（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.23 No.5, 2021.9

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

スーパーコンピュータシステム 利用負担金表

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

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

※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。

※括弧（ ）内は付与するトークン量。実行したジョブのノード時間積または GPU 時間積と消費係数に応じてトークンが消費される。
付与したトークンは，利用期間内に全量が使用できることを保証するものではない。
トークンは利用期間内に限り有効とし，利用終了後に残量がある場合でも繰越や利用負担金の返還は行わない。
トークンの他のシステムへの移行については，「トークン移行におけるノード時間積の換算表」を参照。

※公募制度による申し込み，ノード固定の申し込みには審査を要する。

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

※GPU 専有申込の申込単位

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

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

コース		負担金額(税込)		ディスク容量	備考
		大学・公共機関等	企業		
パーソナルコース		申込 1 セット当り, 最大 3 セットまで 100,000 円 (8,640 ノード時間)		申込 1 セット当り /work 4TB 利用者当り /home 50GB	最大ノード数 256 ノード
グループコース	一般申込	申込 1 セット当り 100,000 円 (8,640 ノード時間)	申込 1 セット当り 120,000 円 (8,640 ノード時間)	グループ 1 セット当り /work 4TB 利用者当り /home 50GB	最大ノード数 256 ノード
	ノード固定	申込 1 セット当り 150,000 円 (8,640 ノード時間)	申込 1 セット当り 180,000 円 (8,640 ノード時間)		
トークン追加		8,400 円 (720 ノード時間)	10,000 円 (720 ノード時間)		
ディスク追加		6,480 円/(1TB*年)			1TB 単位で申込可 (/work のみ)

※トークン消費係数は 1.00 である。
※括弧内のノード時間は付与するトークン量。実行したジョブのノード時間積と消費係数に応じてトークンが消費される。
付与したトークンは, 利用期間内に全量が使用できることを保証するものではない。
トークンは利用期間内に限り有効とし, 利用終了後に残量がある場合でも繰越や利用負担金の返還は行わない。
トークンの他のシステムへの移行については,「トークン移行におけるノード時間積の換算表」を参照。
※ノード固定の申し込みには審査を要する。
※/home のディスク容量はパーソナルコースやグループコースに複数所属していても利用者当り 50GB 固定。

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

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

※トークン消費係数は 1.00 である。
※括弧内のノード時間は付与するトークン量。実行したジョブのノード時間積と消費係数に応じてトークンが消費される。
付与したトークンは, 利用期間内に全量が使用できることを保証するものではない。
トークンは利用期間内に限り有効とし, 利用終了後に残量がある場合でも繰越や利用負担金の返還は行わない。
トークンの他のシステムへの移行については,「トークン移行におけるノード時間積の換算表」を参照。
※/home のディスク容量はパーソナルコースやグループコースに複数所属していても利用者当り 50GB 固定。

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

コース		負担金額(税込)		ディスク容量	備考
		大学・公共機関等	企業		
パーソナルコース		申込 1 セット当り, 最大 2 セットまで 75,000 円 (8,640 ノード時間)		申込 1 セット当り /lustre 1TB 利用者当り /home 2GB	Reedbush-H 最大ノード数 32 ノード Reedbush-L 最大ノード数 16 ノード
グループコース	一般申込	申込 1 セット当り 75,000 円 (8,640 ノード時間)		グループ 1 セット当り /lustre 1TB 利用者当り /home 2GB	Reedbush-H 最大ノード数 32 ノード Reedbush-L 最大ノード数 16 ノード
	公募制度 Reedbush-H	申込 1 セット当り 公募制度 180,000 円 (21,600 ノード時間)	申込 1 セット当り 公募制度 216,000 円 (21,600 ノード時間)	グループ 1 セット当り /lustre 4TB 利用者当り /home 2GB	最大ノード数 32 ノード
	公募制度・ ノード固定 Reedbush-L	申込 1 セット当り 公募制度 300,000 円 ノード固定 450,000 円 (34,560 ノード時間)	申込 1 セット当り 公募制度 360,000 円 ノード固定 540,000 円 (34,560 ノード時間)	グループ 1 セット当り /lustre 4TB 利用者当り /home 2GB	最大ノード数 16 ノード
トークン追加		6,300 円 (720 ノード時間)	7,500 円 (720 ノード時間)		
ディスク追加		6,480 円／(1TB*年)			1TB 単位で申込可 (/lustre のみ)

※トークン消費係数は下記の通りである。

Reedbush-H: 2.50, Reedbush-L: 4.00

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

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

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

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

※公募制度・ノード固定の申し込みには審査を要する。

※/home のディスク容量はパーソナルコースやグループコースに複数所属していても利用者当り 2GB 固定。

トークン移行におけるノード時間積の換算表

移行元 ＼ 移行先	Reedbush-H/L システム	Oakforest-PACS システム	Oakbridge-CX システム	Wisteria/BDEC-01 システム
Reedbush-H/L システム	ー	1.5	0.75	1.2
Oakforest-PACS システム	0.6	ー	0.5	0.8
Oakbridge-CX システム	1.3	2	ー	1.6
Wisteria/BDEC-01 システム	0.8	1.2	0.6	ー

移行先に追加されるトークン量（ノード時間）＝移行トークン量×係数

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

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

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） 正式運用期間	制限時間（経過時間） 試験運用期間	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	30 分	28	○	○
short-o	1 ～ 72 (3,456)	8 時間	4 時間	28	○	○
(regular-o)						
small-o	1 ～ 144 (6,912)	48 時間	12 時間	28	○	○
medium-o	145 ～ 576 (27,648)	〃	〃	〃	○	○
large-o	577 ～ 1,152 (55,296)	〃	〃	〃	○	○
x-large-o	1,153 ～ 2,304 (110,592)	24 時間	6 時間	〃	○	○
priority-o	1 ～ 288 (13,824)	48 時間	12 時間	28	○	○
challenge-o	1 ～ 7,680 (368,640)	24 時間	－	28	★	★
(interactive-o) ※4						
interactive-o_n1	1 (48)	30 分	30 分	28	○	○
interactive-o_n12	2 ～ 12 (576)	10 分	10 分	〃	○	○
prepost	1 (56)	6 時間	3 時間	340	○	○
prepost1_n1 ～ prepost4_n1	1 (56)	1～6 時間	1～3 時間	340	○	○
prepost1_n4	1 ～ 4 (224)	1～6 時間	1～3 時間	340	○	○
prepost1_n8	1 ～ 8 (448)	1～6 時間	1～3 時間	340	○	○

★ 審査による課題選定の上，月1回の一定期間のみ利用可能(原則として月末処理日前日の朝～翌日朝)
※1 キューの指定(〝#PJM -L“rscgrp=キュー名”〞)は，regular-o，debug-o，short-o を小文字で指定する
regular-o キューはノード数の指定(〝#PJM -L“node=ノード数”〞)でノード数別のキューに投入される
※2 トークンの消費係数は 1 ノード当り 1.00。ただし priority-o は優先利用ノード群のためトークン消費係数は 1.50
※3 1ノード当りの利用者が利用可能なメモリー容量
※4 インタラクティブジョブの起動は次のとおり（トークン消費なし）

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

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

キュー名※1	ノード数・GPU 数※2（最大 GPU 数）	制限時間（経過時間） 正式運用期間	制限時間（経過時間） 試験運用期間	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	30 分	448	○	○	○	○
short-a	1 ～ 2 ノード (16)	2 時間	1 時間	448	○	○	○	○
(regular-a)								
small-a	1 ～ 2 ノード (16)	48 時間	12 時間	448	○	○	○	○
medium-a	3 ～ 4 ノード (32)	〃	〃	〃	○	○	○	○
large-a	5 ～ 8 ノード (64)	24 時間	6 時間	〃	○	○	○	○
share-debug	1, 2, 4 GPU	30 分	30 分	56	○	○	○	○
share-short	1, 2, 4 GPU	2 時間	1 時間	56	○	○	○	○
(share)								
share-1	1 GPU	48 時間	12 時間	56	○	○	○	○
share-2	2 GPU	〃	〃	〃	○	○	○	○
share-4	4 GPU	24 時間	6 時間	〃	○	○	○	○
challenge-a	1 ～ 39 ノード (312)	24 時間	－	448	★	★	★	★
任意	1 ノード (8)	任意 ※4	－	448	×	×	○	○
interactive-a ※5	1 ノード (8)	10 分	10 分	56	○	○	○	○
share-interactive	1 GPU	〃	〃	〃	○	○	○	○

★ 審査による課題選定の上，月1回の一定期間のみ利用可能(原則として月末処理日前日の朝～翌日朝)
※1 キューの指定(〝#PJM -L“rscgrp=キュー名”〞)は，regular-a，debug-a，short-a を小文字で指定する
regular-a キューはノード数の指定(〝#PJM -L“node=ノード数”〞)でノード数別のキューに投入される
※2 トークンの消費係数は 1GPU 当り 3.00
※3 1ノード当りの利用者が利用可能なメモリー容量
※4 申込ノード数の合計以内ならば，キュー名・制限時間(原則 48 時間以内)は相談の上，任意に設定可能
※5 インタラクティブジョブの起動は次のとおり（トークン消費なし）

pjsub --interact -g グループ名 -L“rscgrp=interactive-a,node=1”

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

キュー名※1	ノード数※2（最大コア数）			制限時間（経過時間）	メモリー容量（GB）※3	パーソナルコース	グループコース	
							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"

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

キュー名※1	ノード数※2（最大コア数）			制限時間（経過時間）	メモリー容量（GB）※3	パーソナルコース	グループコース
debug-cache	1 ～	128	(8704)	30 分	82	○	○
debug-flat	1 ～	128	(8704)	30 分	96	○	○
(regular-cache)							
small-cache	1 ～	128	(8704)	48 時間	82	○	○
medium-cache	129 ～	512	(34816)	〃	〃	○	○
large-cache	513 ～	1024	(69632)	〃	〃	○	○
x-large-cache	1025 ～	2048	(139264)	24 時間	〃	○	○
(regular-flat)							
small-flat	1 ～	128	(8704)	48 時間	96	○	○
medium-flat	129 ～	512	(34816)	〃	〃	○	○
large-flat	513 ～	1024	(69632)	〃	〃	○	○
x-large-flat	1025 ～	2048	(139264)	24 時間	〃	○	○
challenge	1 ～	8208	(558144)	24 時間	82 / 96	★	★
(interactive-cache) ※4							
interactive_n1-cache		1	(68)	2 時間	82	○	○
interactive_n16-cache	2 ～	16	(1088)	10 分	〃	○	○
(interactive-flat) ※4							
interactive_n1-flat		1	(68)	2 時間	96	○	○
interactive_n16-flat	2 ～	16	(1088)	10 分	〃	○	○
prepost		1	(28)	6 時間	222	○	○

★ 審査による課題選定の上，月1回の一定期間のみ利用可能(原則として月末処理日前日の朝～翌日朝)
※1 キューの指定("#PJM -L "rscgrp=キュー名" ")は，regular-cache/flat，debug-cache/flat を小文字で指定する
regular-cache/flat キューはノード数の指定("#PJM -L "node=ノード数" ")でノード数別のキューに投入される
※2 トークンの消費係数は 1.00
※3 1ノード当りの利用者が利用可能なメモリー容量
※4 インタラクティブジョブの起動は， pjsub --interact -g グループ名 -L "rscgrp=キュー名,node=ノード数"
(キュー名は interactive-cache/flat，トークン消費なし)

Reedbush スーパーコンピュータシステム（Reedbush-H） ジョブクラス制限値（2018 年 9 月 27 日）

キュー名※1	ノード数※2（最大 GPU 数）	制限時間（経過時間）	メモリー容量（GB）※3	パーソナルコース	グループコース	
					一般申込	公募制度
h-debug	1 ～ 4 (8)	30 分	244	○	○	○
h-short	1 ～ 4 (8)	2 時間	244	○	○	○
(h-regular)						
h-small	1 ～ 4 (8)	48 時間	244	○	○	○
h-medium	5 ～ 8 (16)	〃	〃	○	○	○
h-large	9 ～ 16 (32)	〃	〃	○	○	○
h-x-large	17 ～ 32 (64)	24 時間	〃	○	○	○
h-challenge	1 ～ 120 (240)	24 時間	244	★	★	★
(h-interactive) ※4						
h-interactive_1	1 (2)	2 時間	244	○	○	○
h-interactive_2	2 (4)	30 分	〃	○	○	○
(h-regular-low) ※5						
h-small-low	1 ～ 4 (8)	12 時間	244	▲	▲	▲
h-medium-low	5 ～ 8 (16)	〃	〃	×	▲	▲
h-large-low	9 ～ 16 (32)	〃	〃	×	▲	▲
h-x-large-low	17 ～ 32 (64)	6 時間	〃	×	▲	▲

▲　パーソナルコースは 最大 1 ノード，グループコースは申込ノード数の 4 分の 1 まで実行可（公募制度による申し込みの場合は申込ノード数まで実行可）
★　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（〝#PBS -q キュー名〟）は，h-short，h-regular，h-debug を小文字で指定する
h-regular キューはノード数の指定（〝#PBS -l select=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 2.50
※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）
qsub -I -q h-interactive -l select=ノード数 -l walltime=XX:XX -W group_list=グループ名
※5 非優先ジョブクラス（低プライオリティキュー）は，トークンの追加申込が締め切られ，ジョブ実行に必要なトークン残量が不足した場合のみ実行可

Reedbush スーパーコンピュータシステム（Reedbush-L） ジョブクラス制限値（2020 年 4 月 1 日）

キュー名※1	ノード数※2（最大 GPU 数）	制限時間（経過時間）	メモリー容量（GB）※3	パーソナルコース	グループコース		
					一般申込	公募制度	ノード固定
l-debug	1 ～ 4 (16)	30 分	244	○	○	○	○
(l-regular)							
l-small	1 ～ 4 (16)	168 時間	244	○	○	○	○
l-medium	5 ～ 8 (32)	〃	〃	○	○	○	○
l-large	9 ～ 16 (64)	〃	〃	○	○	○	○
任意	申込数	任意 ※4	244	×	×	×	○
(l-interactive) ※5							
l-interactive_1	1 (4)	24 時間	244	○	○	○	○
l-interactive_2	2 (8)	6 時間	〃	○	○	○	○
l-interactive_4	3 ～ 4 (16)	1 時間	〃	○	○	○	○
(l-regular-low) ※6							
l-small-low	1 ～ 4 (16)	12 時間	244	▲	▲	▲	▲
l-medium-low	5 ～ 8 (32)	〃	〃	×	▲	▲	▲
l-large-low	9 ～ 16 (64)	〃	〃	×	▲	▲	▲

▲　パーソナルコースは 最大 1 ノード，グループコースは申込ノード数の 4 分の 1 まで実行可（公募制度・ノード固定による申し込みの場合は申込ノード数まで実行可）
※1 キューの指定（〝#PBS -q キュー名〟）は，l-debug，l-regular を小文字で指定する
l-regular キューはノード数の指定（〝#PBS -l select=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 4.00
※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 168 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（トークン消費あり）
qsub -I -q l-interactive -l select=ノード数 -l walltime=XX:XX -W group_list=グループ名
※6 非優先ジョブクラス（低プライオリティキュー）は，トークンの追加申込が締め切られ，ジョブ実行に必要なトークン残量が不足した場合のみ実行可

センターから

サービス休止等のお知らせ

2021 年 9 月下旬からの計算機サービス予定は以下のとおりです。

Wisteria/BDEC-01 スーパーコンピュータシステム【 8 月 2 日～ 正式運用開始 】

○ Wisteria/BDEC-01 スーパーコンピュータシステム正式運用開始のお知らせ

2021 年 8 月 2 日（月）10:00 より正式運用を開始しました。正式運用については，
本誌別記事「Wisteria/BDEC-01 スーパーコンピュータシステム正式運用に関するお知らせ(再掲)」をご覧ください。

○ 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
9 月 22 日（水）	9:00 ～ 22:00 までサービス休止	月末処理
10 月 29 日（金）	9:00 ～ 22:00 までサービス休止	月末処理
11 月 26 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
10 月 28 日（木）8:30 ～ 17:30 まで 11 月 25 日（木）8:30 ～ 17:30 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
9 月 24 日（金）～ 9 月 27 日（月）	9/24 9:00 ～ 9/27 17:00 までサービス休止	月末処理、空調機メンテナンス、特別高圧受変電設備点検・二次変電設備点検
10 月 29 日（金）	9:00 ～ 20:00 までサービス休止	月末処理
11 月 26 日（金）	9:00 ～ 20:00 までサービス休止	月末処理

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

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

大規模 HPC チャレンジ 実施期間
9 月 22 日（水）8:30 ～ 17:30 まで 10 月 28 日（木）8:30 ～ 17:30 まで 11 月 25 日（木）8:30 ～ 17:30 まで

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

Oakforest-PACS スーパーコンピュータシステム

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

日　付	利用者サービス	センター内作業
9 月 24 日（金）～ 9 月 27 日（月）	9/24 9:00 ～ 9/27 17:00 までサービス休止	月末処理、空調機メンテナンス、特別高圧受変電設備点検・二次変電設備点検
10 月 27 日（水）	9:00 ～ 22:00 までサービス休止	月末処理
11 月 25 日（木）	9:00 ～ 22:00 までサービス休止	月末処理

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

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

大規模 HPC チャレンジ 実施期間
9 月 22 日（水）8:30 ～ 17:30 まで 10 月 26 日（火）8:30 ～ 17:30 まで 11 月 24 日（水）8:30 ～ 17:30 まで

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

Reedbush スーパーコンピュータシステム

○ Reedbush スーパーコンピュータシステム（Reedbush-H，Reedbush-L）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
9 月 24 日（金）～ 9 月 27 日（月）	9/24 9:00 ～ 9/27 17:00 までサービス休止	月末処理・二次変電設備定期点検
10 月 22 日（金）～ 10 月 25 日（月）	10/22 9:00 ～ 10/25 17:00 までサービス休止	月末処理、空調機メンテナンス、特別高圧受変電施設定期点検のためサービス休止
11 月 30 日（火）	9:00 ～ サービス終了	サービス終了

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

○ Reedbush スーパーコンピュータシステム（Reedbush-H）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
9 月 21 日（火）9:00 ～ 9 月 22 日（水）9:00 まで 10 月 21 日（木）9:00 ～ 10 月 22 日（金）9:00 まで 11 月 29 日（月）9:00 ～ 11 月 30 日（火）9:00 まで

・上記期間中， Reedbush-H の h-debug, h-short，h-regular，h-interactive 及び講義用キューのサービスを休止します。
Reedbush-L，ログインノード 及び prepost キューは利用可能です。

【注意事項】

- サービス休止等の計画は原稿作成時の予定です。やむを得ずサービスを変更したり，休止したりする場合がありますので，最新の情報は login 時のメッセージ及びスーパーコンピューティング部門の Web ページの運用スケジュール（https://www.cc.u-tokyo.ac.jp/supercomputer/schedule.php）をご確認ください。
- 平日の 9:00～17:00 以外，休日（土・日・祝日等）は，システム障害等でサービスが停止した場合，運転を継続できない場合があります。その場合は，その時間をもってサービスを中止しますのでご了承ください。

＊ Wisteria/BDEC-01，Oakbridge-CX 及び Oakforest-PACS において，新型コロナウイルス感染症拡大防止に配慮し，一部条件を変更して実施しております。詳細は Web ページ(https://www.cc.u-tokyo.ac.jp/guide/hpc/)をご覧ください。なお Reedbush-H においては引き続き中止しております。

システム変更等のお知らせ

(2021.7.1 － 2021.8.31 変更)

1. ハードウェア

1.1 Wisteria/BDEC-01 スーパーコンピュータシステム … なし

1.2 Oakbridge-CX スーパーコンピュータシステム … なし

1.3 Oakforest-PACS スーパーコンピュータシステム … なし

1.4 Reedbush スーパーコンピュータシステム (Reedbush-H/L) … なし

2. ソフトウェア

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

- Wisteria-O (Odyssey)

GROMACS	2021.2	(2021.07.02)
GROMACS (倍精度)	2021.2	(2021.07.29)
GCC Toolset	10	(2021.07.29)

- Wisteria-A (Aquarius)

