapping effect of asynchronous progress control.



(a) Algorithm 1 (b) Algorithm 2

Figure 13: Results without asynchronous progress on Linux and McKernel



(a) Algorithms 3 and 3i (b) Algorithms 4 and 4i

Figure 14: Results comparing synchronous and asynchronous progress for algorithms 4 and 5 on Linux and McKernel

We can see growing size of error bars for all algorithms on top of "Linux no async in 2019" and "Linux async in 2019" in Fig. 13a-14b with increasing the number of node. In [5][6], we examined the impact of the IHK/McKernel [10], and seeing stable measurement results (small error bars) on large-scale systems by reducing OS noise and communication overhead. In the next section, we will discuss the efforts which we try applying IHK/McKernel for GeoFEM with asynchronous progress control.

6.2 Performance results on Linux (measured in 2021)

Since McKernel was required some modifications as described in Section 5, we needed revisited Linux measurements with newer version of environments including newer versions of BIOS, Firmware, OS, Interconnect driver (IFS), Parallel file system (Lustre) driver, macro task software (e.g. scheduler) and MPI library to get new baseline after 2 years. We used same method #4 as the best core

Table 3: Performance improvement of introducing overlaps by MPI_Iallreduce and asynchronous progress control on Linux (measured in 2019)

Node	Alg. 3 non asyc to 3i with async	Alg. 4 non async to 4i with async
128	1.03	1.06
256	1.04	1.11
512	1.05	1.20
1024	1.06	1.25
2048	1.08	1.34
4096	1.19	1.38

mapping with same large problem size up to 4096 node by using asynchronous progress control in addition to method #0 as a baseline similarly in 2019. This second measurement was done on Nov. 2021 by using Intel tools version 2019 Update 9. All performance numbers are listed on the Table 8 in the appendix as a reference.

Results marked as "Linux non async in 2021" on Fig. 13a-14b and "Linux async in 2021" on Fig. 14a-14b show the performance of result of re-measurements by using newer software and driver versions. We can see similar performance behaviors in Fig. 13a-14b without asynchronous progress control while seeing smaller run to run variances (smaller error bars). On Fig. 14a-14b, we can see worse performance results of "Linux non async in 2021" case than "Linux non async in 2019". We are not sure why `MPI_Iallreduce` version's GeoFEM (Alg. 3i and 4i) show poor performance than 2019. Since almost of all software environments are changing from 2019, it is very difficult to identify the root cause. Also due to security fixes and patches in two years, we can not revert entire system same as 2019.

We can compare the effect of asynchronous thread control, similarly, "Linux async in 2019" case and "Linux async in 2021" case on Fig. 14a-14b. Alg. 3i and 4i show better performance by using asynchronous progress threads up to 512 and 1024 node, respectively. But on the larger node counts, we are seeing worse performance results by using synchronous progress threads. It is unexpected results that effective on smaller node count while negative impact on larger node count. In 2019, asynchronous progress threads were effective with growing node count up to 4,096. As same reasons as non async 2021 results, we are not sure and have difficulties to identify the root causes.

Table 4 show summarized performance improvement and degradation results which have comparison of original non asynchronous algorithm (Alg. 3 and 4) and asynchronous versions with asynchronous progress control (Alg. 3i and 4i). As described in section 6.1, We got up to 19% and 38% improvements for Alg. 3 to 3i and Alg. 4 to 4i, respectively in 2019 but we can see the positive effect of asynchronous progress control on fewer node count only.

Table 4: Performance improvement and degradation of introducing overlaps by MPI_Iallreduce and asynchronous progress control on Linux (measured in 2021). Note: Below 1.00 means negative effect.

Node	Alg. 3 non asyc to 3i with async	Alg. 4 non async to 4i with async
128	1.01	1.40
256	0.98	1.26
512	0.95	0.96
1024	0.87	0.82
2048	0.70	0.61
4096	0.53	0.43

6.3 Performance results on McKernel (measured in 2021)

For simplicity and little implementation cost, the modification for asynchronous progress control on McKernel is intended to use first two physical cores only while it is expected more improvement by allocating 4 unused physical cores to McKernel. 8 logical threads provided by first two physical cores handle asynchronous progress control in addition to IHK Linux for this time. It is almost equivalent to Fig. 7 or the method #2. All other measurements conditions are same as Linux in 2021.

Through Fig. 13a to 14b, which have case of non asynchronous progress control, we can see a little performance improvement on larger node count while almost same on lower node count. But we can clearly see error bars on McKernel are significantly smaller than that of Linux. This stable and less variant results are one of the benefits by McKernel. Similarly on Fig. 14a-14b, we can see much smaller error bars on McKernel, which are marked on top of "McKernel non async in 2021" and "McKernel async in 2021" bars.