FTW	3.3.9	(2021.07.02)
HDF5	1.12.0	(2021.07.02)
NetCDF	4.8.0	(2021.07.02)
GNU Scientific Library	2.6	(2021.07.02)
LAMMPS (GPU パッケージ)	29Oct2020	(2021.07.29)
GCC Toolset	10	(2021.07.29)
Red Hat Enterprise Linux (OS)	8.3	(2021.08.24)

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)

gfarm	2.7.19	(2021.07.30)
gfarm2fs	1.2.15	(2021.07.30)

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。

2.3 Red Hat Enterprise Linux 7, CentOS 7 (Oakforest-PACS)

gfarm	2.7.19	(2021.07.28)
gfarm2fs	1.2.15	(2021.07.28)

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。

2.4 Red Hat Enterprise Linux 7 (Reedbush-H/L)

chainer	7.2.0	(2021.07.30)

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ドキュメント閲覧より
利用手引書または各資料をご覧ください。

3. その他

3.1 Oakbridge-CX における pjsub コマンドの--at オプションの有効化について

- 無効となっていた pjsub コマンドの--at オプションを有効に変更しました。(2021.07.30)

3.2 Oakforest-PACS における McKernel 対応について

- McKernel のジョブのノード数上限を 1024 → 2048 ノードへ変更しました。(2021.08.04)

3.3 Wisteria/BDEC-01(Odyssey/Aquarius)における Environment Modules の設定変更について

- システム名(odyssey/aquarius)を選択(module load <システム名>)すると、デフォルト設定をロードするように Environment Modules 構成を変更しました。デフォルトで設定されるコンパイラは以下となります。(2021.07.02)
  Odyssey： 富士通コンパイラと富士通 MPI を設定(ver 1.2.31)
  Aquarius： GCC ver 8.3.1 を設定
- 計算ノードでのみ利用可能なパッケージ・ライブラリについて、ログインノードで module help を表示できるように設定しました。(2021.07.02)
- バッチジョブ・インタラクティブジョブを実行した場合、module load <システム名> を自動実行するように設定しました。(2021.07.30)
  (注： Aquarius をご利用の場合)
  本変更に伴い、Aquarius でバッチジョブ・インタラクティブジョブを実行した場合、デフォルトで設定されるコンパイラが GCC となります。Intel 開発環境をご利用の場合は、module purge を実行後に Intel 開発環境をロードする必要がありますのでご注意ください。
  ロードモジュールに関する詳細は Wisteria/BDEC-01 システム利用手引書をご覧下さい。
  利用手引書は利用支援ポータル (https://wisteria-www.cc.u-tokyo.ac.jp/) の「ドキュメント閲覧」より入手することが可能です。

  「ドキュメント閲覧」
  →「Wisteria/BDEC-01 利用手引書」
  →「Wisteria/BDEC-01 システム利用手引書」
  → 4.1. 環境設定

3.4 Wisteria/BDEC-01 スーパーコンピュータシステムの運用について

Wisteria/BDEC-01 スーパーコンピュータシステムは 2021 年 8 月 2 日 10:00 より正式運用を開始しました。本誌関連記事「Wisteria/BDEC-01 スーパーコンピュータシステム正式運用に関するお知らせ (再掲)」、または、当センターWeb ページ(https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/service/)をご覧ください。

3.5 Wisteria/BDEC-01(Odyssey) regular-o キュー及び priority-o キューにおけるデフォルトのジョブ割当方法の変更について (2021.08.26)

Wisteria/BDEC-01 Odyssey の regular-o キューと priority-o キューにつきましては、正式運用開始後、デフォルトのジョブ割当方法として、離散割り当てモード(隣接しないノードを割り当てる場合がある)が設定されていました。8/26 のサービス再開後より、regular-o キュー及び priority-o キューのデフォルトのジョブ割当方法についてメッシュモード(隣接するノードを単位としてジョブを割り当てる)に変更しました。(その他のキューについては、引き続き離散割り当てモードがデフォルトとなります)

regular-o キュー及び priority-o キューにおきまして、8/26 以降、離散割り当てモードでジョブを実行されたい場合は、pjsub コマンドの -L オプションに「node=<shape>:noncont」と「:noncont」を加える

必要がありますのでご注意ください。オプションの詳細は、Wisteria/BDEC-01 システム利用手引書をご覧ください。

「ドキュメント閲覧」
→「Wisteria/BDEC-01 利用手引書」
→「Wisteria/BDEC-01 システム利用手引書」
→ 5.3.2.ジョブ資源オプション
(参考) 5.4.5.5.複数ノード MPI 並列ジョブ用スクリプト
→「製品マニュアル」
→「FUJITSU Software Technical Computing Suite」
→「ジョブ運用ソフトウェア エンドユーザ向けガイド」
→ 1.6.3 ノード単位での割り当て

3.6 Reedbush-H/L サービス終了について

Reedbush-H/L スーパーコンピュータシステムは 2021 年 11 月末をもってシステムを停止し、すべてのサービスを終了致します。詳細について決まり次第 Web ページ、メール、スーパーコンピューティングニュースにて順次ご連絡致します。

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

- サービス終了後のスーパーコンピュータのご利用につきましては Wisteria/BDEC-01、Oakbridge-CX をご検討ください。
- 一般利用にて Reedbush-H/L をご利用の方は「トークン移行」を行うことが可能です。Wisteria/BDEC-01、Oakbridge-CX への移行をご検討の利用者様につきましては「トークン移行」も併せてご参考ください。「トークン移行」についての詳細は Web ページ(https://www.cc.u-tokyo.ac.jp/guide/application/transfer_token.php)をご参照ください。

1. はじめに

Wisteria/BDEC-01 スーパーコンピュータシステムは、2021 年 5 月 14 日 10:00 ～ 7 月 29 日 9:00 まで試験運用を実施し、8 月 2 日 10:00 より正式サービスを開始しました。基本的なサービス内容については試験運用期間中と変更はありませんが、ジョブクラス制限値（経過時間）の拡大、ノード固定サービスの開始などの変更を行っています。本稿では、試験運用期間と正式サービス後の違いについて記載しています。また、最新の情報は、当センター Web ページ[1] にて広報していますのでご確認ください。

2. 試験運用と正式サービスの違い

2021 年 8 月 2 日（月）10:00 より正式サービスを開始しました。主な変更点は以下の通りです。

・トークンの再設定

利用申込書またはトークン移行申込書に記載されている内容でトークン量が再設定されております（例えば、利用期限を年度末としていた場合、申込セット数に応じて、8 ヵ月分のトークン量が設定されます）。

また、これまで利用した実績値も 0（ゼロ）に設定しました。

・利用負担金

試験運用期間は無料でご利用いただけましたが、正式サービス開始以降のご利用には利用負担金が必要です。利用負担金額は、申込者の所属先により、「大学等・公共機関等」、「企業」の 2 区分となっています。

詳細は、当センター Web ページ[2] または、2021 年 5 月号記事「利用負担金の改正について」をご覧ください。

・ジョブクラス制限値

以下のとおりキューの制限（経過）時間が変更になっております。

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

キュー名	ノード数(最大コア数)			制限(経過)時間		メモリー容量 (GiB)
				正式サービス	試験運用期間	
debug-o	1 ～	144	(6,912)	30 分	30 分	28
short-o	1 ～	72	(3,456)	**8 時間**	**4 時間**	28
(regular-o)						
small-o	1 ～	144	(6,912)	**48 時間**	**12 時間**	28
medium-o	145 ～	576	(27,648)	**48 時間**	**12 時間**	28
large-o	577 ～	1,152	(55,296)	**48 時間**	**12 時間**	28
x-large-o	1,153 ～	2,304	(110,592)	**24 時間**	**6 時間**	28
priority-o	1 ～	288	(13,824)	**48 時間**	**12 時間**	28
(interactive-o)						
interactive-o_n1		1	(48)	30 分	30 分	28
interactive-o_n12	2 ～	12	(576)	10 分	10 分	28
prepost		1	(56)	**6 時間**	**3 時間**	340
prepost1_n1 ～ prepost4_n1		1	(56)	1～**6 時間**	1～**3 時間**	340
prepost1_n4	1 ～	4	(224)	1～**6 時間**	1～**3 時間**	340
prepost1_n8	1 ～	8	(448)	1～**6 時間**	1～**3 時間**	340

[1] https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/service/

[2] https://www.cc.u-tokyo.ac.jp/guide/application/index.php#charge

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

キュー名	ノード数・GPU 数 (最大 GPU 数)	制限(経過)時間		メモリー容量 (GiB)
		正式サービス	試験運用期間	
debug-a	1 ノード (8)	30 分	30 分	448
short-a	1 ～ 2 ノード (16)	**2 時間**	**1 時間**	448
(regular-a)				
small-a	1 ～ 2 ノード (16)	**48 時間**	**12 時間**	448
medium-a	3 ～ 4 ノード (32)	**48 時間**	**12 時間**	448
large-a	5 ～ 8 ノード (64)	**24 時間**	**6 時間**	448
share-debug	1, 2, 4 GPU	30 分	30 分	56
share-short	1, 2, 4 GPU	**2 時間**	**1 時間**	56
(share)				
share-1	1 GPU	**48 時間**	**12 時間**	56
share-2	2 GPU	**48 時間**	**12 時間**	56
share-4	4 GPU	**24 時間**	**6 時間**	56
interactive-a	1 ノード (8)	10 分	10 分	56
share-interactive	1 GPU	10 分	10 分	56

3. 参考記事のご案内

Wisteria/BDEC-01 に関する以下の詳細な情報が、2021 年 7 月号記事[3]にございます。是非ご参照ください。

・「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」の特徴
・Wisteria/BDEC-01 を構成する NVIDIA GPU 及びネットワーク製品の概要
・第 3 世代インテル® Xeon® スケーラブル・プロセッサーのご紹介

4. 問い合わせ先

最新の情報は、本センター Web ページ[1] にて広報しています。利用方法の詳細については Wisteria/BDEC-01 利用支援ポータル[4] の「システム利用手引書」をご覧ください。メールによる問い合わせについては、事前に本センター Web ページ[1] にて情報がないかご確認の上、問い合わせの内容により、以下のどちらかのメールアドレスまでご連絡ください。

・利用申込に関する内容については、受付窓口 uketsuke@cc.u-tokyo.ac.jp までお願いいたします。
・利用相談、プログラム相談等に関する内容については、「問い合わせ」Web ページ[5] から詳細を記載した上で、ご連絡をお願いいたします。

[3] https://www.cc.u-tokyo.ac.jp/public/news.php#VOL23
[4] https://wisteria-www.cc.u-tokyo.ac.jp/
[5] https://www.cc.u-tokyo.ac.jp/supports/contact/#soudan

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（臨時）

Wisteria/BDEC-01、Oakbridge-CX、Oakforest-PACS スーパーコンピュータシステムでは、「大規模 HPC チャレンジ」を実施しています。「大規模 HPC チャレンジ」は、スーパーコンピュータシステムがもつ最大規模のノード数を、最大 8 時間・1 研究グループで計算資源の占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拡大防止に配慮し、当面の間特別スケジュール・条件のもと実施いたします。

1. 提供資源

以下のスーパーコンピュータシステムの計算ノードを最大 8 時間専有利用することができます。
ただし活動制限指針レベルに応じて実施時間を延ばす可能性があります。

【Wisteria/BDEC-01】

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 ノード又はデータ・学習ノード群（Aquarius）36 ノード或いはその両方

* 各月 Odyssey 1 件、Aquarius 1 件の最大 2 件まで受入可能、ただし 1 グループで両方利用することも可能

【Oakbridge-CX】

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）

【Oakforest-PACS】

Oakforest-PACS スーパーコンピュータシステムの flat モード（4,200 ノード）又は cache モード（3,200 ノード）、或いはその両方

* 各月 flat モード 1 件、cache モード 1 件の最大 2 件まで受入可能、ただし 1 グループで flat モード、cache モード両方利用することも可能

＊ 大規模 HPC チャレンジ実施前後の運用切り替えは実施しない

2. 利用案内

・1 ヶ月に 1 回、原則として月末処理前日 9:00～17:00 までの最大 8 時間、提供資源を専有利用することが可能です。
（システム障害等が発生しない場合、flat モードは最大 4,346 ノード，cache モードは最大 3,346 ノードまで利用できる可能性がございます。）

・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、およ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。ただし Wisteria/BDEC-01、Oakforest-PACS は最大 2 件まで受入可能です。

・**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または最先端共同 HPC 基盤施設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。

・センターまたは最先端共同 HPC 基盤施設の主催、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。

3. 実施日程

2021 年度の今後の「大規模 HPC チャレンジ」実施日程は表 1～3 のとおりです。

表 1. 2021 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
2021 年 12 月 16 日(木) 9:00 ～ 17:00 2022 年 1 月 27 日(木) 9:00 ～ 17:00 2022 年 2 月 24 日(木) 9:00 ～ 17:00 2022 年 3 月 30 日(水) 9:00 ～ 17:00	2021 年 10 月 25 日（月） 17:00【締切】	11 月上旬	11 月中旬

表 2. 2021 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
2021 年 12 月 16 日(木) 9:00 ～ 17:00 2022 年 1 月 27 日(木) 9:00 ～ 17:00 2022 年 2 月 24 日(木) 9:00 ～ 17:00 2022 年 3 月 30 日(水) 9:00 ～ 17:00	2021 年 10 月 25 日（月） 17:00【締切】	11 月上旬	11 月中旬

表 3. 2021 年度 Oakforest-PACS 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
2021 年 12 月 14 日(火) 9:00 ～ 17:00 2022 年 1 月 25 日(火) 9:00 ～ 17:00 2022 年 2 月 21 日(月) 9:00 ～ 17:00 2022 年 3 月 30 日(水) 9:00 ～ 17:00	2021 年 10 月 25 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。

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

4. 研究課題

「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（Oakforest-PACS については筑波大学計算科学研究センターの教職員も含む）、および、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、およ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）

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（Oakforest-PACS については筑波大学計算科学研究センターの教職員も含む）、および、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８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、電子メールにてお送りください。なお、書類のご提出はPDF形式にてお願いいたします。

申込書類に必要な項目は以下の通りです。

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

8. 申込書提出先・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[あっとまーく]cc.u-tokyo.ac.jp
（［あっとまーく］を @（半角）に変換後、ご送信ください）
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[あっとまーく]cc.u-tokyo.ac.jp
（［あっとまーく］を @（半角）に変換後、ご送信ください）
東京大学　情報システム部　情報戦略課　研究支援チーム

萌芽共同研究公募課題 AI for HPC：Society 5.0 実現へ向けた人工知能・データ科学による計算科学の高度化（試行）2021 年度採択課題

中 島 研 吾
東京大学情報基盤センター

スーパーコンピュータを駆使したシミュレーションによる計算科学が「第 3 の科学」と呼ばれるようになって久しいものがあります。昨今は、スーパーコンピュータは様々な分野に使われるようになり、データ科学、機械学習、AI などの分野での利用が特に盛んになっています。シミュレーション（Simulation）、データ（Data）、学習（Learning）の融合（S+D+L）は、特にシミュレーションによる計算科学に新しい道を開き、Society 5.0 実現への貢献とともに、ポストムーア時代に向けた新しい計算パラダイムとしても期待されます。東京大学情報基盤センター（以下、本センター）では、2015 年頃からこのような状況を想定し，「S+D+L」融合を実現するプラットフォームとして「『計算・データ・学習』融合スーパーコンピュータシステム」（通称「BDEC（Big Data & Extreme Computing）システム」）構築を目指して，様々な研究開発を進めてきました。2021 年 5 月に運用を開始した「Wisteria/BDEC-01[1] [2]」は BDEC システムの第 1 号機です。同時に Wisteria/BDEC-01 上で（S+D+L）を実現するための革新的ソフトウェア基盤「h3-Open-BDEC[3]」の研究開発も進めています。

2021 年度に実施する萌芽共同研究公募課題「AI for HPC：Society 5.0 実現へ向けた人工知能・データ科学による計算科学の高度化（試行）」（以下「本共同研究」）では、（S+D+L）融合実現、データ科学、機械学習、人工知能による計算科学の高度化を目指す提案を募集しました。本公募課題は，提案者と本センター教員との共同研究として実施し，次年度以降の学際大規模情報基盤共同利用・共同研究拠点（JHPCN）に研究課題として応募することを目標としています。

本公募課題の詳細につきましては HP[4]をご覧ください。

本公募課題は 2020 年度から試行的に開始しましたが，2021 年度は Oakforest-PACS，Oakbridge-CX に加えて Wisteria/BDEC-01 を対象として，公募を実施しました。2 件の応募があり，本センター教員に外部委員を加えた審査委員会による審査の結果，次頁に示す 2 件が採択となり，2021 年 8 月から研究を開始しています。

[1] https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/service/
[2] https://www.hpcwire.jp/archives/42271
[3] https://www.youtube.com/watch?v=jX51NF2LniE
[4] https://www.cc.u-tokyo.ac.jp/guide/exploratory/

課題名	超巨大アンサンブル計算と機械学習の協調による地球科学シミュレーションの不確実性定量化
氏名（所属）	澤田 洋平（東京大学 工学系研究科附属総合研究機構）
利用システム	Wsiteria/BDEC-01（Odyssey，Aquarius）

地球科学におけるシミュレーションには様々な不確実性が存在し、その定量化と最小化が重要である。モ デル選択、モデルパラメータ選択、初期条件の誤差、境界条件の誤差といった不確実性をもたらす要因の うち、どの要因が結果を大きく左右するのかを明らかにした上で、地球観測を用いて不確実性を効率よく 最小化する”不確実性定量化(Uncertainty Quantification)”手法の確立が求められている。本研究では、 Wisteria/BDEC-01 の CPU ノードと GPU ノード双方の性能を最大限に引き出して水文気象学における 最大計算規模のシミュレーションに対する不確実性定量化問題に挑戦する。大量のパラメータ等の設定 の組み合わせによる超巨大アンサンブルシミュレーションと、その結果を模擬する機械学習の協調により、効率よく大規模な不確実性定量化問題を解く。これにより単一のモデルを用いた決定論的な予測か ら、データに基づいた多様なモデル群による確率的な現象予測へと、地球科学におけるモデリングの新し いパラダイムの創出を目指す。

課題名	数値シミュレーションと機械学習との融合による東京湾の赤潮予測
氏名（所属）	菊地 淳(理化学研究所 環境資源科学研究センター)
利用システム	Wsiteria/BDEC-01（Aquarius），Oakbridge-CX

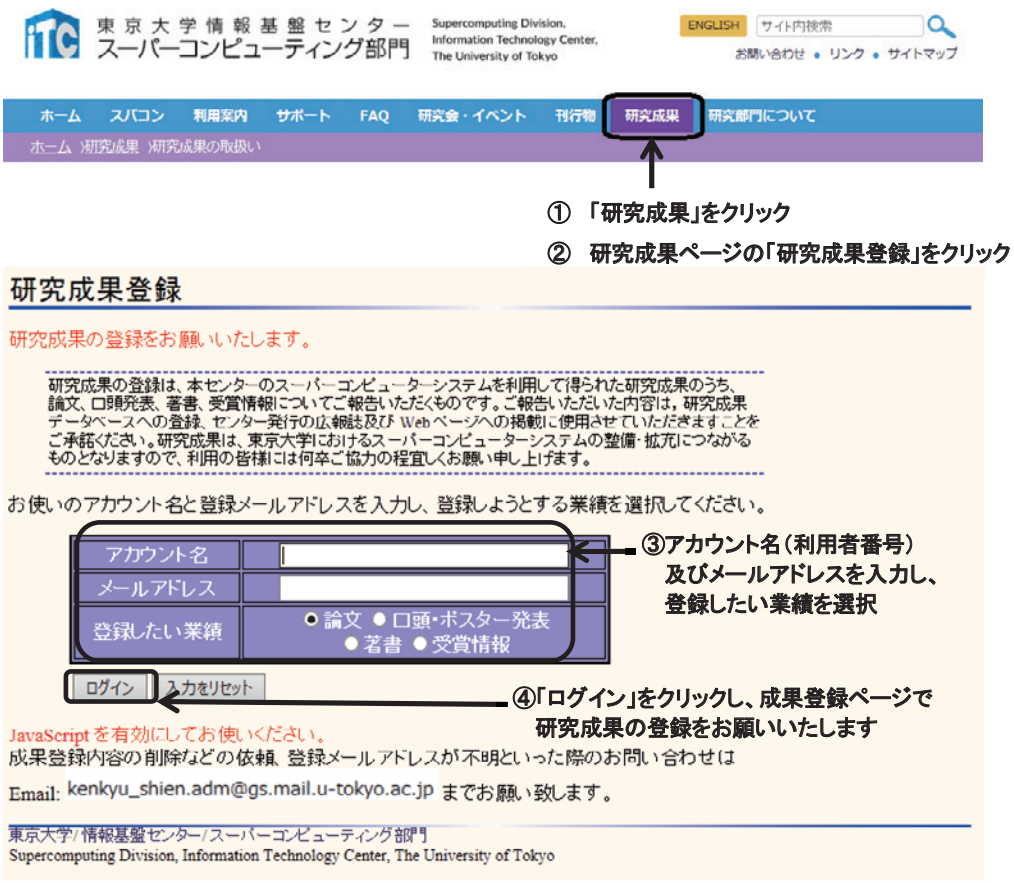
赤潮は, 海中のプランクトンの異常発生により引き起こされ, 水産業への被害をもたらす. 被害を未然 に防止するために, その発生時期・規模を予測することが重要な課題となっている. 機械学習を用いた 赤潮発生予測の手法が近年活発に研究されており, 伝統的な統計学に基づく手法より精度の良い予測が 可能になっているが, 未だ十分とはいえない. 本研究では, 東京湾を対象として, 領域海洋モデルによる数値シミュレーションと現業の気象予報・ 解析データ, そして現場観測データを機械学習により統合し, 高精度な赤潮予測を実現する. 数値シミュ レーションにおいて, 東京湾内のダイナミクスを解像できる高い空間解像度と黒潮や潮汐による外洋水 との混合を十分に表現できるだけの広い計算領域を設定し, 長期間の計算を行うために大型計算機を用 いる. また, 赤潮のみならず水圏生態系について、将来における空間分布を予測するためのビッグデー タを用いた機械学習システムの構築準備を行う。

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。

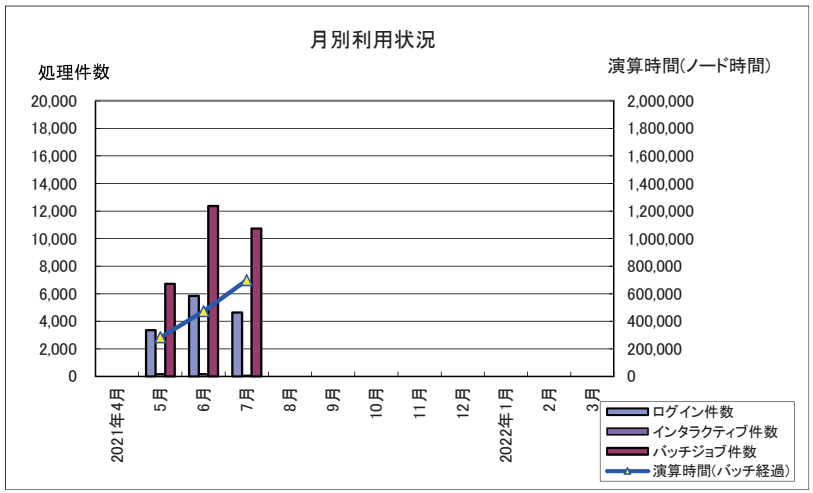


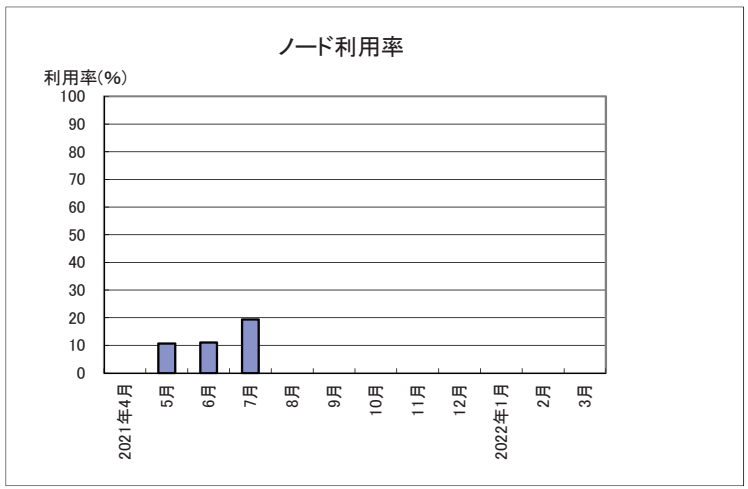
6・7月のジョブ統計

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
合計			13,831	217	438	29,815	61,837			3,766	236	81	1,454,566		

・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



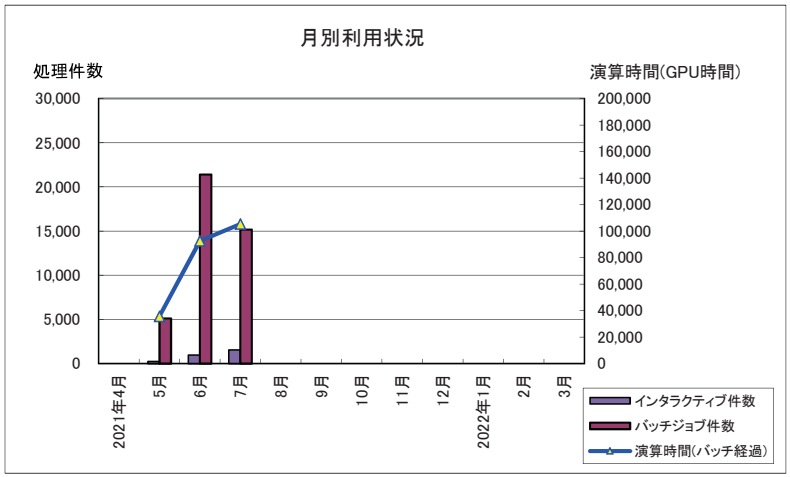


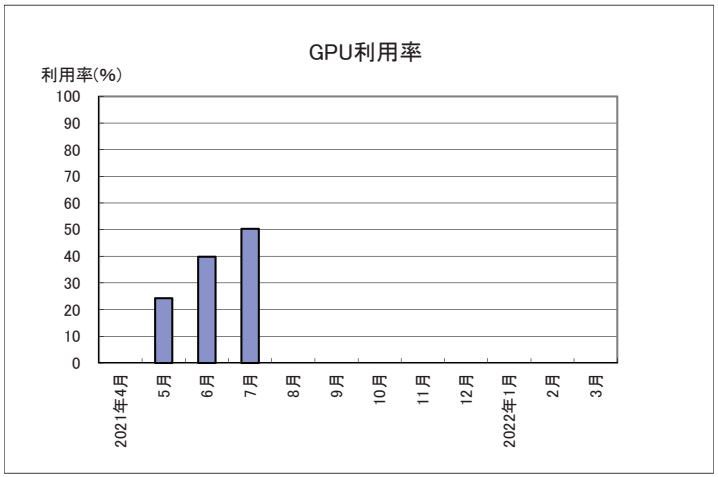
2. Wisteria/BDEC-01スーパーコンピュータシステム（Aquarius）ジョブ処理状況 （RedHat Enterprise Linux 8）

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
合計	2,745	41,644	2,934	233,527		

・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

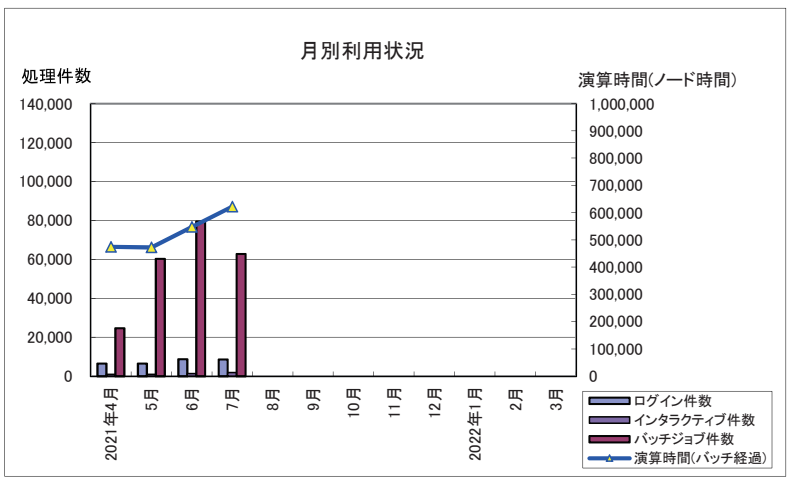


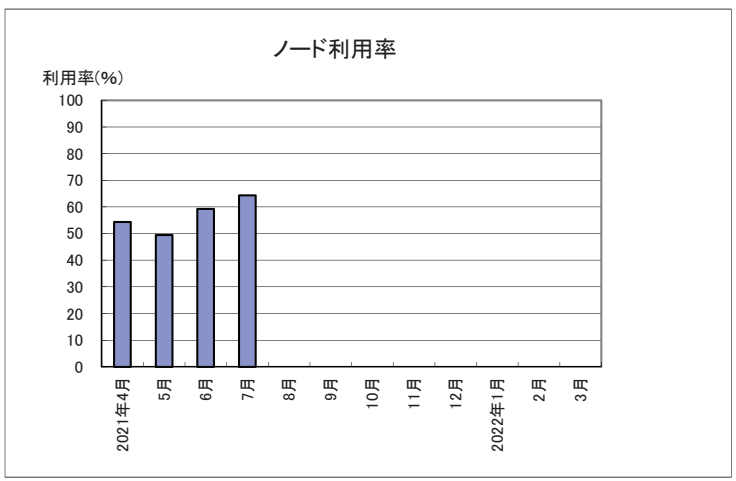


3. Oakbridge-CX スーパーコンピュータシステムジョブ処理状況（Red Hat Enterprise Linux 7、CentOS 7）

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	1,032	190	6,441	2	929	24,568	27,989	913	1,339,482	1,243.13	0	385	474,091	738.6	54.3
5月	1,058	188	6,429	0	984	60,244	40,008	874	1,292,853	8,997.03	0	995	472,465	671.4	49.4
6月	1,086	250	8,690	43	1,365	79,386	48,885	993	1,357,480	2,863.30	1	1,095	547,426	805.1	59.2
7月	1,131	252	8,567	42	1,860	62,712	57,917	1,017	1,430,169	16,811.53	2	675	622,153	874.0	64.3
2020年7月	1,378	347	18,263	118	1,760	80,506	53,532	1,040	1,319,119	5,753.21	519	1,179	668,610	957.8	70.4
8月	1,371	295	8,780	67	1,450	63,090	27,508	1,078	1,411,638	9,448.79	136	949	498,413	938.4	69.0
9月	1,535	299	11,804	48	1,152	80,846	39,017	1,130	1,539,134	3,073.18	119	740	630,436	1,030.7	75.8
10月	1,250	254	13,972	102	1,841	87,992	72,162	1,160	1,700,805	6,384.71	519	829	732,192	1,052.0	77.4
11月	1,241	268	13,169	13	2,707	47,307	62,082	1,172	1,878,875	15,583.41	17	980	716,567	1,086.7	79.9
12月	1,214	241	11,452	56	1,475	75,087	43,511	1,191	1,872,689	9,074.40	253	860	768,743	1,127.9	82.9
2021年1月	1,105	215	10,238	49	1,601	64,441	53,307	1,021	1,891,719	3,322.22	253	982	754,704	1,110.5	81.7
2月	1,110	161	7,326	117	2,215	40,336	28,469	1,049	1,963,710	2,864.16	398	893	591,306	941.2	69.2
3月	1,080	170	7,803	67	1,373	63,001	35,810	1,115	1,502,680	2,660.82	387	671	591,775	854.0	62.8
合計			114,671	606	18,952	749,010	536,665			82,327	2,085	10,054	7,400,271		

・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
・2020年7月分は合計に含まない
・ノード利用数： インタラクティブおよびバッチジョブの経過時間を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

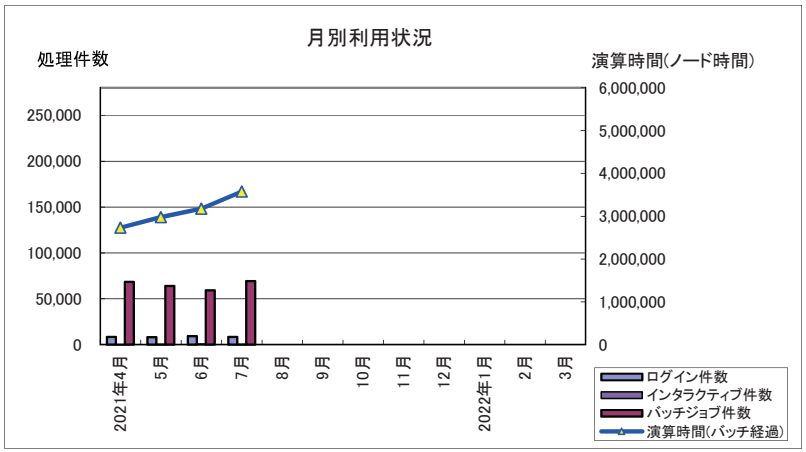


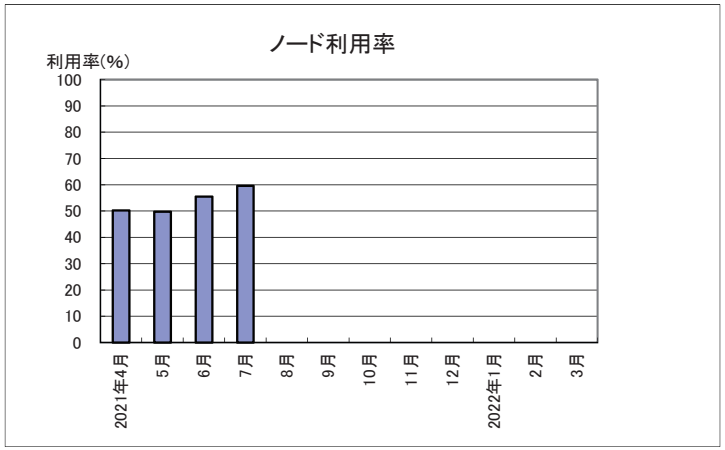


4. Oakforest-PACSスーパーコンピュータシステムジョブ処理状況 （RedHat Enterprise Linux 7、CentOS 7）

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	2,731,333	4,117.0	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
7月	1,622	410	12,928	343	327	52,585	64,608	2,164	5,996,751	2,262.33	492	163	4,121,263	5,909.8	72.0
8月	1,617	401	12,139	315	618	83,908	70,019	2,374	6,332,050	11,191.93	685	514	4,726,977	6,611.0	80.5
9月	1,732	420	9,811	293	795	52,222	56,925	2,441	6,299,074	3,237.18	667	1,390	4,125,115	6,447.3	78.5
10月	1,766	522	12,023	191	464	52,938	82,796	2,552	6,492,672	6,460.30	398	710	4,437,924	6,071.9	74.0
11月	1,822	461	13,149	251	256	65,413	74,790	2,535	6,618,475	3,297.48	516	125	4,468,068	6,400.5	78.0
12月	1,789	471	13,407	419	375	99,699	66,020	2,648	6,634,466	6,273.23	630	149	4,837,811	6,633.6	80.8
2021年1月	1,804	475	12,636	495	556	210,638	84,101	2,687	6,512,605	3,174.92	787	233	5,112,137	6,993.7	85.2
2月	1,786	431	11,384	314	282	79,481	78,257	2,787	6,581,603	5,791.25	546	197	4,733,131	7,166.3	87.3
3月	1,727	472	12,986	384	302	271,840	102,642	2,760	6,629,043	6,329.17	706	174	5,484,594	7,523.7	91.7
合計			131,885	3,449	5,043	1,176,678	867,799			55,460	6,546	3,930	50,388,805		

・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
・2020年7月分は合計に含まない
・ノード利用数： インタラクティブおよびバッチジョブの経過時間を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

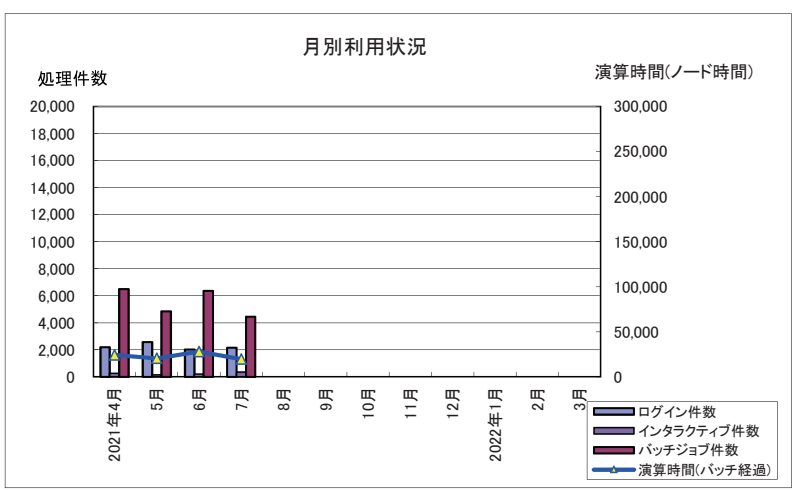


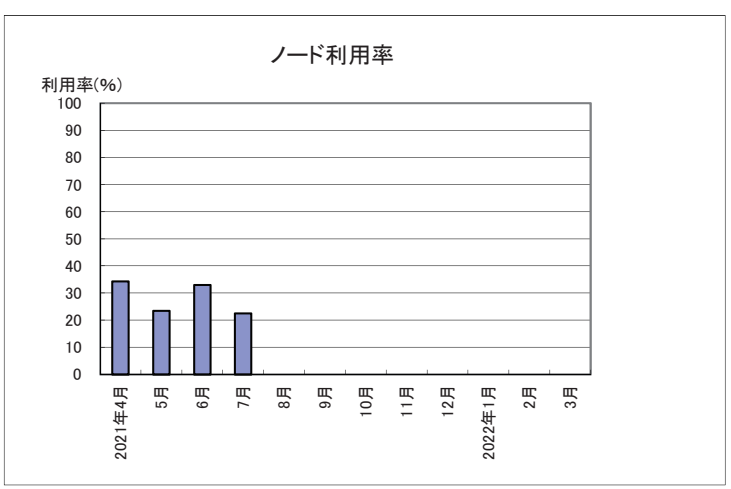


5. Reedbushスーパーコンピュータシステム（Reedbush-H）ジョブ処理状況 （RedHat Enterprise Linux 7）

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）			平均ﾉｰﾄﾞ利用数（ノード）	ﾉｰﾄﾞ利用率（%）
			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4月	986	127	2,183	19	253	6,489	3,140	146	192,825	12.31	2	284	24,386	41.1	34.3
5月	644	116	2,571	0	149	4,839	3,857	134	203,491	13.35	0	179	20,465	28.0	23.4
6月	592	102	2,022	0	200	6,355	3,567	135	200,432	2.60	0	200	27,992	39.6	33.0
7月	587	111	2,149	2	348	4,452	3,564	142	210,761	4.95	3	285	19,598	27.0	22.5
2020年7月	1,069	188	4,736	8	1,053	7,234	7,653	148	299,291	6.06	5	1,024	41,261	57.4	47.8
8月	999	171	4,032	0	897	5,165	6,535	157	298,649	7	0	1,213	34,188	72.2	60.2
9月	986	161	4,364	0	742	6,403	8,054	159	321,098	12.18	0	1,077	26,655	43.3	36.1
10月	1,127	214	5,277	0	869	6,962	7,133	168	327,145	18.74	0	973	47,424	72.9	60.8
11月	1,070	188	4,943	0	717	13,502	6,705	168	337,922	7.48	0	675	28,402	40.8	34.0
12月	1,094	236	5,252	0	1,545	16,439	8,449	180	349,615	6.64	0	1,979	47,463	67.1	55.9
2021年1月	1,112	195	4,585	0	1,010	12,940	7,523	181	356,486	7.35	0	1,173	44,632	62.2	51.8
2月	1,057	188	3,758	0	809	12,800	5,831	188	354,254	16.00	0	954	56,532	86.5	72.1
3月	1,056	176	4,140	0	776	7,345	5,273	189	316,348	13.20	0	920	73,742	102.4	85.4
合計			45,276	21	8,315	103,691	69,631			122	5	9,912	451,479		

・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
・2020年7月分は合計に含まない
・ノード利用数： インタラクティブおよびバッチジョブの経過時間を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