Table 5 shows summarized performance improvement on McKernel without asynchronous progress control, compared with best Linux result without asynchronous progress control of 2019 and 2021. All other measurement results show better performance than Linux due to less noisy and low overhead nature of McKernel. For example, We can see up to 20% performance improvement on 2,048 node and 6% on 4,096 node for the Alg. 4. All performance numbers are listed on the Table 9 in the appendix as a reference.

Next, discussing asynchronous progress control on McKernel. Table 6 show summarized performance improvement and degradation results which have comparison of original non asynchronous algorithm (Alg. 3 and 4) and asynchronous versions with asynchronous progress control (Alg. 3i and 4i). We cannot see beneficial improvement on Alg.3i. With Alg. 4i, asynchronous progress threads give 2-3% performance improvement from 128 node to 1,024 node.

Table 5: Performance improvement on McKernel vs. best of Linux in 2019 and 2021. Note: Below 1.00 means negative effect.

Speedup	Alg. 1	Alg. 2	Alg. 3	Alg. 4
128	1.01	1.01	1.01	1.01
256	1.02	1.02	1.03	1.02
512	1.07	1.06	1.05	1.07
1024	1.10	1.09	1.07	1.08
2048	1.15	1.13	1.15	1.20
4096	1.01	1.06	1.07	1.06

7 RELATED WORK

Non-blocking collective operations were introduced in MPI several years ago [23]. However, only a few recent studies have explored thread placement strategies for asynchronous communication progress thread placement for non-blocking collective operations. Ohlmann et. al. presented Intel MPI's asynchronous progress control with an application case study demonstrating the benefits of progress thread placement [25].

Denis et. al. studied progress thread placement on many-core CPUs with a special focus on symmetric multithreading. Similarly to our findings, they also reported that running asynchronous progress on SMT threads of an application core can degrade performance due to cache effects in intranode communication [24].

Table 6: Performance improvement and degradation of introducing overlaps by MPI_Iallreduce and asynchronous progress control on McKernel (measured in 2021). Note: Below 1.00 means negative effect.

Node	Alg. 3 non asyc to 3i with async	Alg. 4 non async to 4i with async
128	1.00	1.02
256	0.98	1.03
512	0.95	1.02
1024	0.93	1.02
2048	0.91	0.92
4096	0.79	0.84

8 CONCLUSION AND FUTURE WORK

We have demonstrated that the asynchronous progress control is very effective with appropriate algorithm changes for CG solvers to overlap global synchronization and computation. We achieved 38% performance improvement on 4,096 node.

Although re-measurements on Linux in 2021 are not stable and worse than what measured in 2019 on larger node count more than 1,024, we are also able to confirm McKernel realized noise less and stable measurement in addition to performance improvement on GeoFEM. In most case, McKernel provided better performance especially on large node count even if compared with 2019 Linux results. Especially for 2,048 and 4,096 nodes, we got 20% and 6% improvements, respectively.

Furthermore, we have modified McKernel for asynchronous progress control and applied to real GeoFEM applications. From 128 node to 512 nodes, at least, we are able to see performance improvement by the combination of McKernel and asynchronous progress control. Though the effect is limited up to 1,024 node counts, we got at most 2-3% performance improvement from 128 node.

Work is ongoing and it is expected that we will run these performance experiments again after whole system tuning on Linux and McKernel to address the 2021 year's performance degradation than that of 2019. In the case of physical 4 cores allocation to IHK kernel for asynchronous progress control on McKernel will also be a future work. Furthermore, to investigate an effect on SMT threads more could lead to efficient use of asynchronous progress control for more MPI ranks per node or pure MPI parallelism situation without OpenMP threading.

ACKNOWLEDGMENTS

This work is supported by *JSPS Grant-in-Aid for Scientific Research (S)* (19H05662), and by *Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures* (jh20037-NAH, jh20041-NAH). Authors would like to thank Yoshio Sakaguchi, Nobuteru Mizoe and Toshiro Saiki from Fujitsu.