6. Reedbushスーパーコンピュータシステム（Reedbush-L）ジョブ処理状況 （RedHat Enterprise Linux 7）

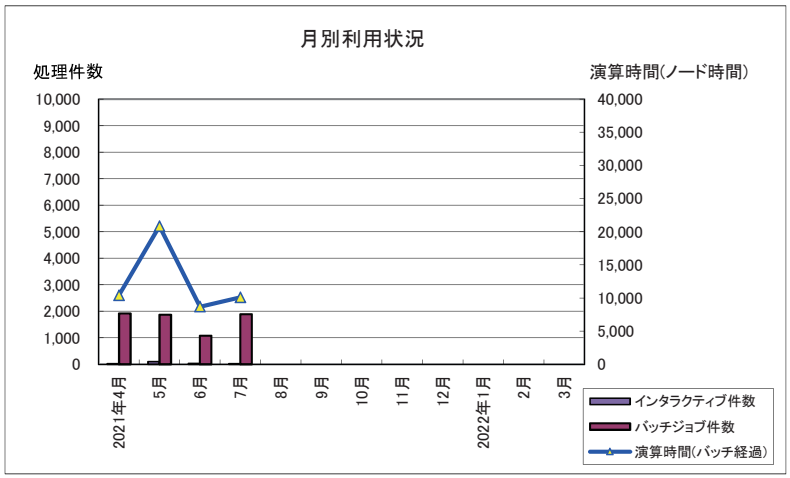
年月	処理件数		演算時間 [ノード時間](経過時間)		平均ﾉｰﾄﾞ利用数（ノード）	ﾉｰﾄﾞ利用率（%）
	インタラクティブジョブ	バッチジョブ	インタラクティブジョブ	バッチジョブ		
2021年4月	19	1,913	3	10,413	17.4	32.8
5月	95	1,867	45	20,833	28.3	53.5
6月	34	1,075	70	8,689	12.3	23.2
7月	25	1,887	44	10,081	13.8	26.0
7月	21	2,883	27	15,123	20.6	38.1
8月	58	1,551	64	10,739	25.6	47.4
9月	57	1,757	49	20,902	32.7	60.6
10月	156	2,427	53	16,400	24.8	45.9
11月	56	1,695	25	24,476	34.4	63.7
12月	105	2,734	106	21,208	28.9	53.6
2021年1月	100	3,171	46	26,404	35.9	66.7
2月	56	2,657	211	28,072	41.1	74.7
3月	16	727	24	34,927	47.9	87.2
合計	777	23,461	740	233,144		

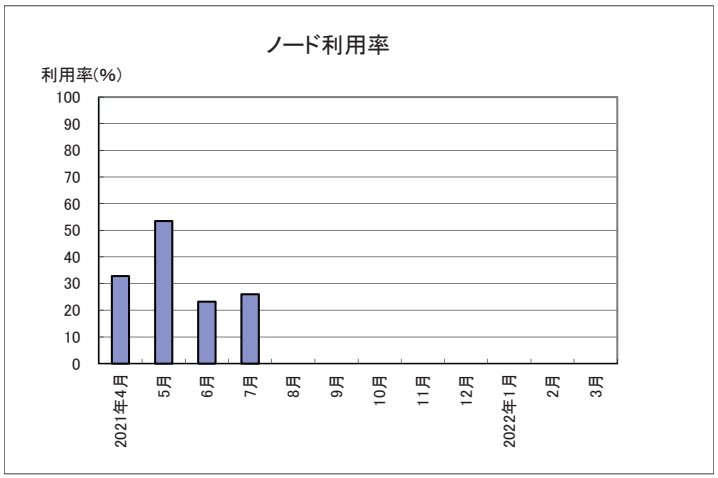
・登録者数、実利用者数、ログイン件数、接続時間、ファイル使用量、ログイン（実CPU）はReedbush-Uと共通。

・2020年7月分は合計に含まない

・ノード利用数： インタラクティブおよびバッチジョブの経過時間を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





機械学習ポテンシャルを用いた金属–固体酸化物の界面構造とイオン伝導特性の大規模解析

清水康司
東京大学大学院工学系研究科マテリアル工学専攻

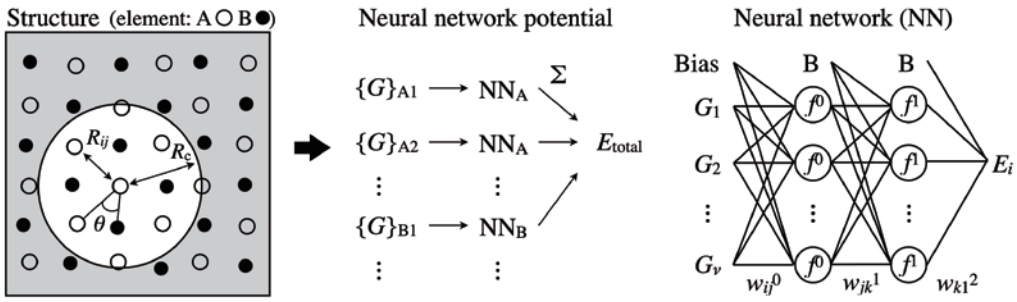
1. はじめに

金（Au）電極/非晶質リン酸リチウム（Li_3PO_4）固体電解質/リチウム（Li）電極を積層した構造は新規の不揮発性メモリとしての応用が検討されている[1, 2]。この素子では，電極間への電圧の印加によって可逆的に異なる 2 値の電圧状態を制御できるため、 これらの情報を記憶することでメモリとして動作する。また，Au 電極の代わりにニッケル（Ni）電極を用いた素子では 3 値のメモリとなる[3]。これらの新規不揮発性メモリでは，電圧印加にともなう固体電解質内部あるいは固体電解質と電極の界面近傍における Li イオン分布の変化が重要であると指摘されており，動作原理の理解には原子スケールでのシミュレーションによる解析が必要である。

密度汎関数理論（Density functional theory; DFT）に基づく第一原理計算は，実験データに合わせて決めるようなパラメータを用いることなく材料の様々な性質をうまく予測することが多くの場合に可能である。例えば，Li_3PO_4 中の Li 拡散の活性化障壁[4, 5]や，Ni 電極の開回路電圧の Li 吸着量依存性[3]は実験結果とよく一致している。しかし，構造探索や原子・イオンの挙動の解析，非晶質材料や界面を含む構造の解析などの目的に対しては計算量が膨大になるため，上記の素子の動作原理理解に向けて十分な解析を幅広く実施することは現在の計算機性能を考慮しても困難である。以上のような背景から，材料の性質予測において，第一原理計算と同等の精度でなおかつ計算コストの低い手法の開発が近年進められており，その代表的なものに機械学習ポテンシャルがある。最近開発が進められている機械学習ポテンシャルとして著名なものに，high-dimensional neural network potential（NNP）[6]，Gaussian approximation potential（GAP）[7]，spectral neighbor analysis potential（SNAP）[8]などがある。筆者らは，その機械学習ポテンシャルの中でも，ニューラルネットワークを用いた原子間ポテンシャルに着目し，金属と固体電解質の界面系への適用を進めてきた。特に，金属として Au を用いた場合には，界面近傍での Li の堆積量に応じて Au との合金化が起こる可能性がある。そこで著者らは，Au-Li 合金系の安定相の探索および合金化の動的過程の解析に適用可能な NNP を作成した。また，その過程で，NNP の作成を効率化する方法を提案した。本稿では，Au-Li 合金系の NNP の作成方法と，作成した NNP での解析結果を中心に報告する。

2. 計算方法

本研究では，高次元ニューラルネットワークポテンシャル（以降単に NNP と記す）[6]を用いて，原子間ポテンシャルを作成した。本手法ではまず，ある構造（原子配置）が与えられたときに，各原子の周囲の原子環境を対称性関数（symmetry functions; SFs）で数値化する。ここでは，Behler と Parrinello の方法[6]に倣い，SFs として動径成分（2 体，G^2）によるものと角度成分（3 体，G^3）を含むものを使用した。具体的な SFs の表式を以下に示す。



第 1 図: ニューラルネットワークポテンシャル（NNP）の概略図。

ある構造における各原子の対称性関数を計算し（左図）, NN の入力とする。構成原子全てについて NN の出力 E_i を足し合わせたものを, その構造の全エネルギー E_{total} とする（中央図）。右図は隠れ層 2 層の場合の NN の構造をあらわしている。

$$G_i^2 = \sum_{j\neq i}^{\text{all}} e^{-\eta\left(R_{ij}-R_s\right)^2} f_c\left(R_{ij}\right),$$

$$G_i^3 = 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} \left(1+\lambda\cos\theta_{ijk}\right)^{\zeta} e^{-\eta\left(R_{ij}^2+R_{ik}^2\right)} f_c\left(R_{ij}\right) f_c\left(R_{ik}\right).$$

ここで, G_i の添字 i は原子の番号をあらわしており，R_{ij}（R_{ik}）は原子 i と j （k）の距離，θ_{ijk} は原子 i とその周囲の原子 j と k で構成される角度をあらわしている。f_c はカットオフ関数と呼ばれ，ここでは以下の表式を用いた。

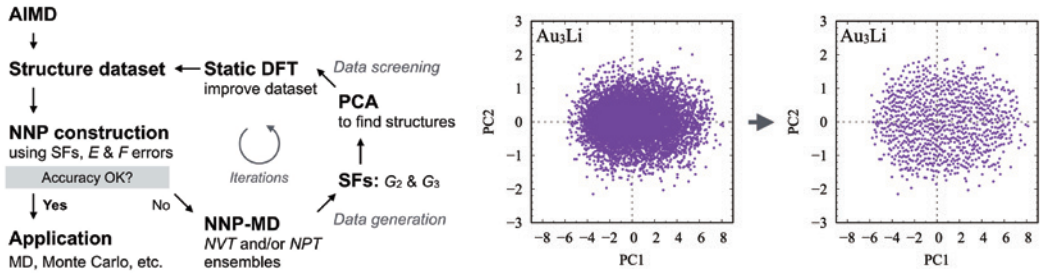
$$f_c\left(R_{ij}\right) = \begin{cases} 0.5\times\left[\cos\left(\dfrac{\pi R_i}{R_c}\right)+1\right] & R_i < R_c \\ 0 & R_i > R_c \end{cases}.$$

次に，上述の SFs をニューラルネットワーク（NN）の入力とし、その出力を原子のエネルギー E_i と考える。本研究で使用した隠れ層 2 層の NN の表式は以下のようになる。

$$E_i = f^2\left[w_{01}^2 + \sum_k w_{k1}^2 f^1\left\{w_{0k}^1 + \sum_j w_{jk}^1 f^0\left(w_{0j}^0 + \sum_i w_{ij}^0 G_i\right)\right\}\right].$$

w は各層間のノードを結ぶ重みパラメータ，f は活性化関数である。そして，構成原子のエネルギーをすべて足し合わせたものを系の全エネルギー E_{total} と考える。

$$E_{\text{total}} = \sum_{i=1}^{N} E_i.$$



第2図: ニューラルネットワークポテンシャル（NNP）作成のためのワークフロー図 [10]。

左図は本研究で開発した高精度な NNP を効率的に作成するためのワークフローをあらわしている。右図は Au_3Li のデータセットに対して，対称性関数（SFs）に基づく主成分分析（PCA）を用いて構造の類似性を評価し，データの多様性を保ちつつデータ数を削減した様子をあらわしている。

また，各原子にかかる力は以下の解析的な形で求めることができる。

$$F_{\alpha_i} = -\frac{\partial E_{\mathrm{total}}}{\partial \alpha_i} = -\sum_{\nu} \frac{\partial E_{\mathrm{total}}}{\partial G_\mu} \frac{\partial G_\mu}{\partial \alpha_i}, (\alpha = x, y, z).$$

そして，第一原理計算で得られるエネルギーと力の情報をもとに，以下に示す誤差関数 Γ を最小化するように $\boldsymbol{w}$ を更新することで原子間ポテンシャルを作成する。

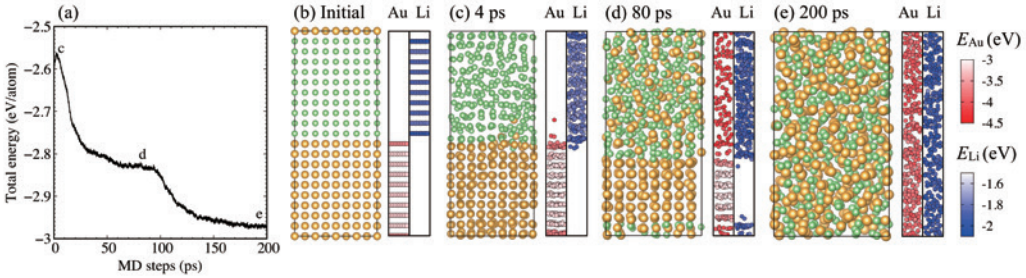
$$\Gamma(\boldsymbol{w}) = \frac{\alpha}{N_{\mathrm{train}}} \sum_{i=1}^{N_{\mathrm{train}}} \left(\frac{E_i^{\mathrm{NNP}} - E_i^{\mathrm{DFT}}}{n_i}\right)^2 + \frac{\beta}{N_{\mathrm{train}}} \sum_{i=1}^{N_{\mathrm{train}}} \left\{\sum_{j=1}^{3n_i} \frac{|F_j^{i,\mathrm{NNP}} - F_j^{i,\mathrm{DFT}}|^2}{3n_i}\right\}$$

なお，本研究では L-BFGS 法[9]により重みパラメータを最適化した。以上の NNP の概略を第1図に示す。

3. 結果

本章ではまず，本研究で開発した効率的な NNP 作成のための構造データセットの作成方法[10, 11]について概説する。Au と Li の合金化過程の解析に適応可能な NNP を作成することを念頭に，Au と Li に加えて，規則合金の Au_3Li, AuLi, $AuLi_3$，および Au/Li の超格子構造を訓練データに含む基本構造とした。そして，これらの基本構造に対して第一原理分子動力学計算（ab inito molecular dynamics; AIMD）を短時間（1 ps）実施し、そこで得られたデータセットを用いて NNP を作成した。通常，ここでの一時的な NNP は実用的な精度を期待できないが，訓練データを拡充する用途として使用することは可能である。そこで，NNP を用いた MD 計算を行うために，原子シミュレーションソフトウェアとして著名な LAMMPS[12, 13]のインターフェースを作成した。NNP は AIMD よりも 3~4 桁程度高速であることを活用し，様々な条件での MD 計算を比較的長時間（10~100 ps）行い，多様な構造データを作成した。

次に，比較的少数の訓練データで多様な構造的特徴を機械学習できるようにすることを目的に，上述の NNP による MD 計算の軌跡から得られた構造に対して，SFs を用いた主成分分析（principal component analysis; PCA）[14, 15]を行った。ここでは，一つの構造に含まれる SFs のパラメータ数×N原子の次元を



第 3 図: Au/Li 超格子構造の合金化過程の図 [10]。

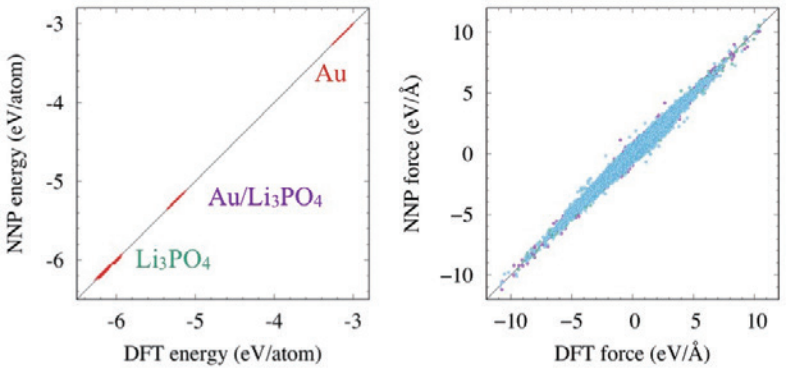
ニューラルネットワークポテンシャルを用いた分子動力学計算による（a）エネルギープロファイル，および（b-e）各シミュレーション時間における構造のスナップショット。また，Au と Li の原子エネルギーを各原子の相対位置に対してカラーマップ で示している。

2 次元（PC1 および PC2）に削減した。そして，主成分空間において近接するデータは類似性の高い構造とみなし，この空間内での各データ間の距離を指標に類似構造の選別を行った。図 2 に Au_3Li でのデータ数削減前後の主成分分布を示しており，データ選別後は多様な構造的特徴を均一にサンプルできていることがわかる。次に，データ選別後の構造のみに対して DFT 計算を行い，これらを訓練データに追加する。その後，NNP を再度作成し，必要な精度に達するまでこのサイクルを繰り返した。この方法により，計算コストの高い DFT 計算を少なく抑えることができ，NNP 作成の効率化に成功した。以上の NNP 作成のためのワークフローを第 2 図に示す。

上述の方法で作成した NNP を用いて，Au-Li 合金系の安定相の探索を行った[10]。固溶系材料の安定層探索は多くの組成比や結晶構造，原子配置について網羅的な調査が必要であり，膨大な計算コストがかかるとともに，二相間の自由エネルギー差が僅かである場合には高い計算精度が求められる。そこで本研究ではまず，Alloy Theoretic Automated Toolkit (ATAT) ソフトウェア[16]を用いて様々な合金構造を生成した。そして，そこで得られた約 77 万構造に対して NNP による構造最適化計算および混合エネルギー計算を行うことで，$Au_{1-x}Li_x$ における安定相を得た。また，各組成 x において NNP での予測エネルギーが低い 5 つの構造に対して DFT 計算を行った。その結果，両者の安定相にはいくつかの相違がみられるが，各組成での混合エネルギーの最小値はよく一致しており，NNP の予測精度の高さを示すことができた。なお，DFT と NNP の計算時間については，640 原子モデルの 1 つの構造のエネルギーと力を得るために Reedbush-U（MPI 144 並列）では約 24 分を要したが，NNP では約 0.2 秒と大幅に短縮することができた。

次に，先ほどと同じ NNP を用いて，Au と Li の合金化過程を調査した。ここでは，第 3 図(b)に示すように Au と Li が相分離した超格子構造を初期状態とし，MD 計算（圧力一定条件，N = 640 原子，P = 0 GPa，T = 500 K）による構造の変遷を調べた。その結果，シミュレーション開始早々（4 ピコ秒，第 3 図(c)）に Au 原子が Li へと固溶し，その後 80 ピコ秒まで断続的に Au 原子の固溶が進行し，リチウム全体が合金化している（第 3 図(d)）。この時点ではまだ固溶していない Au の領域が残っているが，200 ピコ秒では全域で合金化していることがわかる。さらに，2560 原子を含むモデルを用いた MD 計算では，合金化の描像をより明確にとらえることができた[10]。

最後に，Au/Li_3PO_4 界面系モデルに対する NNP の作成について報告する。4 元素系（Au–Li–P–O）の NNP 作成には，Au と Li_3PO_4 のバルク構造と表面構造，Au(111)/Li_3PO_4 界面構造を訓練データとして使用した。また，Li_3PO_4 中の Li イオン分布の変化を考慮するために欠陥を含む構造の考慮が必要となる。そこ



第 4 図: NNP と DFT のエネルギーと力の比較図 [18]。

Au/Li_3PO_4 界面系に対して作成した NNP による（左図）エネルギーと（右図）力の予測値と DFT 計算で得られた値を比較した図。

で，Li 空孔と格子間 Li の欠陥を含む構造も訓練データとして作成した。ここでも，第 2 図で概説した方法を用いて多様な原子配置の構造を作成し，DFT 計算で得られたエネルギーと力の情報をもとに NNP を作成した。作成した NNP によるエネルギーと力の予測値と DFT 計算で得られた値との比較を第 4 図に示す。ここでは，訓練に使用したデータ数は 2926 構造とまだ少ないが，バルクや界面のいずれの構造についてもエネルギーと力をよく予測できていることがわかる。今後，訓練データを追加して NNP の精度を向上させた後，比較的な大きな界面モデルを用いて Li イオン分布の解析を進める予定である[17]。

4. 結論

本研究では，金属–固体酸化物の界面構造とイオン伝導特性の大規模解析を行うことを目的に，第一原理計算と同等の計算精度で計算コストを大幅に低減できる機械学習ポテンシャルの作成に取り組んだ。作成したポテンシャルの事例としては，本稿では，$Au/Li_3PO_4/Li$ 積層型の新規不揮発性メモリの動作原理の理解に重要となる，Au-Li の合金化過程の解析結果を主に報告した。この過程で考案した効率的な NNP の作成方法などを活用して，界面系の NNP 作成とそれを用いた大規模解析を今後さらに進めていきたい。

5. 謝辞

本研究は、東京大学情報基盤センター「若手・女性利用者推薦」平成 30 年度（前期）の支援を受けたものであり，同センターおよび関係各位に感謝の意を表する。また本稿は，東京大学の渡邉聡教授と Elvis F. Arguelles 博士，産業技術総合研究所の安藤康伸博士と李文文博士（現 Preferred Networks），分子科学研究所の南谷英美准教授との共同研究の成果をもとに作成されたものであり，ここに感謝する。

参 考 文 献

[1] I. Sugiyama, R. Shimizu, T. Suzuki, K. Yamamoto, H. Kawasoko, S. Shiraki, and T. Hitosugi, APL Mater. **5**, 046105 (2017).

[2] K. Shimizu, W. Liu, W. Li, Y. Ando, S. Kasamatsu, E. Minamitani, and S. Watanabe, Phys. Rev. Matter. **4**, 015402 (2020).

[3] Y. Watanabe, S. Kobayashi, I. Sugiyama, K. Nishio, W. Liu, S. Watanabe, R. Shimizu, and T. Hitosugi, ACS Appl. Mater. Interfaces **11**, 45150 (2019).
[4] Y.A. Du and N.A. Holzwarth, Phys. Rev. B **76**, 174302 (2007).
[5] W. Li, Y. Ando, E. Minamitani, and S. Watanabe, J. Chem. Phys. **147**, 214106 (2017).
[6] J. Behler and M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007).
[7] A.P. Bartók, M.C. Payne, R. Kondor, and G. Csányi, Phys. Rev. Lett. **104**, 136403 (2010).
[8] A.P. Thompson, L.P. Swiler, C.R. Trott, S.M. Foiles, and G.J. Tucker, J. Comput. Phys. **285**, 316 (2015).
[9] J.L. Morales and J. Nocedal, ACM Trans. Math. Softw. **38**, 1 (2011).
[10] K. Shimizu, E.F. Arguelles, W. Li, Y. Ando, E. Minamitani, and S. Watanabe, Phys. Rev. B **103**, 094112 (2021).
[11] S. Watanabe, W. Li, W. Jeong, D. Lee, K. Shimizu, E. Minamitani, Y. Ando, and S. Han, J. Phys. Energy **3**, 012003 (2021).
[12] S. Plimpton, J. Comput. Phys. 117, 1 (1995).
[13] https://lammps.sandia.gov/
[14] B. Onat, E.D. Cubuk, B.D. Malone, and E. Kaxiras, Phys. Rev. B **97**, 094106 (2018).
[15] E.D. Cubuk, B.D. Malone, B. Onat, A. Waterland, and E. Kaxiras, J. Chem. Phys. **147**, 024104 (2017).
[16] A. van de Walle and G. Ceder, J. Phase Equilibria **23**, 348 (2002).
[17] K. Shimizu et al., to be published.

ロボット遠隔操作のための
複数台搭載魚眼カメラを用いた周囲環境の3次元再構成

小松 廉

東京大学大学院工学系研究科精密工学専攻

1．はじめに

ロボット遠隔操作の際にはオペレータは遠隔地にいるため周囲環境を直接見ることができない．したがって，ロボットに取り付けられた第一人称視点の映像からではロボット周囲環境の状況把握が非常に困難である．そこで，ロボットに取り付けられた複数台の魚眼カメラを用いて，ロボット周囲環境の3次元再構成を行うことで，オペレータが周囲環境を把握することが容易になる．本研究では，屋内環境を対象とした複数の魚眼映像から周囲360度の密な深度推定手法を提案する．特に複数視点映像からの深度推定手法として，学習ベースのPlane-sweeping stereoに注目する．実験においてカメラ姿勢の異なる複数のデータセットを用いて評価することで，従来手法は学習時のカメラ配置と大きく異なると深度推定精度が大きく低下するのに対して，提案手法においてはカメラ配置が大きく変化しても高精度な深度推定が可能であることを示した．また，提案手法は計算効率が非常に高く， GPUを搭載したノートPCを用いて，4枚の魚眼映像から0.42 sで全方位深度推定が可能であった．ソースコードはGithub[1]で公開している．

本研究の成果は，2020 IEEE/RSJ International Conference on Intelligent Robots and Systemsに採択されたものである [1]．詳細は[1]を参照することとし，本稿では概要を述べる．

2．提案手法

深層学習の発展により，通常の透視投影映像にConvolutional neural network（ConvNet）を適用することで，Plane-sweeping stereoを用いて高精度の深度推定が可能になった．しかし，全方位映像や魚眼映像は映像中に大きな歪みが存在するため，普通のConvNetの適用がうまく行かないことがある．例えば，OmniMVS [2]においては魚眼カメラ映像に直接ConvNetを適用したため，学習時のカメラ配置から姿勢変化した際に性能が著しく低下する．したがって，本研究では全方位映像や魚眼映像のために特別に設計された二十面体（Icosahedron）ベースの新しい表現方法とConvNetsを提案する．この表現方法は折り紙で作った王冠に似ていることからそのConvNetに「CrownConv」と名付けた．CrownConvは，魚眼映像および全方位映像の一般的な表現形式である正距円筒映像へ適用することができ，効果的な特徴量抽出が可能である．また，CrownConvは従来研究であるHexConv [3]の計算効率のボトルネックを解消することで，非常に高速な計算が可能である．

さらに，抽出された特徴量からIcosahedron上にCost volumeを生成するために，Icosahedronベースの Plane sweepingを提案する．Cost volumeは3次元CrownConvを用いて正則化し，最終的なCost volumeから全方位深度推定を行う．従来手法においては，Cost volume作成時のみ

[1] https://github.com/matsuren/crownconv360depth

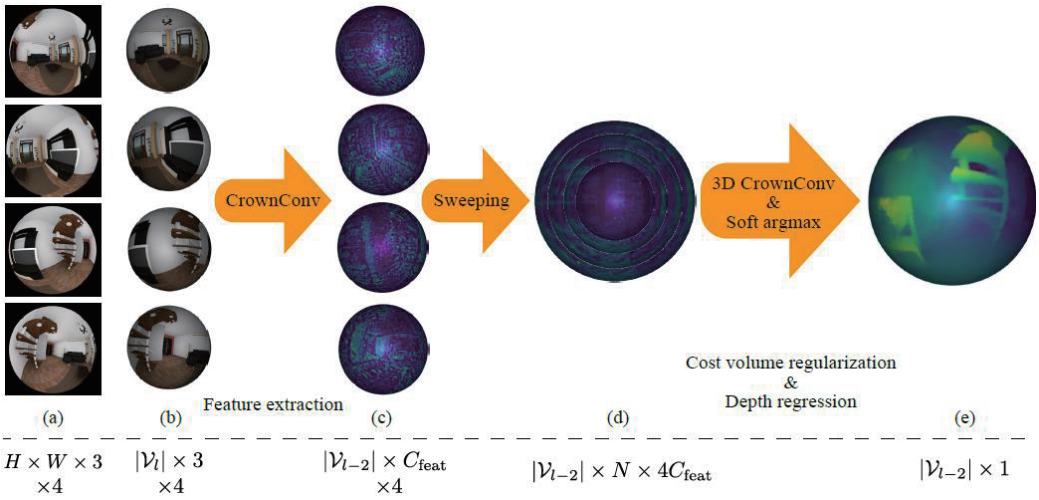


図 1　提案手法の流れ.

に外部カメラパラメータを用いていたが，提案手法においては特徴量抽出時においても外部パラメータを用いることで，カメラ配置の変化に頑健な深度推定が可能である．したがって，提案手法においてはカメラ配置が変化するたびに学習し直す必要がないという大きな利点がある．

提案手法の流れを図 1 に示す．図 1(a)がロボットに取り付けられた 4 台の魚眼カメラ映像である．まずは図 1(b)に示すように，魚眼カメラ映像の歪みを除去するために魚眼カメラ映像を Icosahedron 上に投影する．ここで，提案手法である CrownConv を用いて Icosahedron 上で魚眼カメラ映像の特徴量抽出を行う（図 1(c)）．得られた複数台の魚眼カメラ映像の特徴量は Icosahedron ベースの Plane sweeping を用いて Cost volume を作成する（図 1(d)）．最後に Icosahedron 上の Cost volume を 3 次元 CrownConv を用いて正則化し，全方位の深度推定を行う（図 1(e)）．本研究は，魚眼カメラ映像を幾何学的に扱うことで魚眼カメラ映像の歪みを補正し，提案手法の CrownConv を用いることで高速に全方位の深度推定を行えるようになった．

３．実験および結果

本実験において，学習時のカメラ配置から姿勢変化した際の性能を比較した．これは，ロボットに応じてカメラが設置可能である領域が違うことを想定している．従来手法のようにカメラ配置が学習時と異なる場合に性能が低下するということは，使用するロボットが変わりカメラ配置が変化するたびに学習が必要となってしまい現場適用性が低いことを意味する．

実験結果を図 2 に示す．図 2 において，全方位の深度推定結果を正距円筒画像の深度マップで示している．深度マップにおいて明るい色はカメラ群から近い領域であり，暗い色はカメラ群から遠い領域を示してる．図 2 で示すように，15 度，30 度，45 度と学習時から異なるカメラ姿勢を用いると，従来手法の OmniMVS [2]においては部分的に推定結果が悪化していくのに対して，提案手法においては安定した推定結果が可能であることがわかった．

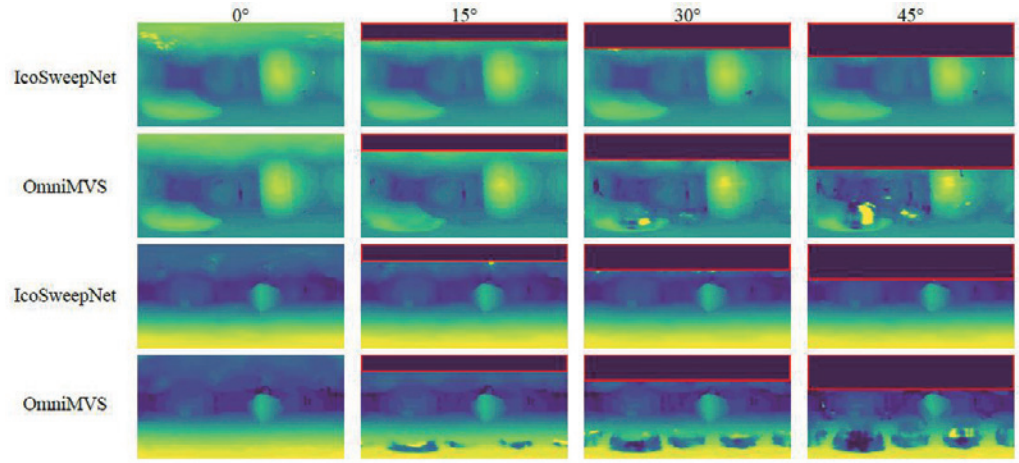


図 2　深度推定比較結果．提案手法（IcoSweepNet）と従来手法（OmniMVS [2]）．上に示した角度は，学習時のカメラ配置からの姿勢変化を表している．

4．まとめ

本研究では，屋内環境を対象とした複数の魚眼映像から周囲 360 度の密な深度推定手法を提案した．魚眼カメラ映像を Icosahedron 上に投影することで魚眼カメラ映像の歪みを除去し，提案手法である CrownConv を用いることで Icosahedron 上で特徴量抽出および Cost volume の正則化が可能になった．提案手法により，カメラ配置が学習時から大きく変化しても複数台の魚眼カメラ映像から高精度な深度推定が可能であることを示した．

謝辞

本研究の一部は，2019 年度後期の若手・女性利用者推薦制度の採択課題として行われ，東京大学情報基盤センターの Reedbush-H/L を用いた．また東京大学大学院工学系研究科精密工学の淺間一教授および山下淳准教授に多くの指導および助言をいただいた．

参　考　文　献

[1] Ren Komatsu, Hiromitsu Fujii, Yusuke Tamura, Atsushi Yamashita, and Hajime Asama, “360° Depth Estimation from Multiple Fisheye Images with Origami Crown Representation of Icosahedron”, Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 10092-10099, 2020.

[2] C. Won, J. Ryu, and J. Lim, “OmniMVS: End-to-End Learning for Omnidirectional Stereo Matching”, Proceedings of the 2019 IEEE International Conference on Computer Vision, pp. 8987-8996, 2019.

[3] C. Zhang, S. Liwicki, W. Smith, and R. Cipolla, “Orientation-aware semantic segmentation on icosahedron spheres”, Proceedings of the 2019 IEEE International Conference on Computer Vision, pp. 3533-3541, 2019.

Task Priority Control for the HPX Runtime System[1][2]

Suhang Jiang
Tohoku University

1. INTRODUCTION

In recent years, high-performance computing (HPC) is experiencing a phase change with the challenges of programming, management of heterogeneous multicore system architectures, and large scale system configurations. Synchronization [1] is the process of coordinating the behaviors of two or more processes, which may be running on different processors. Synchronization is needed for various collective communications such as parallel reduction. As the HPC system size increases, the overhead of a global synchronization involving a large number of processes is likely to become larger. Therefore, there is a demand for avoiding such expensive global synchronizations as much as possible to achieve high performance on an unprecedentedly large HPC system in the future.

Task-based execution is one promising approach to minimizing global synchronizations, because it does not perform expensive synchronization as long as task dependencies are not violated. High Performance ParalleX (HPX) [2][3] is one promising candidate of task-based programming and execution models, which provides a C++ class library to describe tasks and their dependencies, and also a runtime system for parallel computing based on the partitioned global address space (PGAS) model [4]. Therefore, this work focuses on HPX for task-based parallel execution.

In HPX, an OS-level thread called a worker thread executes tasks in a queue in a first-in first-out (FIFO) policy; tasks in the queue cannot overtake their preceding tasks. As a result, the execution of a task on the critical path can be delayed by executing other non-critical tasks. The OpenMP specification version 4.5 [5] and later support task priority, resulting in higher performance. However, as far as we know, such a task priority control mechanism has hardly been discussed for the HPX runtime system. Therefore, this paper discusses task priority control to improve the performance of a task-based HPX application by decreasing the waiting time of critical tasks in the queue.

The main contributions of this work are as follows:

1) Task priority control in HPX is achieved by using decoupled thread pools,

2) An appropriate thread mapping strategy to prioritize critical tasks is explored, and

3) Performance improvement by task priority control is quantitatively evaluated.

---

[1] 課題採用時の情報は、2019年度若手・女性後期採択課題の「並列プログラミングモデルHPXとXMPの比較と改善」となります。

[2] この論文はiWAPT2020にて発表しました。

The rest of this paper is organized as follows. Section 2 briefly reviews the related work. Section 3 presents the proposed mechanism to achieve task priority control in the HPX runtime system. Section 4 shows some evaluation results, and discusses the impact of task priority control on performance. Finally, Section 5 gives some concluding remarks, describes our future work.

2. RELATED WORK

Task-based execution is promising to prevent global synchronizations and thus to achieve high performance on a largescale future HPC system, on which a global synchronization is even more expensive [6]. HPX [2][3] is a parallel runtime system designed to overcome the limitations of conventional runtime systems such as starvation, communication latencies, remote resource access overheads, and the waiting time until contention resolution. It also provides a C++ class library for task-based programming.

HPX uses standard C++ classes for asynchronous operations, such as future and promise. A future class object is used to retrieve a value that is set by a different thread using a promise class object. In C++11 and later, thus, two different threads can share data using future and promise class objects. In HPX, the thread using a promise object could run on a different node. First, when a future class object is created by a thread running on one node, the thread is not blocked and continues to execute. In the background, another thread is launched on the same node to implicitly manage the inter-node communication. Then, the latter thread is waiting until the value of a promise class object is set by a different thread potentially running on another node. Finally, the value is shared via inter-node communication.

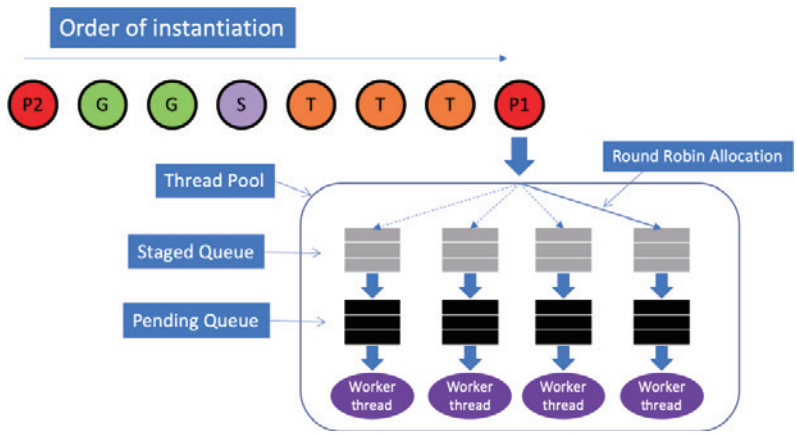


Fig. 1. Task execution with the original mechanism. Every task is executed by one of worker threads in the default thread pool. A red task (P1 and P2) is critical and can start running right after its preceding purple task (S).

Figure 1 illustrates how tasks are executed in the original HPX runtime system. In the HPX runtime system, a task is created when a future class object is instantiated,

and then assigned to one of the worker threads in a round-robin fashion. Each task can be in one of five states. The first one is running, which means that the context is active, and the code is running as if it runs with a native thread. The second state is suspended, which means that the task is waiting for a synchronization event. The third state is staged, which means that the task has been created, but cannot start execution yet, because its dependency is not satisfied. The fourth state is pending, which means that the task is ready to run, but still needs to wait in the task queue until a worker thread becomes available. The fifth state is terminated, which usually means that the task execution is finished. To manage the stages of each task, a worker thread has two kinds of task queues, the staged and pending queues. When a task is assigned to a worker thread, the task is first pushed to the staged queue until its dependency is satisfied. When the task dependency is satisfied, the task is moved to the pending queue and waits until the corresponding worker thread becomes available for the execution. Tasks in the pending queue are executed in a FIFO policy, and cannot overtake their preceding tasks. Thus, the execution of a task on the critical path could be delayed by executing other threads, if the critical task is moved to the pending queue after the noncritical ones. Suppose that red tasks in Figure 1, P1 and P2, are critical, and P2 needs only the result of task S. Then, task P2 can start running right after task S. However, task P2 needs to wait for task G if tasks G and P2 are assigned to the same worker thread. As a result, the critical task needs to wait longer than necessary, and hence the total execution time increases. Therefore, this paper improves the HPX performance by giving a higher execution priority to critical tasks.

The OpenMP specification version 4.5 [5] and later support a mechanism to specify the priority of a task in the task queue. This priority is a hint to the OpenMP runtime system for the order of task execution. During the execution, the task scheduler will execute higher priority tasks before lower priority ones as long as the task dependencies are not violated. In HPX, a task can still wait in the pending queue even if its dependency is satisfied. The task will wait until a worker thread becomes available in the default thread pool. In contrast to the mechanism of OpenMP, our proposed method uses decoupled thread pools to control the priority of tasks. By using different thread pools for critical and non-critical tasks, the critical tasks can be executed by worker threads without interference from non-critical tasks.

This paper also discusses a thread mapping method to map each worker thread to a different processor core. A better thread mapping method can mitigate the load imbalance and improve the performance of a task-based application. HPX provides four different built-in thread mapping methods. One of the four built-in methods is selected when an HPX application is launched. The first method is the most standard way of thread mapping, called compact. Using this method, all the threads will concentrate on as fewer sockets or NUMA domains [2] as possible. The second method, called scatter,

evenly assigns threads to physical cores and NUMA domains. The third and fourth methods, called balanced and NUMA-balanced, are similar to scatter. They assign threads to physical cores and NUMA domains in a round-robin fashion.

However, balanced assigns consecutive threads to different physical cores, while NUMA-balanced assigns consecutive threads to different NUMA domains. One problem of the built-in mapping methods is that all the methods assume to use the default task management of HPX. If multiple thread pools are used for the execution as proposed later in this paper, the built-in mapping methods may not effectively improve performance. In this work, therefore, it is necessary to design and implement a thread mapping method for exploiting the potential of multiple thread pools.