REFERENCES

[1] Intel's Asynchronous Progress Control: https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top/additional-supported-features/asynchronous-progress-control.html
[2] Ghysels, P. and W. Vanroose, Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm, Parallel Computing 40-7, 224-238, 2014
[3] Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator, ACM/IEEE Proceedings of SC 2003, 2003
[4] Joint Center for Advanced High Performance Computing (JCAHPC): http://jcahpc.jp/
[5] Nakajima, K., Gerofi, B., Ishikawa, Y., Horikoshi, M., Parallel Multigrid Methods on Manycore Clusters with IHK/McKernel, IEEE Proceedings of 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA 2019) in conjunction with SC19, Denver, CO, 2019
[6] Nakajima, K., Gerofi, B., Ishikawa, Y., Horikoshi, M., Efficient Parallel Multigrid Method on Intel Xeon Phi Clusters, ACM Proceedings of IXPUG HPC Asia 2021, 2021
[7] Taku Shimosawa, Balazs Gerofi, Masamichi Takagi, Gou Nakamura, Tomoki Shirasawa, Yuji Saeki, Masaaki Shimizu, Atsushi Hori and Yutaka Ishikawa, Interface for Heterogeneous Kernels: A Framework to Enable Hybrid OS Designs targeting High Performance Computing on Manycore Architectures, 21th Intl. Conference on High Performance Computing HiPC, 2014
[8] Balazs Gerofi, Akio Shimada, Atsushi Hori and Yutaka Ishikawa, Partially Separated Page Tables for Efficient Operating System Assisted Hierarchical Memory Management on Heterogeneous Architectures, 13th Intl. Symposium on Cluster, Cloud and Grid Computing CCGRID, 2013
[9] Balazs Gerofi, Masamichi Takagi, Atsushi Hori, Guo Nakamura, Tomoki Shirasawa and Yutaka Ishikawa, On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel, IEEE International Parallel and Distributed Processing Symposium IPDPS, 2016
[10] Gerofi, B, Riesen, R., Takagi, M., Boku, T., Nakajima, K., Ishikawa, Y., Wisniewski, R.W., Performance and Scalability of Lightweight Multi-kernel Based Operating Systems, IEEE Proceedings of IPDPS 2018, 116-125, 2018
[11] Saad, Y., Iterative Methods for Sparse Linear Systems (2nd Edition), SIAM, 2003
[12] Smith, B., P. Bjostad, and W. Gropp, Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge Press, 1996
[13] Demmel, J., Hoemmen, M., Mohiyuddin, M., Yelick, K., Avoiding communication in sparse matrix computations, IEEE Proceedings of 22nd International Symposium on Parallel and Distributed Processing, 2008 (IPDPS 2008), 2008
[14] Nakajima, K.and Ogita, T., Evaluations of Stability of Parallel Conjugate Gradient Methods based on Pipelined Algorithmss, IPSJ SIG Technical Reports 2018-HPC-167-26 (in Japanese), 2018
[15] Chronopoulos, A.T. and Gear, C.W., s-Step iterative methods for symmetric linear systems. Journal of Computational and Applied Mathematics. v25 i2. 153-168
[16] Hanawa, T., Nakajima, K., Ohshima, S., Hoshino, T., Ida, A., Performance Evaluation of Pipelined CG Method, IPSJ SIG Technical Report, Vol.2016-HPC-157, No.6, 2016 (in Japanese)
[17] Supercomputing Research Division, Information Technology Center, The University of Tokyo (ITC/U.Tokyo): http://www.cc.u-tokyo.ac.jp/
[18] Gropp, W., Update on libraries for blue waters, https://wgropp.cs.illinois.edu/bib/talks/tdata/2011/Stream-nbcg.pdf
[19] Horikoshi, M., Nakajima, K., Gerofi, B., Ishikawa, Y., Parallel Preconditioned Itertative Solvers on Oakforest-PACS, 28th Seminar of MEPA (Algorithms for Matrix / Eigenvalue Problems and their Applications, JSIAM (Japan Society for Industrial and Applied Mathematics)), 2019
[20] STREAM Benchmark: https://www.cs.virginia.edu/stream/
[21] Amit Ruhela, Hari Subramoni, Sourav Chakraborty, Mohammadreza Bayatpour, Pouya Kousha and Dhabaleswar K.(DK) Panda, Efficient design for MPI asynchronous progress without dedicated resources, Parallel Computing 85,13-26, 2019
[22] Horikoshi, M., Meadows L., Elken T., Sivakumar P., Mascarenhas, M., Erwin J., Durnov D., Sannikov A., Hanawa T., Boku, T., Scaling collectives on large clusters using Intel(R) architecture processors and fabric, ACM Proceedings of IXPUG HPC Asia 2018, 2018
[23] Hoefler T., Kambadur P., Graham R.L., Shipman G., Lumsdaine A. (2007) A Case for Standard Non-blocking Collective Operations. In Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2007. Lecture Notes in Computer Science, vol 4757. Springer, Berlin, Heidelberg
[24] Alexandre Denis, Julien Jaeger, Hugo Taboada, Progress Thread Placement for Overlapping MPI Non- Blocking Collectives using Simultaneous Multi-Threading. COLOC: 2nd workshop on data locality, in conjuction with Euro-Par 2018, 2018
[25] Sebastian Ohlmann, Fabio Baruffa, Markus Rampp, Overlapping communication and computation using the Intel MPI library's asynchronous progress control, IXPUG Meeting 2020, https://www.ixpug.org/resources/overlapping-communication-and-computation-using-the-intel-mpi-library-s-asynchronous-progress-control

A APPENDIX

A.1 Table of measurement data in 2019

Table 7: Results on Linux (measured in 2019). Alg. 3i and 4i used asynchronous progress threads while others did not use. From 128 to 4,096 nodes.