3. TASK PRIORITY CONTROL FOR HPX

As shown in Figure 1, in the original HPX runtime system, a task is created when a future class object is instantiated. Then, tasks are assigned to worker threads in a round-robin fashion. Each worker thread has two kinds of task queues. A task waits in the staged queue until the task dependency is satisfied. After that, the task moves to the pending queue and waits until the worker thread is available for the task execution. As a result, tasks assigned to one worker thread are executed in an FIFO fashion; a subsequent task cannot overtake its preceding tasks. Therefore, the execution of critical tasks can be delayed by executing its preceding tasks no matter if the preceding ones are critical.

In this work, we propose a task priority control mechanism for the HPX runtime system so that critical tasks can be executed right after their task dependencies are satisfied. In the proposed mechanism, worker threads are grouped into two different thread pools. Worker threads in one thread pool are used only for critical tasks, while worker threads in the other thread pool are for non-critical tasks. Therefore, since critical and non-critical tasks are respectively assigned to different thread pools, the execution of critical tasks is never blocked by the execution of non-critical tasks.

In the proposed mechanism, critical tasks are prioritized as follows. Suppose that the task dependencies of an application are represented as a directed acyclic graph (DAG). Then, critical tasks can be selected with considering the DAG. The proposed mechanism assumes that programmers are responsible for identifying critical tasks of an application. As a typical example, in this paper, critical tasks are defined as the tasks that appear most frequently on the critical path of an application. The critical tasks are automatically detected by using the following three steps. First, we find the critical path of the DAG by finding the longest path in the DAG. Second, we calculate the number of occurrences of each task in the critical path. Finally, the tasks that have the highest number of occurrences in the critical path are detected as the critical tasks. After that, only the critical tasks are assigned to a dedicated thread pool instead of the default thread pool, while the other non-critical tasks are

assigned to the default thread pool. In addition, since worker threads in each of the two thread pools can be mapped to processor cores in various ways, this paper also proposes a thread mapping method for decoupled thread pools.

As shown in Figure 1, in the original HPX runtime system, all tasks are assigned to worker threads in the default thread pool. If the result of one task is required by other subsequent tasks, the subsequent tasks are blocked in the staged queues until the result of the preceding task becomes available. Suppose that a critical task is blocked. Then, if some other tasks are in the pending queue, they are executed earlier than the critical task, potentially increasing the waiting time of the critical task and hence prolonging the critical path. Therefore, the original HPX mechanism needs an additional mechanism for giving a higher priority to critical tasks so as to prevent prolonging the critical path.

Figure 2 illustrates how tasks are executed with the proposed mechanism. Unlike the original mechanism, the proposed mechanism can prioritize to execute critical tasks by using a dedicated thread pool for the critical tasks, called a critical thread pool. In the proposed mechanism, critical task P2 can be executed right after task S. Since the critical tasks and non-critical tasks are assigned to different thread pools, the critical tasks in the pending queues do not need to wait for noncritical tasks. Therefore, the proposed mechanism can prevent delaying the execution of critical tasks. Because for each task queue, it is divided into staged queue and pending queue, between the decoupled thread pools, task dependencies can be achieved better.

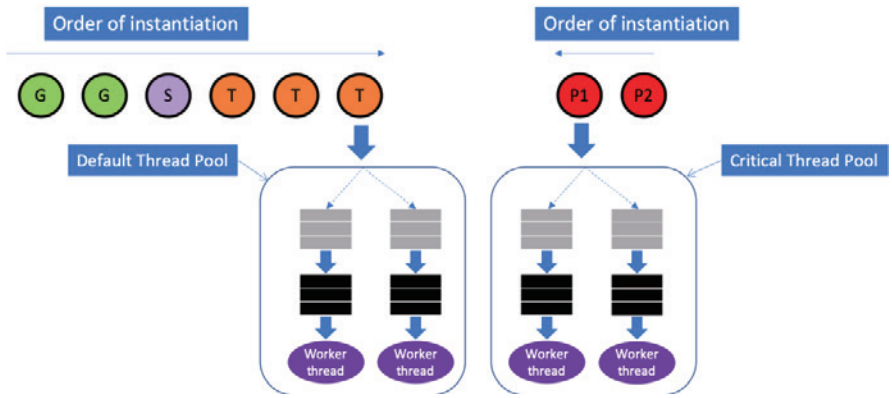


Fig. 2. Task execution with the proposed mechanism. Only critical tasks (P1 and P2) are assigned to worker threads in the critical thread pool.

In the original HPX runtime system, all worker threads are in a single thread pool, and selected for individual tasks while considering only their localities, i.e., NUMA domains. As shown in Figure 2, however, the proposed mechanism uses two types of worker threads for critical and non-critical tasks, respectively. A newly arisen problem is how to map worker threads to processor cores with considering not only their localities but also their types. Therefore, in Section IV, we discuss the performance gain by the

proposed mechanism, and also the effect of using a different thread mapping method on performance.

In HPX, a task can be executed by any threads of a thread pool by using a task executor called **pool_executor**. We use **hpx::async** function call to execute all the tasks asynchronously. Which thread pool is used for executing a task can be determined by specifying the parameter of thread pool in the executor. Thus, to assign the critical tasks to the critical pool, we set the critical pool as a parameter of the executor that runs the critical tasks. On the other hand, we use the default thread pool for the executor running the non-critical tasks. Listings 1 and 2 show the code snippets of the merge sort benchmark using the default thread pool and the decoupled thread pools, respectively. In this benchmark, the tasks are divided in a recursive way, and they are run in parallel by using the hpx::async function. The differences between the default and the proposed mechanisms are shown in the code, starting from Line 22. As shown in Listing 2, the proposed mechanism first sets the critical pool as the thread pool of the critical_executor (Lines 23-24). Then, the critical pool is used for executing the merge task by specifying the critical_executor as the task executor for the task.

Listing 1. The merge sort with the default thread pool.

```
1  std::vector<int> mergesort_par(std::vector<int> v)
2  {
3    if (v.size() <= 2)
4    {
5      // Sort task
6      return serial_sort(v);
7    }
8    else
9    {
10     //Split task
11     std::tuple<std::vector<int>,
12       std::vector<int>> splits = split(v);
13     std::vector<int> left = std::get<0>(splits);
14     std::vector<int> right = std::get<1>(splits);
15
16     //Divide tasks recursively
17     hpx::future<std::vector<int>> f_left = hpx::async(
18       mergesort_par, hpx::find_here(), left);
19     hpx::future<std::vector<int>> f_right = hpx::async(
20       mergesort_par, hpx::find_here(), right);
21
22     //Merge task using the default thread pool
23     hpx::future<std::vector<int>> fm =
24       hpx::async(merge, hpx::find_here(), f_left.get(),
25          f_right.get());
26   }
27 }
```

Listing 2. The merge sort with the decoupled thread pools.

```
1  std::vector<int> mergesort_par(std::vector<int> v)
2  {
3    if (v.size() <= 2)
4    {
5      // Sort task
6      return serial_sort(v);
7    }
8    else
9    {
10     //Split task
11     std::tuple<std::vector<int>,
12       std::vector<int>> splits =  split(v);
13     std::vector<int> left = std::get<0>(splits);
14     std::vector<int> right = std::get<1>(splits);
15
16     //Divide tasks recursively
17     hpx::future<std::vector<int>> f_left = hpx::async(
18         mergesort_par, hpx::find_here(), left);
19     hpx::future<std::vector<int>> f_right = hpx::async(
20       mergesort_par, hpx::find_here(), right);
21
22     //Merge task using the critical thread pool
23     hpx::threads::executors::pool_executor
24       crit_executor(CRITICAL_POOL_NAME);
25     hpx::future<std::vector<int>> fm =
26       hpx::async(critical_executor, merge,
27         hpx::find_here(), f_left.get(), f_right.get());
28   }
29 }
```

A. Thread Mapping for Multiple Thread Pools

Since the built-in mapping methods of HPX are only applicable for the default thread pool, this work proposes a thread mapping algorithm for multiple thread pools. We adopt the built-in NUMA-balanced method of HPX to implement the proposed algorithm, NUMA-balanced-mtp. The key difference between these two algorithms is that NUMA-balanced-mtp distributes the threads from multiple thread pools among the NUMA domains. We choose the NUMA-balanced mapping because it can effectively improve performance on a NUMA system by reducing the load imbalance among the NUMA domains [7][8].

Algorithm 1 depicts the NUMA-balanced-mtp algorithm. First, the algorithm retrieves the available NUMA domains using the resource::partitioner object of HPX (Line 1). Then, it iterates the threads of all the thread pools. The next(pools) function starts from the critical thread pools to give a higher priority to the critical thread pools to be mapped than the other thread pools (Lines 2 and 9). For each thread, the algorithm selects a target NUMA domain in a round-robin fashion (Line 6). The threads are distributed among the NUMA domains to reduce the load imbalance. Then, it maps the thread to a processor core of the target NUMA domain (Line 7). To map a thread to a processor core, the map function uses the add_resource routine of the resource::partitioner object.

Algorithm 1 The NUMA-balanced-mtp Algorithm

Input: *rp* {Resource partitioner object of HPX}
Input: *pools* {Thread pools}
1: $domains \leftarrow rp.numa\_domains()$
2: $pool \leftarrow next(pools)$
3: **while** *current_pool* **do**
4: **for** *thread in pool* **do**
5: {Iterate the NUMA domains in a round-robin fashion}
6: $target\_domain \leftarrow next(domains)$
7: $map(thread, target\_domain)$
8: **end for**
9: $pool \leftarrow next(pools)$
10: **end while**

TABLE I THE CONFIGURATIONS OF THE OAKBRIDGE-CX AND KNL4 SYSTEMS.

Machine	NUMA domains	Cores/domain	Threads/core
Intel Xeon Phi KNL	4	18	4
Oakbridge-CX	2	28	1

4. EVALUATION AND DISCUSSIONS

As shown in Figure 1, in the original HPX runtime system, a task is created when a future class object is instantiated. Then, tasks are assigned to worker threads in a round-robin fashion. Each worker thread has two kinds of task queues. A task waits in the staged queue until the task dependency is satisfied. After that, the task moves to the pending queue and waits until the worker thread is available for the task execution.

A. Evaluation Setup

In this section, we evaluate the performance gain by the proposed mechanism, using two benchmark programs on the following two different systems. The first system is a large cluster system of Intel Xeon Platinum 8280 processors, called Oakbridge-CX (OCX) [9], which is installed at the Information Technology Center, the University of Tokyo. The second is an Intel Xeon Phi Knights Landing (KNL) system [10], called KNL4. The system is configured as a four-node NUMA system by setting Sub-NUMA clustering (SNC) mode as the clustering mode of the KNL. In this mode, the system is partitioned into four NUMA nodes, with 72 logical cores per NUMA node. The hardware configurations of OCX and KNL4 are shown in Table I. HPX version 1.4.0 is used as the runtime system for all the experiments.

One benchmark used to evaluate the performance is Cholesky factorization [11] to compute,

$$A = LL^*; \quad (1)$$

where A is a Hermitian positive-definite matrix, L is a lower triangular matrix with real and positive diagonal entries, and L* denotes the conjugate transpose of L. The benchmark program consists of four kinds of tasks; Cholesky decomposition (potrf), solving the triangular matrix equation (trsm), matrix multiplication (gemm), and

symmetric rank-k update (syrk). A DAG of the task dependency for tiled Cholesky factorization of 4×4 tiles is shown in Figure 3. Once the k-th panel is factorized and then broadcast, the next most urgent task to complete is factorization of panel k + 1, which is a so-called **look-ahead** [12]. Therefore, it is already known that the potrf tasks in the DAG are critical.

The other benchmark is an implementation of merge sort. In the case of merge sort, the DAG consists of split, sort and merge tasks, and there is no clear critical path, as shown in Figure 4. However, as shown in the graph, the split and marge tasks appear most frequently on every path. Compared with the split task, the merge task is more computationally expensive because it needs to merge the previous results of the sort tasks in a sorted order. Therefore, in this work, the merge task is considered as a critical task. Only merge tasks are assigned to worker threads in the critical thread pool.

B. Evaluation Results

First, the performance gain by the proposed mechanism is evaluated with the tiled Cholesky factorization whose DAG is shown in Figure 3. The tile size is set to 4×4 elements, and thus each task operates on a submatrix of 4×4 elements. Two thread mapping methods, the default and NUMA-balanced-mtp thread mapping methods are compared in the following evaluation. In the HPX runtime system, as shown in the Figure 5, balanced mapping is used as the default mapping. Thus, the default mapping distributes worker threads to all the physical cores without considering the NUMA domains. On the other hand, the NUMA-balanced-mtp distributes all worker threads of the decoupled thread pools among the NUMA domains as follows. First, it assigns consecutive threads of the critical thread pool to different NUMA domains. Then, it assigns consecutive threads of the default pool to different NUMA domains.

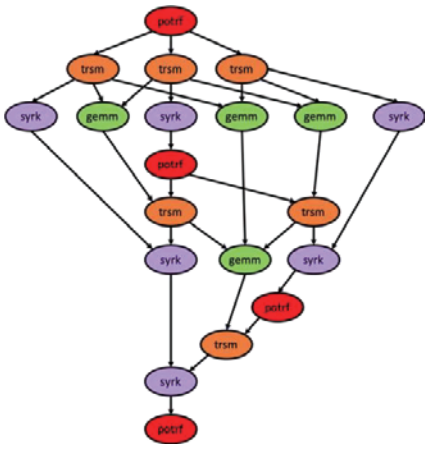


Fig. 3. Task dependency for tiled Cholesky factorization of 4×4 tiles. The potrf tasks are critical.

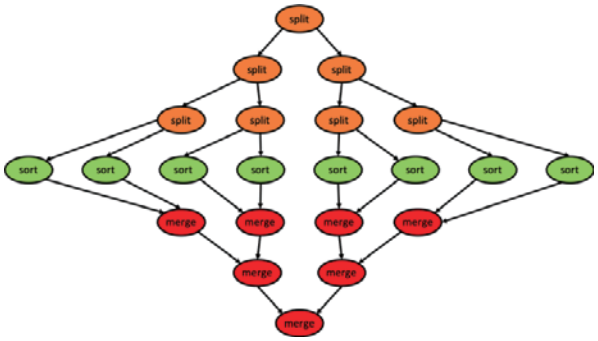


Fig. 4. Task dependency for merge sort (8-way parallel execution). The merge tasks are handled as critical tasks in the evaluation.

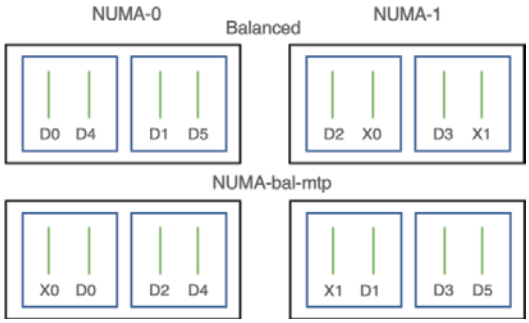


Fig. 5. Example of mapping results with two NUMA domains, six threads in the default pool, two threads in the critical pool. Here Xi is the worker thread i of the critical thread pool, and Di is the worker thread of the default thread pool.

For the evaluation on OCX, only one node is used and thus 56 threads are executed on two NUMA domains. The size of the matrix is set to 2048×2048, and the numbers of tiles are thus 512. The evaluation results are shown in Figure 6. In the figure, the horizontal axis indicates the configuration of thread pools used for executing the benchmark. The tuple shown in the horizontal axis represents the numbers of threads in the critical thread pool and the default thread pool. A tuple (c, d) means that c threads are in the critical thread pool, and d threads in the default thread pool. The first column, called default, uses the single default thread pool. As shown in the figure, the use of decoupled thread pools with the default thread mapping method could decrease 33.5% of the execution time at most on OCX by properly adjusting the configuration of thread pools. The results from (28, 28) configuration show that balancing the number of threads in default and critical thread pools cannot achieve the best performance. On the other hand, the (50, 6) configuration shows the highest performance, indicating that allocating more threads to the critical thread pool can significantly improve the performance even if the number of threads of critical pool is much higher than that of the default pool. These results also suggest that the time spent for waiting the

critical tasks has a significant impact on the performance of the Cholesky benchmark.

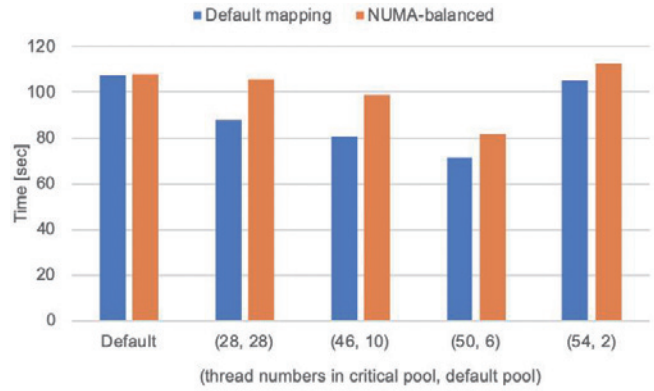


Fig. 6. Performance evaluation results of Oakbridge-CX for the Cholesky benchmark.

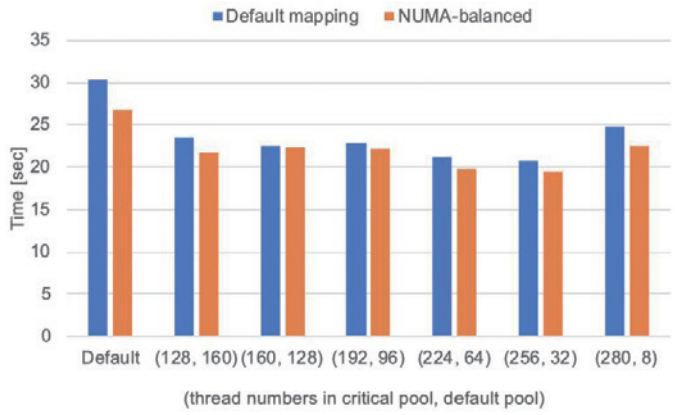


Fig. 7. Performance evaluation results of Knights Landing for the Cholesky benchmark.

On the KNL4 system, all of 288 threads are running on different logical cores. The size of the matrix is set to 512×512, and the numbers of tiles are thus 128. The evaluation results on KNL4 system are shown in Figure 7. The results indicate that the performance gain by the proposed mechanism increases as the number of worker threads in the critical pool, and the execution time is reduced by 31.8% at the thread pool configuration of (256, 32) with the default thread mapping method. These results show that the use of decoupled thread pools can significantly improve performance even if the number of threads in the critical pool is sufficiently large. The performance results on OCX and KNL4 clearly show that the proposed mechanism can achieve a higher performance than the default mechanism of HPX by giving a higher priority to critical tasks. Figure 8 shows the waiting time in the staged and pending queues with different numbers of threads allocated for critical and non-critical tasks. These results are obtained using two HPX performance counters, called the staged time and the pending

time. As shown in the figure, the configuration with 256 threads in critical thread pool can achieve the shortest pending time and also the shortest staged time. This explains why this configuration can achieve the shortest execution time in Figure 7. Compared with the default, all the configurations of the proposed mechanism can reduce the waiting time. If the number of threads in the default thread pool is lower than 32, the number of threads for the non-critical tasks is too small, the waiting time of the tasks will increase significantly. Therefore, the proposed mechanism can reduce the waiting time of tasks even if the optimal number of worker threads in the critical thread pool is not known in advance.

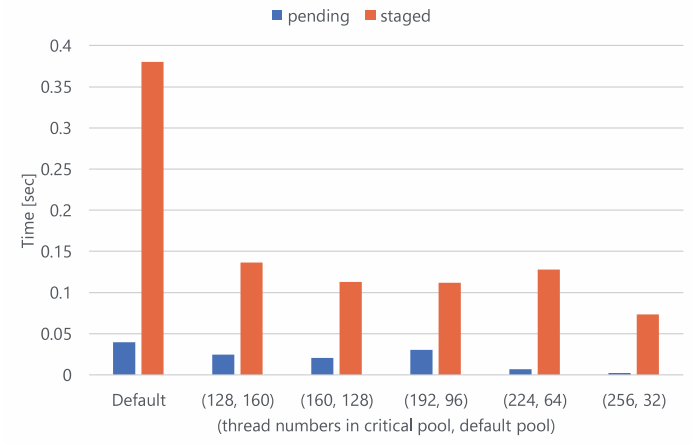


Fig. 8. The waiting time in the staged and pending queues.

Next, the effects of thread mapping policies on performance are investigated using the Cholesky benchmark program. Figures 6 and 7 also show the performance evaluation results with the two thread mapping methods. Figure 7 shows that, for the same problem, using the NUMA-balanced-mtp thread mapping can further increase the performance by 4.8% on KNL4. It is because, by distributing worker threads in different thread pools over different NUMA domains, the NUMA-balanced-mtp thread mapping method reduces the load imbalance among NUMA domains. The results also show that the proposed mechanism with the NUMA-balanced-mtp thread mapping method can further improve the performance, suggesting that the load balance among NUMA domains is important in improving performance of the proposed mechanism. However, Figure 6 shows that the NUMA-balanced-mtp thread mapping method decreases the performance on OCX, because the communication cost among NUMA domains on OCX is larger and thus cancels the performance gain by reducing the load imbalance. It is because the number of processor cores in the OCX is much lower than that in the KNL4. The impacts of reducing the load imbalance to the execution time of the Cholesky benchmark is lower than those of the communication cost. Therefore, the best thread mapping method depends on the hardware configuration, and hence the thread mapping method should be carefully selected considering the target system.

Finally, the performance gain by the proposed mechanism is also done with the merge sort benchmark on the KNL4 system. In this evaluation, the thread mapping method of NUMA-balanced-mtp is used because it can improve the performance of the proposed mechanism as discussed above. The evaluation results with the merge sort benchmark on the KNL4 system is shown in Figure 9. The merge sort benchmark uses an array of 10^5 numbers as input. As shown in the figure, all the configurations of the proposed mechanism can achieve a higher performance than the default mechanism. In this particular benchmark program, the best configuration is the (256, 32) configuration, which decreases the execution time by 47% in comparison with the default mechanism. Accordingly, the results clearly show that the proposed mechanism can improve the performance even if there is no clear critical path in the DAG. As previously analyzed, the merge tasks are more computationally-expensive than the others, and hence they should be executed earlier so that the execution of the merge tasks can be overlapped with that of other tasks. These results also show the importance of identifying the critical tasks in improving the performance of the task-based application.

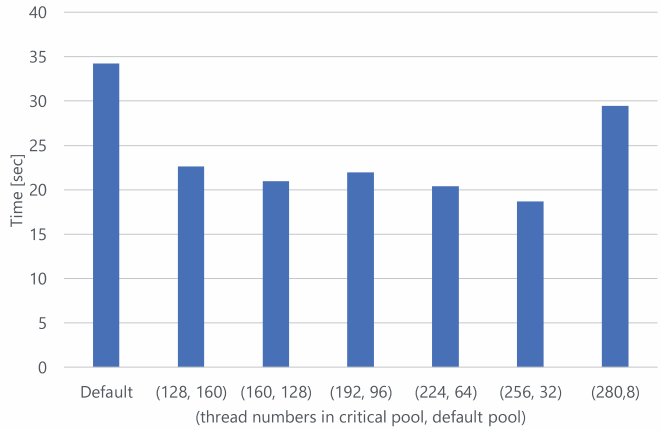


Fig. 9. Performance evaluation results of Knights Landing for the merge sort benchmark.

TABLE II RATIOS OF THE NUMBER OF EXECUTED THREADS AND THREAD TIME OF THE BENCHMARKS.

Ratios	Number of executed threads	Thread time
Cholesky decomposition	0.1	13.3
Merge sort	0.99	0.6

As an analysis of the results on the KNL4 system, Table II shows the ratios of the number of executed threads and thread times of the critical thread pool and the default thread pool. As shown in the table, the ratio of the number of executed threads of the two benchmarks are significantly different. For the Cholesky decomposition benchmark, the number of executed threads of the critical thread pool is much lower than that of the default thread pool. The ratio of the number of executed threads of Cholesky is

0.1, which means that the number of threads executed for critical tasks is only 10% of that is executed for non-critical tasks. In contrast, for the merge sort benchmark, the number of executed threads of the critical thread pool is almost equal to that of the default thread pool. As previously analyzed from the DAG of the merge sort benchmark, the numbers of split and merge tasks in each path are equal. The results of the number of executed threads show that these two benchmarks have different workloads, although the best number of threads configurations of the benchmarks are the same. Moreover, the ratio of thread times of the merge sort benchmark shows that the decoupled thread pools mechanism can significantly decrease the average execution time of critical tasks. The ratio of thread times of the merge sort is 0.6, which means that the average execution time of critical tasks is 60% shorter than that of the non-critical tasks. The results shown in Table II suggest that the impacts of the decoupled thread pools mechanism depend on the application workload and the hardware configuration of the system.

5. CONCLUSIONS

This paper has proposed a task priority control mechanism that uses decoupled thread pools in order to prioritize the execution of critical tasks. By using a pool of worker threads dedicated to critical tasks, the proposed mechanism can prevent critical tasks from waiting for non-critical tasks. As a result, the proposed mechanism can significantly reduce the waiting time of critical tasks, and hence the total execution time. In addition, the effects of using different thread mapping methods are also investigated empirically. The performance evaluation results clearly demonstrate that the proposed mechanism can reduce the staged time and the pending time, in which tasks are waiting for other tasks. As a result, the proposed mechanism can reduce the total execution time of the Cholesky benchmark program by approximately 33.5% and 36.1% at maximum on OCX and KNL4, respectively. In addition, the NUMA-balanced-mtp thread mapping method can further improve the KNL4 performance, even though it degrades the OCX performance. Accordingly, the proposed mechanism can improve the performance of task-based HPX applications by prioritizing critical tasks with a low runtime overhead especially if its parameters, such as the number of threads in each thread pool and the thread mapping method, are appropriately adjusted for the target system.

As mentioned above, several parameters of the proposed method are empirically adjusted by hand for the application and target system in advance. Since the performance is sensitive to the parameter values, auto-tuning of them will be an interesting research topic. Automatic detection of critical tasks in a DAG would also be important in practical use. In addition, we can expect that the proposed mechanism can improve the runtime systems for other task-based PGAS languages, such as XcalableMP [13]. These topics will be discussed in our future work.

ACKNOWLEDGMENTS

This work was propelled by Suhang Jiang, Mulya Agung, Ryusuke Egawa, Hiroyuki Takizawa.

The authors would like to thank Prof. Hiroaki Kobayashi of Tohoku University and Dr. Kentaro Sano of RIKEN R-CCS for fruitful discussions on this work.

This work is partially supported by MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program "R&D of A Quantum-Annealing- Assisted Next Generation HPC Infrastructure and its Applications," Grant-in-Aid for Scientific Research(B) #16H02822 and #17H01706, and Initiative on Promotion of Supercomputing for Young or Women Researchers, Information Technology Center, The University of Tokyo.

REFERENCES

[1] T. E. Hart, P. Mckenney, A. K. D. Brown, and J. Walpole, "Performance of memory reclamation for lockless synchronization," Journal of Parallel and Distributed Computing, vol. 67, no. 12, pp. 1270-1285, 2007.

[2] H. Kaiser, T. Heller, B. Adelstein-Lelbach, A. Serio, and D. Fey, "HPX: A task based programming model in a global address space," in 2015 IEEE International Conference on Cluster Computing, 2015, pp. 682-
689.

[3] H. Kaiser, M. Brodowicz, and T. Sterling, "ParalleX: an advanced parallel execution model for scaling-impaired applications," in International Conference on Parallel Processing Workshops, 2009, pp. 394-401.

[4] K. Yelick, D. Bonachea, W.-Y. Chen, P. Colella, K. Datta, J. Duell, S. L. Graham et al., "Productivity and performance using partitioned global address space languages," in The 2007 International Workshop on Parallel Symbolic Computation, 2007, pp. 24-32.

[5] OpenMP Architecture Review Board, "OpenMP application programming interface, version 4.5," 2015. [Online]. Available: https://www.openmp.org/specifications/

[6] J. Loaiza, S. Chandrasekaran, and N. MacNaughton, "Using local locks for global synchronization in multi-node systems," 2008, uS Patent 7,376,744.

[7] F. Gaud, B. Lepers, J. Funston, M. Dashti, A. Fedorova, V. Qu´ema, R. Lachaize, and M. Roth, "Challenges of memory management on modern numa systems," Communications of the ACM, vol. 58, no. 12, pp. 59-66, 2015.

[8] M. Agung, M. A. Amrizal, R. Egawa, and H. Takizawa, "Deloc: A locality and memory-congestion-aware task mapping method for modern numa systems," IEEE Access, vol. 8, pp. 6937-6953, 2020.

[9] K. Nakajima, "Parallel multigrid method on multicore/manycore clusters," in Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops, ser. HPCAsia2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 5-9. [Online]. Available: https://doi.org/10.1145/ 3373271.3373273

[10] J. Jeffers, J. Reinders, and A. Sodani, Intel Xeon Phi Processor High Performance

Programming: Knights Landing Edition. Morgan Kaufmann, 2016.
[11] J. Dorris, J. Kurzak, P. Luszczek, A. YarKhan, and J. Dongarra, “Taskbased Cholesky decomposition on knights corner using OpenMP,” in International Conference on High Performance Computing, 2016, pp.
544-562.
[12] J. J. Dongarra, P. Luszczek, and A. Petitet, “The LINPACK benchmark: past, present and future,” Concurrency and Computation: practice and experience, vol. 15, no. 9, pp. 803-820, 2003.
[13] K. Tsugane, J. Lee, H. Murai, and M. Sato, “Multi-tasking execution in PGAS language XcalableMP and communication optimization on many-core clusters,” in International Conference on High Performance Computing in Asia-Pacific Region, 2018, pp. 75-85.

学際大規模情報基盤共同利用・共同研究拠点（JHPCN）

公募型共同研究　2021 年度採択課題一覧

飯 野 孝 浩

東京大学情報基盤センター

１．公募概要

2021 年度の国際・一般研究課題には 56 課題の応募があり，2021 年 2 月に行われた課題審査委員会での厳正な審査により，49 課題が採択された。うち，東京大学の計算機資源を利用する課題の一覧を表 1 に示す。

表 1. 2021 年度 JHPCN 採択課題（一般・国際研究課題，東京大学共同研究分）

国際共同研究課題

研究課題名	研究課題代表者 （所属）
Developing Accuracy Assured High Performance Numerical Libraries for Eigenproblems	片桐孝洋 （名古屋大学）
High resolution simulation of cardiac electrophysiology on realistic whole-heart geometries	中島研吾 （東京大学）
Hierarchical low-rank approximation methods on distributed memory and GPUs	横田理央 （東京工業大学）

一般共同研究課題

研究課題名	研究課題代表者 （所属）
GPU・CPU・ARM プロセッサに対する原子力 CFD アプリケーション用の混合精度ポアソン解法	小野寺直幸 （日本原子力研究開発機構）
電磁流体力学乱流の高精度・高並列 LES シミュレーションコード開発研究	三浦英昭 （核融合科学研究所）
HPC と高速通信技術の融合による大規模データの拠点間転送技術開発と実データを用いたシステム実証試験	村田健史 （情報通信研究機構）
Deep Learning を用いた医用画像診断支援に関する研究	佐藤一誠 （東京大学）
多粒子分散系の乱流輸送に関する大規模シミュレーション	渡邊威 （名古屋工業大学）
高性能かつ高信頼な数値計算手法とその応用	荻田武史 （東京女子大学）

Development of physics informed machine learning for soft matter: polymer flows and beyond	John Molina （京都大学）
ハイブリッドクラウド構築とゲノム情報解析の効率的な運用に関した研究	長﨑正朗 （京都大学）
三次元強震動シミュレーションとリアルタイムデータ同化の融合	中島研吾 （東京大学）
時空間領域境界積分方程式法の高速解法の開発と巨大地震シミュレーションへの応用	安藤亮輔 （東京大学）
Innovative Multigrid Methods II	藤井昭宏 （工学院大学）
機械学習モデルのリアプノフ指数ならびにリアプノフベクトルの解析	齊木吉隆 （一橋大学）
大規模分散医用画像処理に向けた医用画像処理アプリケーションの最適化	大島聡史 （名古屋大学）
高レイノルズ数乱流のデータ科学プラットフォームの構築	石原卓 （岡山大学）
GPU の高速並列計算で実現する交差禁止制御可能な高分子シミュレータの開発	萩田克美 （防衛大学校）
合成人口プロジェクト：従業地・通学地属性の確率的割当てと深層学習による空中写真からの住宅判別	村田忠彦 （関西大学）
エクサスケール時代の数値計算手法に対する性能予測技術	深谷猛 （北海道大学）
グラフ構造で一般化された動的負荷分散フレームワークの構築と重合メッシュ法への適用	森田直樹 （筑波大学）
Developping data driven analysis methods for extreme scale numerical simulations	朝比祐一 （日本原子力研究開発機構）
Development of Fast Surrogate for Approximating Large-scale 3D Blood Flow Simulation	下川辺隆史 （東京大学）
財務ビッグデータの可視化と統計モデリング	地道正行 （関西学院大学）

学際大規模情報基盤共同利用・共同研究拠点（JHPCN）

第 13 回シンポジウム開催報告

飯 野 孝 浩

東京大学情報基盤センター

１．概要

7 月 8 日・9 日の 2 日間に渡って，学際大規模情報基盤共同利用・共同研究拠点（JHPCN）の第 13 回シンポジウムがオンラインにて実施された．本シンポジウムに先立ち，シンポジウムの目的や形式等を議論する WG が 2020 年度に 2 回実施され，ダブルトラックとして発表・議論の時間を確保すること，参加者間のネットワーキングのための施策を試行するなどの新機軸が決定されており，今回は実施形式を大きく変えての開催となった．新型コロナウイルス感染症のまん延のため，前年度に引き続いてオンラインでの開催となったが，前年度・今年度の課題代表者による発表は全てプログラムされ，新たに基調講演や，新規計算機資源の紹介等が企画された．基調講演では，「The future of medicine in the era of big data and AI」と題して，医療 AI 研究の世界的権威である Daniel Rückert 博士（ミュンヘン工科大学）の講演を実現した．参加登録者は約 350 名であり，オフライン実施であった例年の典型的な人数（約 200 名）を大きく超える人数であった.2.に当日のプログラムを示す.詳細はシンポジウムのウェブサイト(https://jhpcn-kyoten.itc.u-tokyo.ac.jp/ja/sympo/13th) を参照されたい.

２．プログラム

7 月 8 日（木）

10:00～10:10 主催者挨拶: 総括拠点長　田浦 健次朗 （東京大学 情報基盤センター長　教授）
10:10～10:20 来賓挨拶：宅間　裕子　様（文部科学省　計算科学技術推進室長）
10:20～12:00　オーラルセッション 1　（A 会場・B 会場）
13:20～14:00　新規提供計算資源紹介：東北大学，東京大学，大阪大学　（A 会場）
14:00～17:40　オーラルセッション 2　（A 会場・B 会場）
18:00～18:50 基調講演　（A 会場）

7 月 9 日（金）

10:00～12:00 オーラルセッション 3　（A 会場・B 会場）
13:00～13:30 ポスターセッション　コアタイム A　（A 会場・B 会場）
13:30～14:00 ポスターセッション　コアタイム B　（A 会場・B 会場）
14:20～17:40 オーラルセッション 4　（A 会場・B 会場）
17:40～18:00 閉会挨拶:共同研究課題審査委員長　片桐 孝洋（名古屋大学　情報基盤センター教授）

Wisteria/BDEC-01 利用事例（I）

Wisteria/BDEC-01（Odyssey）における疎行列解法の動的ループスケジューリングによる通信最適化

中 島 研 吾

東京大学情報基盤センター

0. 「Wisteria/BDEC-01 利用事例」について

東京大学情報基盤センター（本センター）では，2021 年 5 月 14 日に「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）の運用を開始した〔1〕。

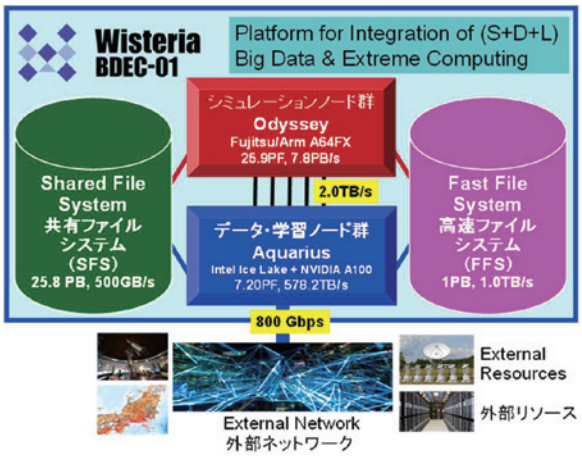


図 1　Wisteria/BDEC-01 の概要〔1〕

「Wisteria/BDEC-01」はシミュレーションノード群（Odyssey）とデータ・学習ノード群（Aquarius）の 2 つの計算ノード群を有するヘテロジニアスなシステムであり，特に「計算・データ・学習」融合が推進され，サイバー空間（仮想）とフィジカル空間（現実）を高度に融合させた Society 5.0 の実現に大きく貢献することが期待されている。「スーパーコンピューティングニュース」では，これから 1 年程度をかけて，「Wisteria/BDEC-01」の様々な利用事例を紹介していく予定である。今回は第 1 回として，シミュレーションノード群（Odyssey）における大規模並列数値解法の高速化について紹介する。

シミュレーションノード群（Odyssey）は「FUJITSU Supercomputer PRIMEHPC FX1000」20 ラックから構成され，世界最高性能を有するスーパーコンピュータ「富岳」と同じ富士通株式会社の「FUJITSU Processor A64FX」を 7,680 ノード（368,640 コア）搭載している。「A64FX」は，最先端の 7nm プロセスで製造され，48 個の演算コアと 2 個または 4 個のアシスタントコアを有し，倍精度浮動小数点演算で 3.3792 TFLOPS の理論ピーク性能を実現し，合計ピーク性能は 25.9 PFLOPS である。各ノードは 32 GiB の HBM2 メモリを搭載し，シミュレーションノード群（Odyssey）の総メモリ容量は 240 TiB，総メモリバンド幅は 7.8 PB／秒である。各ノードはバイセクションバンド幅が 13.0 TB／秒のノード間相互結合ネットワーク（Tofu インターコネクト D）で結合されている。

1. はじめに

大規模な並列計算機を使用する場合，ノード数の増加によって通信のオーバーヘッドは増加する傾向にある。並列計算において通信は必須のプロセスであるが，通信をできる限り効率的に実施し，削減することは EFLOPS 級システムにおいて重要である。通信

の削減がアルゴリズムに不安定をもたらす可能性もある。本稿では，計算科学，計算工学において広く使用されている偏微分方程式の数値解法である，有限要素法から導出される疎行列を係数とする大規模線形方程式（連立一次方程式）を，前処理付き並列反復法（Preconditioned Parallel Iterative Methods）によって解く場合について検討する。本稿の手法は著者等の先行研究〔2〕に基づくものである。

2. 並列有限要素法アプリケーション「GeoFEM/Cube」

本稿で対象としているアプリケーションは，GeoFEM プロジェクト〔3，4〕で開発された並列有限要素法アプリケーションを元に整備した性能評価のためのベンチマークプログラム「GeoFEM／Cube」であり，本センターでは，スーパーコンピュータの調達時の性能評価試験に使用している。本ベンチマークは，均質場における三次元弾性静解析問題（Cube 型モデル（図2)）に有限要素法を適用することにより導かれる大規模疎行列を係数行列とする連立一次方程式を並列前処理付き反復法（クリロフ部分空間法〔5〕）によって解くものであり，反復法ソルバーの実行時性能を様々な条件下で計測する。要素タイプは三次元一次六面体要素（tri-linear）であり，各要素 8 つの節点を有している。各要素は辺長さ＝1 の単位立方体であり，x，y，z 各方向の節点数はそれぞれ，N_x，N_y，N_z である（従って要素数は，N_x-1，N_y-1，N_z-1)。