	Alg.1 [sec]	Alg.2 [sec]	Alg.3 [sec]	Alg.3i [sec]	Alg.4 [sec]	Alg.4i [sec]
128	1.39E+01	1.37E+01	1.42E+01	1.38E+01	1.42E+01	1.34E+01
256	9.21E+00	9.01E+00	9.34E+00	9.01E+00	9.39E+00	8.47E+00
512	6.37E+00	6.10E+00	6.24E+00	5.95E+00	6.40E+00	5.33E+00
1024	5.00E+00	4.69E+00	4.75E+00	4.49E+00	4.91E+00	3.93E+00
2048	4.51E+00	4.16E+00	4.22E+00	3.89E+0	4.58E+00	3.41E+00
4096	4.50E+00	4.03E+00	4.91E+00	4.14E+00	4.55E+00	3.29E+00

A.2 Tables of measurement data in 2021

Table 8: Results on Linux (measured in 2021). Alg. 3i and 4i used asynchronous progress threads while others did not use. From 128 to 4,096 nodes.

	Alg.1 [sec]	Alg.2 [sec]	Alg.3 [sec]	Alg.3i [sec]	Alg.4 [sec]	Alg.4i [sec]
128	1.39E+01	1.38E+01	1.43E+01	1.42E+01	1.43E+01	1.35E+01
256	9.15E+00	9.06E+00	9.35E+00	9.58E+00	9.47E+00	8.92E+00
512	6.49E+00	6.24E+00	6.43E+00	6.78E+00	6.56E+00	6.08E+00
1024	4.91E+00	4.68E+00	4.80E+00	5.50E+00	4.88E+00	4.79E+00
2048	5.09E+00	4.74E+00	4.82E+00	6.88E+00	5.05E+00	6.27E+00
4096	3.60E+00	3.34E+00	3.36E+00	6.33E+00	3.54E+00	5.89E+00

Table 9: Results on McKernel (measured in 2021). Alg. 3i and 4i used asynchronous progress threads while others did not use. From 128 to 4,096 nodes.

	Alg.1 [sec]	Alg.2 [sec]	Alg.3 [sec]	Alg.3i [sec]	Alg.4 [sec]	Alg.4i [sec]
128	1.38E+01	1.36E+01	1.40E+01	1.40E+01	1.40E+01	1.37E+01
256	9.01E+00	8.81E+00	9.10E+00	9.32E+00	9.20E+00	8.95E+00
512	5.96E+00	5.77E+00	5.96E+00	6.24E+00	5.99E+00	5.85E+00
1024	4.45E+00	4.29E+00	4.44E+00	4.78E+00	4.52E+00	4.43E+00
2048	3.91E+00	3.68E+00	3.66E+00	4.00E+00	3.82E+00	4.13E+00
4096	3.56E+00	3.16E+00	3.14E+00	3.99E+00	3.36E+00	3.99E+00

科学技術計算Ⅱ／コンピュータ科学特別講義Ⅱ／ハイブリッド分散並列コンピューティング：「並列有限要素法入門」（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 年度冬学期に実施した，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」[1]について紹介する。「新型コロナウイルス感染症」のため，前年度に引き続き，全ての講義を Zoom によるオンラインで実施した。

2014 年度までは，夏学期，冬学期に，科学技術計算 I・II／コンピュータ科学特別講義 I・II「科学技術計算プログラミング（有限要素法）」[2]を実施してきた。偏微分方程式の数値解法として，様々な科学技術分野のシミュレーションに使用されている有限要素法（Finite-Element Method，FEM）について，背景となる基礎的な理論から，実用的なプログラムの作成法まで，連立一次方程式解法などの周辺技術も含めて講義を実施し，プログラミングの実習を実施してきた。題材としては一次元及び三次元弾性静力学を扱い，プログラミング言語としては C 言語を使用していた。夏学期（I）と冬学期（II）に分けて，夏学期は有限要素法の理論とプログラミングの基礎，冬学期はその並列化についての講義・実習を行い，冬学期は東大情報基盤センターのスーパーコンピュータを使った実習を実施してきた。2011 年度までは T2K 東大を使用していたが，2012 年度からは Fujitsu PRIMEHPC FX10（Oakleaf-FX，2012 年 4 月運用開始），2016 年度からは「データ解析・シミュレーション融合スーパーコンピュータシステム（Reedbush）[3]」のうち，汎用 CPU（Intel Broadwell/EP）のみから構成される Reedbush-U（2016 年 7 月運用開始）を使用してプログラミング実習を実施してきた。更に，2019 年度冬学期からは，「大規模超並列スーパーコンピュータシステム（Oakbridge-CX，OBCX）[4]」を使用している。OBCX は 2019 年 7 月に運用を開始し，各々2 基の Intel Xeon Platinum 8280 から構成される計算ノードを 1,368 ノード搭載している。

2014 年度までの講義では，冬学期（Ⅱ）の履修は夏学期（Ⅰ）の履修を前提としていたが，昨今の大学の国際化に伴い，10 月に入学する留学生が増加しており，そのような条件を満たさない履修者が増えてきた。そこで 2015 年度からは，方針を変更し，両者をある程度独立した科目として履修できるよう：

- 夏学期（Ⅰ）：計算ノード内のマルチスレッド並列化に関する内容[5]
- 冬学期（Ⅱ）：分散並列環境における並列化に関する内容

[1] http://nkl.cc.u-tokyo.ac.jp/21w/
[2] http://nkl.cc.u-tokyo.ac.jp/14s/, http://nkl.cc.u-tokyo.ac.jp/14w/
[3] https://www.cc.u-tokyo.ac.jp/supercomputer/reedbush/service/
[4] https://www.cc.u-tokyo.ac.jp/supercomputer/obcx/service/
[5] http://nkl.cc.u-tokyo.ac.jp/21s/

のように実施することとした。留学生の受講，国際化に配慮して英語版教材のみを提供するとともに，**2017 年度からは英語で講義を実施している。**

表 1 に講義日程と内容を示す。上記のように，様々な分野で広く利用されている有限要素法を題材とし，一次元・三次元定常熱伝導方程式を扱った。一次元・三次元有限要素法，MPI（Message Passing Interface）による並列プログラミング，並列要素法の順番で講義・演習を実施した。また，ハイブリッド並列プログラミングモデルの重要性を考慮して，MPI＋OpenMP ハイブリッド並列プログラミングに関する講義・演習を実施した。MPI による並列有限要素法のプログラムの各プロセスに OpenMP を適用して並列化を実施した。

登録者は 35 名（うち 17 名が留学生），オンラインということもあり常時出席者は 25 名程度，単位を取得したのは 19 名（留学生 11 名）であった。2020 年度は登録者 36 名，単位取得者 12 名であったのに比べると，単位取得者の割合は増加している。

表 1：講義日程，内容

	Date	Time	Title
1	Oct.06(W)	0830-1015	Introduction, Introduction to FEM (1/2)
2	Oct.13(W)	0830-1015	Introduction to FEM (2/2), 1D/3D FEM (1/4)
3	Oct.20(W)	0830-1015	1D/3D FEM (2/4)
4	Oct.27(W)	0830-1015	1D/3D FEM (3/4)
5	**Nov.03 (W)**	**0900-1015**	**1D/3D FEM (4/4)**
6	Nov.10 (W)	0830-1015	Introduction to Parallel FEM, Login to OBCX, MPI (1/5)
7	Nov.17 (W)	0830-1015	MPI (2/5)
8	Nov.24 (W)	0830-1015	MPI (3/5)
9	Dec.01 (W)	0830-1015	Report S1, MPI (4/5)
10	Dec.08 (W)	0830-1015	MPI (5/5)
11	Dec.15 (W)	0830-1015	Report S2, Parallel FEM (1/4)
12	Dec.22 (W)	0830-1015	Parallel FEM (2/4)
13	Jan.05 (W)	0830-1015	Parallel FEM (3/4)
	Jan.12 (W)		**(No Class)**
14	**Jan.17 (M)**	**0830-1015**	**Parallel FEM (4/4), Hybrid OpenMP/MPI (2/3)**
15	Jan.19 (W)	0830-1015	Hybrid OpenMP/MPI (2/3)
16	Jan.26 (W)	0830-1015	Hybrid OpenMP/MPI (3/3)

教育活動報告：Computational Earthquake Engineering/計算地震工学E

市村 強 ・ 藤田 航平

東京大学 地震研究所

本稿では、2021年度A2セメスターに実施した、Computational Earthquake Engineering/計算地震工学E（工学系研究科・社会基盤学専攻大学院生、工学部・社会基盤学科学部3年生対象、月・木3限@オンライン講義、英語で実施）について紹介する。本講義では、地震工学分野における数値計算に焦点を当てた数値解析の基礎的な知識の習得を目的に、数値関数展開（有限差分法・有限要素法）、時間積分、数値積分、線形方程式の求解手法、及び、高性能計算(SIMD)に関する座学・演習を行った（表１参照）。本講義は例年、座学部分は工学部1号館、実習部分は情報基盤センターの大演習室においてそれぞれ対面で行い、実習における計算機環境として教育用計算機システム(ECCS)を用いてきたが、本年度においては昨年度に引き続き座学・実習ともにオンライン(Zoom)で行い、実習における計算機環境として情報基盤センターのOakbridge-CXを教育利用させて頂いた。評価は例年通りレポートの評点で行った。対面の演習において学生の画面を見つつ個別の議論が可能であった状況を再現できるよう、オンライン演習においてはZoomのブレイクアウトセッションを活用し、また、講義時間外の質問受付時間(Zoom)を数回設定するなど、個々の学生の理解に応じた個別の議論ができる体制を取った。数値計算・高性能計算に馴染みのない受講生でも自宅にいながら数値計算・高性能計算に少しでも興味を持ってもらえる機会を提供したと考えられ、将来のこの分野の人材育成につながると期待される。

表1:講義日程

回数	日付	形式	内容
1	11/29	座学1	導入
2	12/6	座学2	有限差分法・数値解の収束性
3	12/9	座学3	1次元有限要素法、動的・静的問題
4	12/13	実習1	実習環境説明
5	12/16	座学4	時間積分・クーラン条件
6	12/20	座学5	吸収境界条件・2次元有限要素法・ドロネー分割
7	12/23	座学6	数値積分、共役勾配法、前処理
8	12/27	実習2	レポート1（1次元有限要素法）
9	1/6	実習3	レポート2（2次元有限要素法）
10	1/13	実習4	レポート3（2次元有限要素法・ドロネー分割）
11	1/17	実習5	レポート4（SIMD演習）
12	1/20	実習6	レポート5（1次元動的有限要素法・吸収境界条件）うち、1次元動的有限要素法
13	1/24	実習7	レポート5（1次元動的有限要素法・吸収境界条件）うち、吸収境界条件

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・メニィコア並列プログラミング入門（Wisteria/BDEC-01（Odyssey，A64FX 搭載））（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，2022 年 3 月 9 日（水）にオンライン開催した第 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・メニィコア並列プログラミング入門（Wisteria/BDEC-01（Odyssey，A64FX 搭載））[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。

東京大学情報基盤センター（以下，本センター）では 2007 年からスーパーコンピュータを使用した「お試しアカウント付き並列プログラミング講習会」を開催しているが，新型コロナウイルス感染症対策のため，2020 年 4 月からは全ての講習会を Zoom 使用による「オンライン」講習会として実施している。「OpenMP によるマルチコア・メニィコア並列プログラミング入門」は，オンラインとなってから，第 131 回（2020 年 4 月 27 日），第 143 回（2020 年 11 月 2 日），第 151 回（2021 年 4 月 21 日）と 3 回，Oakbridge-CX（OBCX）[2]を使用して実施してきた。2019 年度までは 2 日にわけて実施していたものを，2020 年度からは 1 日にまとめて実施している。

近年マイクロプロセッサのマルチコア化が進み，様々なプログラミングモデルが提案されている。中でも OpenMP は指示行（ディレクティヴ）を挿入するだけで手軽に「並列化」ができるため，広く使用されており，様々な解説書も出版されている。メモリへの書き込みと参照が同時に起こるような「データ依存性（data dependency）」が生じる場合に並列化を実施するには，適切なデータの並べ替えを施す必要があるが，このような対策は OpenMP 向けの解説書でも詳しく取り上げられることは余り無い。第 151 回以前の講習会では，「有限体積法から導かれる疎行列を対象とした ICCG 法」を題材として，科学技術計算のためのマルチコアプログラミングにおいて重要なデータ配置，reordering などのアルゴリズムについての講義，スパコンを使用した実習を実施した。内容は筆者が本学大学院工学系研究科・情報理工学系研究科の講義として実施している「計算科学アライアンス特別講義 I」，「科学技術計算 I」，「スレッド並列コンピューティング」の内容に準拠している[3]。本来であれば 105 分×10 回程度の講義の内容を 1 日（正味 7.5 時間（450 分）程度）に圧縮しての内容となっているため，アンケートでも「1 日に詰め込む内容としては項目が多すぎる」が散見され，第 151 回のアンケートでは満足度が 3.73（満点 5.00）と比較的低い値となった。

2021 年度からは，2021 年 5 月 14 日に運用を開始した Wisteria/BDEC-01（Odyssey）[4]導入に合わせて内容を一新し，データ依存性に関連した内容を完全に削除したカリキュラムを策定した。大体 105 分の講義にして 4 回程度の内容であり，1 日，450 分程度で実施するには手頃な量

1 https://www.cc.u-tokyo.ac.jp/events/lectures/173/
2 https://www.cc.u-tokyo.ac.jp/supercomputer/obcx/service/
3 http://nkl.cc.u-tokyo.ac.jp/20s/
4 https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/service/

であった。また，Wisteria/BDEC-01（Odyssey）に搭載されている A64FX についても詳細な解説を実施した。内容もやや簡単になったため，満足度も高めとなったことから，今回もこの路線を踏襲した。

運用開始から 1 年近くが経過し，5 月の時点では性能にやや問題があった C コンパイラの性能等も改善された。また性能評価のための詳細プロファイラの使用法など新たな項目を追加するなど，内容としては前回と比較して充実したものとなった。

合計 14 名の参加申込みがあり，受講者は 10 名（大学・研究機関教職員：2，大学院生：2，学部生：4，企業：1）であった。アンケート結果（回収 9 名）について表 2 に示す。平均満足度は 4.44 と高めであった。

表 1　本講習会の概要

09：00-11：00	有限体積法
11：00-12：30	OpenMP 入門
13：30-16：00	オーダリング
16：00-17：30	OpenMP 並列化
17：30-18：00	質疑

表 2　アンケート集計結果

	評点	1	2	3	4	5
(a) 講習会時間	短い⇔長い		2	4	3	
(b) 講習会講義内容（プレゼン）	簡単⇔難			7	2	
(c) 配布資料内容	簡単⇔難			8	1	
(d) サンプルプログラム内容	簡単⇔難			7	2	
(e) 満足度（平均 4.44）	不満⇔満足				5	4

講義内容の詳細については，ウェブページから資料及び録画ビデオをダウンロードできるのでそちらを参照いただきたい（https://www.cc.u-tokyo.ac.jp/events/lectures/173/）。また，従来実施していたデータ依存性を含むケースについては，3 日にわけてプログラムを再編成し，10 月以降に実施する予定である。

第 174 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，2022 年 4 月 14 日（木）にオンライン開催した第 174 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充分に理解できるものであるが，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，2021 年 12 月 21 日に同様の内容で第 145 回[4]・第 171 回[5]講習会を実施し，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 2021 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」[6] 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[7]」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。

---

1 https://www.cc.u-tokyo.ac.jp/events/lectures/174/
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 https://www.cc.u-tokyo.ac.jp/events/lectures/171/
6 http://nkl.cc.u-tokyo.ac.jp/21w
7 http://nkl.cc.u-tokyo.ac.jp/FEM/01-FEMintro.pdf

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/174/）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 11 名，出席者は 7 名（学生：2 名，大学・研究機関：3 名，企業：2 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：4）。全体的な満足度の平均値は 5 点満点で 4.75 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。当日は，学内ネットワークのトラブルで 10 分ほどの中断を余儀なくされる時間帯があったが，急遽スマートフォンによるテザリングに切り替えた。受講者の皆さんにご迷惑をおかけしたこと，この場を借りてお詫び申しあげる次第である。

アンケートの自由記述欄のコメントを以下に示す：

- 丁寧に説明いただきありがとうございます。有限要素法の並列計算の講習会にも日程の都合がつけば出てみたいと思います。
- 今後もオンラインで開催していただけると参加しやすいです。
- サンプルプログラムの説明を詳細にしていただいて，FEM の理論とプログラムの対応が理解しやすかった。

第 175 回お試しアカウント付き並列プログラミング講習会

「スーパーコンピュータ超入門」

芝　隼人

東京大学情報基盤センター

2022 年 4 月 22 月に、第 175 回お試しアカウント付き並列プログラミング講習会「スーパーコンピューター超入門」（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））を開催したので、本稿にてその実施報告を行う。

東京大学情報基盤センターにおいて「超入門」と題した講習会は、2020 年 9 月 18 日（第 138 回）以来３回にわたっておよそ半年おきに実施しており、いずれも好評を博している。以前より Linux の簡単な操作・ログインなどの事柄から始めてスーパーコンピュータを用いるための基本を学ぶ講義及び実習を行ってきたが、今回は内容について少し見直して、機械学習をスーパーコンピュータで行うためのチュートリアルを追加した。アジェンダは次の通りであるが、後半の「より進んだ利用に向けて」は、今回は時間の不足で結果として取り扱うことができなかった。

13:00 - 14:00　ガイダンス・初めてのログイン（講義・実習）
14:15 - 15:45　サンプルプログラムによる並列計算超入門（講義・実習）
16:00 - 17:00　スパコンにおける機械学習実行超入門（講義・実習）
　　より進んだ利用に向けて（講義）

申込受付時にいずれの言語を使用するかのアンケートを行っているが、今回参加申込のあった 22 名での内訳は次の通りとなった。

・FORTRAN　8 名
・C/C++　4 名
・Python　10 名

通常通り、地球惑星科学や機械系、物理学など、科学技術計算分野への従来的なスパコン利用分野からの参加が多かったようである。一部の参加者は、これら分野での研究活動を行う中で機械学習などへの応用を考えていると見られ、このことが、Python の選択者の多さに現れていると見受けられる。

今回は、事前に日本語が得手でない、という参加者２名から個別に問い合わせをいただいた。今回は既に日本語での講習会の開設をアナウンスしていたため、講義については日本語で行うことを伝えた上で、ページ数を揃えた英語版の同内容の資料を用意しお渡しすることで対応した。演習に際しては日本語でのやりとりのほか、必要に応じて英語による説明を挟む形とした。

次の表１には、受講者からのアンケートの結果の集計を示す。当日の出席者数は 19 名であったが、当日参加者のうち 17 名から回答をいただくことができた。

表１　アンケート集計結果

	評点	1	2	3	4	5
(a) 講習会時間	短い⇔長い	1		15	1	
(b) 講習会講義内容	簡単⇔難	6	5	6		
(c) 配布資料内容	簡単⇔難	8	2	7		
(d) サンプルプログラム内容	簡単⇔難	8	2	7		
(e) 参加した満足度	不満⇔満足	1		3	2	11

同じシリーズの以前 3 回におけるアンケート結果と異なる傾向として、(b)(c)(d)の項目について内容が「簡単である」という側に回答が偏っている。かなりの割合の参加者は、本講習会のうち、スパコンでの機械学習実行についてのチュートリアルが追加されたことをきっかけに申し込んでおり、結果として以前の回と異なる層の参加者があったようである。今回のような参加者の構成であれば、Linux におけるコマンドの使用、公開鍵認証などの項目については不要であったかもしれない。

アンケートの自由記述欄には次のような意見が寄せられた。

- 機械学習のチュートリアルまであったのは有難かった。
- とても分かりやすく、自身の教育活動にも役に立ちそうです。
- 今後、有限要素法や線形計算の並列化についての講習が開催されれば参加したい。
- Oakbridge-CX にインストールされている Quantum Espresso の使用を検討しているので、ジョブ投入の仕方の説明が利用支援ポータルにあるとありがたい。
- スパコン初心者の私が知りたい内容で、解説もとても理解しやすかったです。講習中だけでなく講習後にも質問時間をいただき、疑問についても自身の経験をふまえて親身に教えていただきました。
- 複数のプログラム言語を使っていて煩雑になっていた。

また、Zoom によるオンライン開催については次のような感想が寄せられた。

- リアルタイムで演習を交えながら、スパコンに関する講義を受ける事ができました。
- オンラインなので参加しやすい。
- リアルタイムで演習を交えながら、スパコンに関する講義を受ける事ができました
- 移動時間が不要、旅費が不要で大変ありがたい（地方からの参加です)。大型ディスプレイで作業ができてやりやすい。
- Lecturer kindly took the time to answer questions and helped in English as well. Also, the slides translated into English were very helpful.
- ターミナル上のエラーを zoom の画面共有ですぐに確認して解決していただくことができて良かった。

今回は参加者による熱心な質問が随時行われたこと、また英語の説明を並行して行う箇所があったことなどにより、やや時間が押してしまった。担当者自身の講義能力は、経験の蓄積により前回までより部分的に改善したものの、最後のほうがやや駆け足となってしまった。説明がわかりにくくなるとともに不正確な言い回しが目立つ結果となり、担当者として少し反省を感じている。

当日の講習会は、前回に引き続きビデオ収録を行い、YouTube 上での公開を行った。講義資料および実習資料と併せて、当センターの第 175 回講習会ホームページ（ https://www.cc.u-tokyo.ac.jp/events/lectures/175/ ）からアクセスすることが可能である。特に第 3 部の講義はスパコン上での機械学習の環境構築および実行についての説明という点で価値あるものになったと考えている。今後に向けた改善点や要望などをお寄せいただければ幸いである。

次回の同種の講習会は、9〜10 月に実施する予定である。今回の反省内容を踏まえて時間割と内容を再検討したい。今回まで 4 回にわたって、PuTTY かターミナル系か、いずれかを選択してもらっていたが、前者の利用者が減少の傾向にあるようである。これは、Windows 10 以降でコマンドプロンプトから容易に OpenSSH が利用可能とな機能が実装されたことが関係していると思われる。この理由から、次回以降 PuTTY のサポートは廃止することにしたいと思う。また、センター利用者についても外国から新たに来られる方々など背景が多様化していることを踏まえて、次回の当該講習会は英語で試行する予定である。内容的には、前回までを踏まえたものとする予定であり、もちろん日本語を通常使われる方々の受講も歓迎したい。

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　原稿枚数には特に指定はありませんが、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 までご提出願います。
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)

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 (Odyssey/Aquarius)	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

※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）

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ニュース
Vol.24 No.3(2022.5)

目　　次