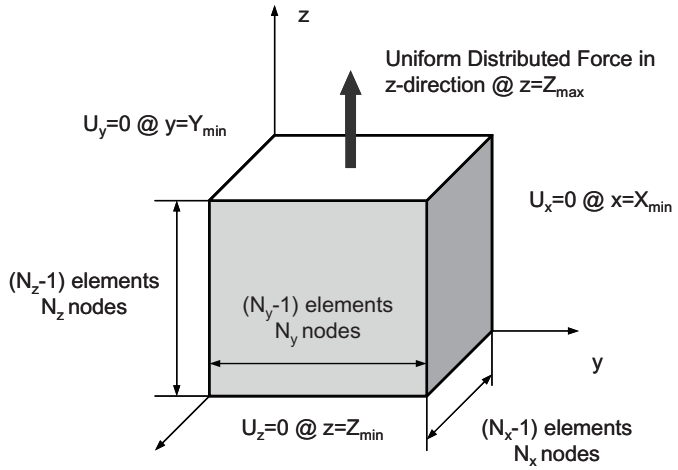


図 2　GeoFEM/Cube の解析対象〔2,3,4〕

三次元弾性問題では 1 節点あたり 3 つの自由度があるため，これらを 1 つのブロックとして取り扱っている（図 3)。係数行列はこのブロック型の特性を利用したブロック CRS 形式（Compressed Row Storage）によって格納されている。

有限要素法において（厳密にはガラーキン法に基づく有限要素法において)，三次元弾性静解析問題では係数行列が対称正定な疎行列となることから，前処理を施した共役勾配法（Conjugate Gradient，CG）法〔5〕（図 4）によって連立一次方程式を解いている。前処理手法としては，様々な手法が利用可能であるが，本稿では，図 3 に示す対角ブロックに対してフル LU 分解を適用する手法（Block Diagonal LU Factorization）〔2〕を適用している。この手法は，各節点単位で独立に前処理を適用できるため，並列化が容易であり，MPI による通信も発生しない。

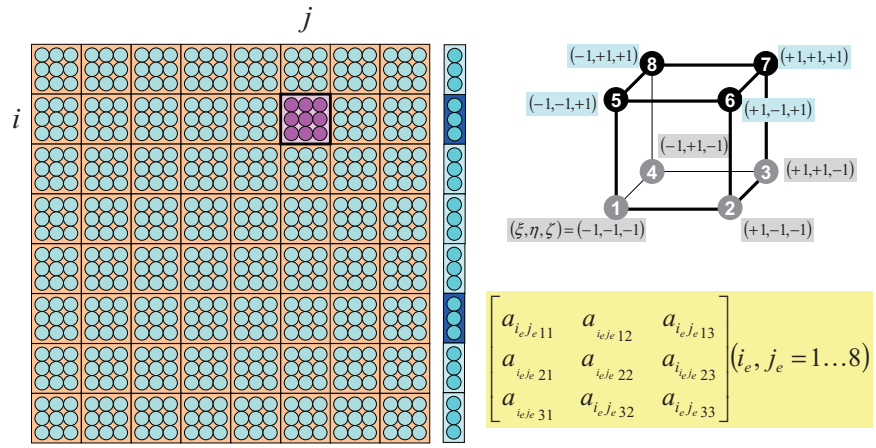


図 3　8 節点六面体要素，1 節点 3 自由度の処理〔2,3,4〕

有限要素法を並列化する場合には，オーバーラップ要素を有する節点ベースの分割手法〔2,3,4〕が広く使用されており，本稿でもそれに基づいている。有限要素法においては，大規模連立一次方程式の求解に計算時間の大部分が費やされることから，並列計算時には自由度が定義される節点（Node）が各プロセスで均等となるように領域分割を適用する。GeoFEM/Cube は規則正しい形状を扱うが，プログラム内に並列メッシュ生成機能を内蔵し，プロセス間負荷バランスを自動的に調整している。図 5 は，25 節点，16 要素（四角形）から構成される二次元解析対象を 4 分割した例である。PE#0～PE#3（PE=Processing Element）の 4 領域に分割され，各領域に MPI プロセスが割り当てられる。図 5（b）は各プロセスで扱う局所分散データを示している。

```
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(1)= 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 4　前処理付き共役勾配法のアルゴリズム〔5〕

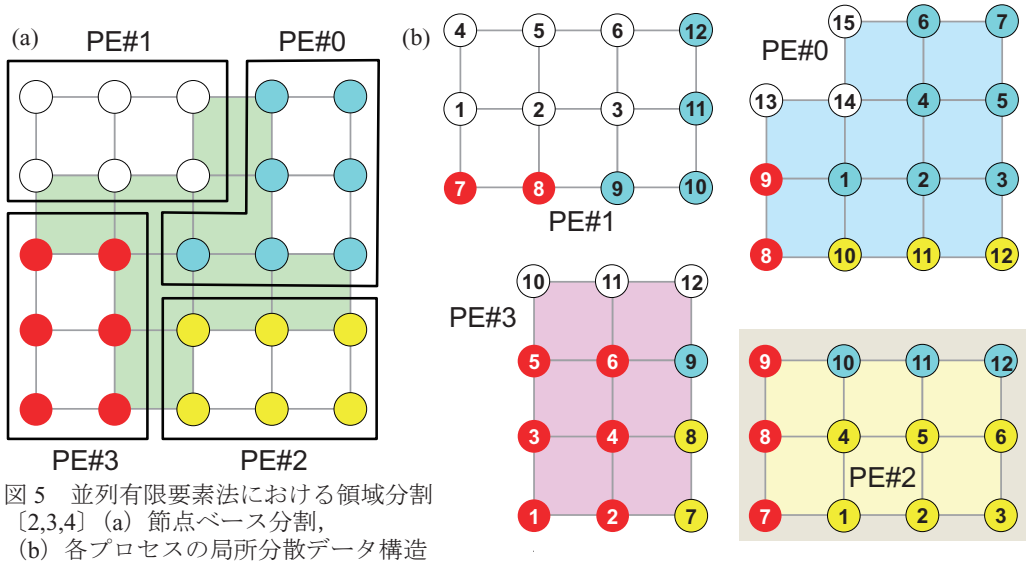


図 5　並列有限要素法における領域分割〔2,3,4〕（a）節点ベース分割，
（b）各プロセスの局所分散データ構造

図 5（b）に示す PE#2 は，図 5（a）に示すように本来黄色い節点が割り当てられているが，それ以外に，黄色い節点を含む要素，それに含まれる他の領域に割り当てられた節点（赤（PE#3），青（PE#1））が含まれている。並列有限要素法における局所分散データ構造においては，節点を以下の 3 種類のカテゴリに分類している〔2,3,4〕。

- 内点（Internal Nodes）：本来その領域に割り当てられた節点（図 5（a）PE#2 の黄色い節点）
- 外点（External Nodes）：本来異なる領域に割り当てられた節点
- 境界点（Boundary Nodes）：「内点」のうち，他の領域の「外点」である節点（外点に接する）

図 5 に示すようなオーバーラップ要素，外点を含む局所データ構造を適用することにより，要素マトリクス，全体マトリクスの生成を通信無しに実施することができ，また，共役勾配法のような反復法を並列化することも容易である〔2,3,4〕。**各局所分散データにおける局所節点番号は，図 5（b）に示す「内点⇒外点」のようにすると通信等が容易に実施できる。**なお，OpenMP/MPI ハイブリッド並列プログラミングモデルを適用する場合には，図 5（b）の各 MPI プロセスに割り当てられた局所分散データに対して OpenMP によるスレッド並列化を適用する〔2,3,4〕。

並列有限要素法で MPI プロセス間のグローバルな通信が発生するのは，本稿で扱うような線形，定常の問題の場合には，並列前処理付き反復法による線形方程式の求解部のみである。本稿では，Block Diagonal LU Factorization 前処理を適用しているため，図 4 に示すような前処理付き共役勾配法において，前処理適用部分（solve [M]$z^{(i-1)}=r^{(i-1)}$）は，各節点完全に独立に計算し，MPI プロセス間の通信は発生しない。したがって，MPI プロセス間通信が発生するのは，①内積，②疎行列ベクトル積（$q^{(i)}$= [A]$p^{(i)}$）のみである。②では，図 5（b）に示した局所分散データの特性を利用する。各 MPI プロセスで $q^{(i)}$= [A]$p^{(i)}$を計算するためには，外点における $p^{(i)}$の値を求めておく必要がある。そのためには MPI プロセス間通信によって，外点の値を他の MPI プロセスから予め受け取っておく必要がある。GeoFEM/Cube では，①境界点の値を MPI_Isend によって送信，②外点の値を MPI_Irecv によって受信，③MPI_Waitall によって通信を完了，という手順によって実施している〔2,3,4〕（図 6）。外点のことを「Halo」と呼ぶことがあり，図 6 に示すような通信を「Halo 通信」と呼ぶこともある。

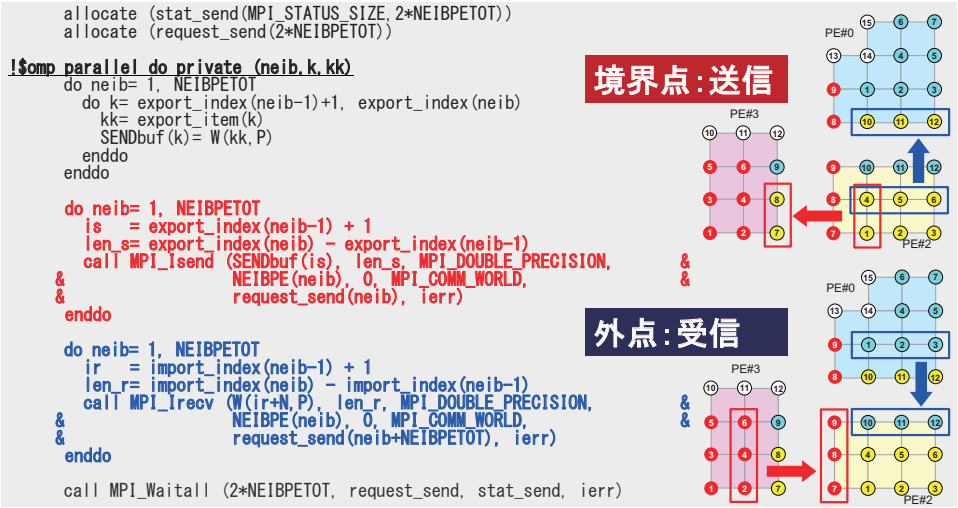


図 6　MPI_Isend/MPI_Recv/MPI_Waitall による境界点送信・外点受信（Halo 通信）〔2,3,4〕

3. 通信と計算のオーバーラップ

分散メモリ型並列計算機上で並列前処理付き反復法を実施する場合は，①疎行列ベクトル積，②内積，③前処理，で通信が発生する可能性がある。本稿における，例では前処理部分では通信は発生しない。疎行列ベクトル積では隣接プロセスとの 1 対 1 通信（Point-to-Point Communication），内積では全プロセスとの集合通信（Collective Communication）が発生する。MPI を使用する場合は，①は図 6 で示したような Halo 通信を適用し，MPI_Isend，MPI_Irecv を MPI_Waitall と組み合わせ，②においては MPI_Allreduce のような関数が使用される。

通信によるオーバーヘッドを削減するために，通信回避・削減型アルゴリズム（Communication Avoiding/Reducing Algorithm）の研究開発が盛んに進められている。パイプライン型共役勾配法（Pipelined CG Method）〔6〕は，漸化式の適用によって，図 4 に示したオリジナルの前処理付き CG 法のアルゴリズムが変わらないように計算の順序を変更する手法である。内積の直後に疎行列ベクトル積，前処理などの計算量が多く，かつ直前で実施した内積の結果を使わないような処理を実行するように計算順序が変更される。Pipelined 法では，これに MPI-3 でサポートされている MPI_Iallreduce 等の非同期集団通信（Asynchronous Collective Communication）を組み合わせることによって，集団通信と疎行列ベクトル積，前処理等の演算をオーバーラップすることができ，通信によるオーバーヘッドの隠蔽が可能となる〔6〕。

本稿では，特に疎行列ベクトル積部分の高速化について検討する。疎行列ベクトル積の計算を実施する場合には，図 6 に示すような Halo 通信によって，各 MPI プロセス間で通信を実施し，最新の p ベクトルの値を得たのち，各 MPI プロセスで内点に対して疎行列ベクトル積の演算を実施する〔2,3,4〕。内点のうち，境界点は外点に接しているため，外点の最新の値がないと疎行列ベクトル積を計算できないが，境界点以外の内点（純内点（Pure Internal Nodes））では通信の完了を待たずに計算を開始できる。これに基づき，図 6 に示す MPI プロセス間の Halo 通信と，「純内点」の計算を同時に実施する手法が，古典的な「通信と計算のオーバーラップ」〔7〕である（図 7）。図 8「Static」で計算手順の概要を紹介しているように，MPI_Isend，MPI_Irecv によって Halo 通信を起動させ，純内点の計算の後に MPI_Waitall を呼んで，通信と計算をオーバーラップさせている。純内点，境界点の計算を別々のループで実施していることから，図 7 に示すように，内点に reordering を施し，「純内点⇒境界点」となるように並び替えている。こうすることによって，各ループ内は分岐もなく，アクセスも連続となるため，計算効率が向上する〔2,8〕。

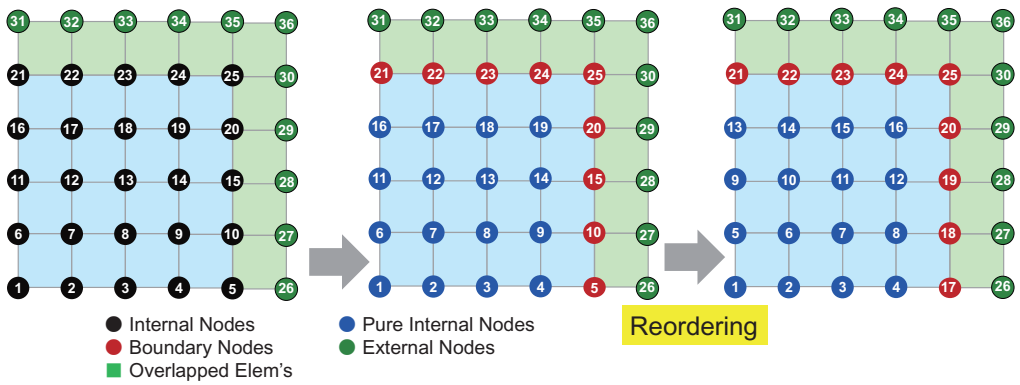


図 7　疎行列ベクトル積における通信と計算のオーバーラップ〔2,7,8〕

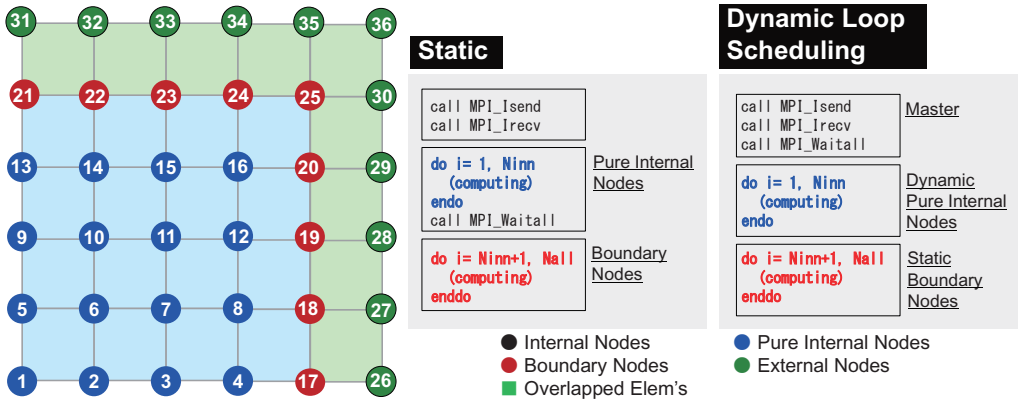


図 8　疎行列ベクトル積における計算と通信のオーバーラップ（Static：通信と純内点（Pure Internal Nodes）の計算をオーバーラップ，Dynamic Loop Scheduling：OpenMP のマスタースレッドが通信を担当する）

通信と計算のオーバーラップを更に効率化するために，Halo 通信に OpenMP の動的ループスケジューリング（Dynamic Loop Scheduling）機能を適用する手法が考案されている〔2，8〕。この手法では，Halo 通信を OpenMP のマスタースレッドが受け持ち，純内点の計算とオーバーラップさせる。図 8 の「Dynamic Loop Scheduling」がこれに相当している。

より詳細な実装を図 9 に示す。Halo 通信を OpenMP のマスタースレッドが受け持ち，残りのスレッドを使って純内点の計算を，Halo 通信とオーバーラップさせて実施している。「!$omp do schedule (dynamic,200)」とある「200」は「チャンクサイズ（Chunk Size）」である。これは，マスタースレッドが担当している通信が終了したかどうかをチェックするための間隔で，もし通信が終了していれば，マスタースレッドも純内点の計算に参加する。チャンクサイズが小さければ，それだけ細かい間隔で正確に通信の状況をチェックできるので，マスタースレッドが純内点の計算により早く参加でき，その分計算時間が短縮される可能性がある。ただ，その分，通信終了チェックのための回数が増加し，オーバーヘッドが大きくなる可能性もある。

```
!$omp parallel private (neib,j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
!$omp&         private (istart,inum,ii,ierr)

!$omp master              Communication is done by the master thread (#0)
!C
!C- Send & Recv.
(…)
      call MPI_WAITALL (2*NEIBPETOT, req1, sta1, ierr)
!$omp end master

!C                          The master thread can join computing of internal
!C-- Pure Internal Nodes    nodes after the completion of communication

!$omp do schedule (dynamic,200)    Chunk Size= 200
      do j= 1, Ninn
        (…)
      enddo
!C
!C-- Boundary Nodes          Computing for boundary nodes are by all threads

!$omp do                    default: !$omp do schedule (static)
      do j= Ninn+1, N
        (…)
      enddo

!$omp end parallel
```

図 9　動的ループスケジューリング（Dynamic Loop Scheduling）の疎行列ベクトル積への適用例

4. 計算事例

3.までに示した手法について，Wisteria/BDEC-01（Odyssey）の最大 6,144 ノードまでを使って評価を実施し，同じく本センターの Oakforest-PACS（OFP）〔9〕を最大 2,048 ノード使用した場合と結果を比較した。Oakforest-PACS（OFP）と Odyssey のノード単体諸元を表 1 に示す。本稿に示したケースでは，OFP については Flat モードを適用し，MCDRAM のみを使用した。OFP は各ノード 68 コアを搭載しているが，64 コアを使用した。

表 1　Oakforest-PACS（OFP）と Odyssey のノード単体諸元

システム名	Oakforet-PACS〔9〕	Wisteria/BDEC-01 (Odyssey)〔1〕
略称	OFP	Odyssey
CPU 名称	Intel Xeon Phi 7250 (Knights Landing, KNL)	Fujitsu A64FX（2.2GHz）
コア数	68	48
理論演算性能（GFLOPS）	3,046	3,379
主記憶容量（GB）	MCDRAM: 16 DDR4: 96	32
メモリ性能（GB/sec） STREAM Triad〔10〕	MCDRAM: 490+ DDR4: 84.5	840+
コンパイラ	Intel Parallel Studio 2019	Fujitsu FCC

性能評価は下記の 3 種類のプログラムについて，前処理付き CG 法一反復あたりの計算時間によって実施した：

① オリジナルの GeoFEM/Cube
② ①に通信と計算のオーバーラップ（図 8 の「Static」）を適用した例（Static）
③ ②に動的ループスケジューリング（図 8 の「Dynamic Loop Scheduling」）を適用した例，チャンクサイズを最大 1,000 まで変化させた

1 ノード当たりの問題規模を下記の 3 種類のケースに設定し，ノード数を変化させて Weak Scaling として実施した：

- S：96×96×48 節点（=1,327,104 DOF）
- M：128×128×64 節点（=3,145,728 DOF）
- L：200×200×100 節点（=12,000,000 DOF）

最大問題サイズは 7.3728×10^{10}DOF である。また並列プログラミングモデルとして：

- OFP：HB 8×8（各ノード，8 スレッドの MPI プロセス×8 プロセス），HB 16×4（16 スレッド×4 プロセス）
- Odyssey：HB 6×8（6 スレッド×8 プロセス），HB 12×4（12 スレッド×4 プロセス）

を実施した。図 10，図 11 は，OFP，Odyssey について，128 ノード，1,024 ノードにおけるオリジナルの GeoFEM/Cube に対する性能改善率である。100〜1,000 の数字は動的ループスケジューリングのチャンクサイズである。

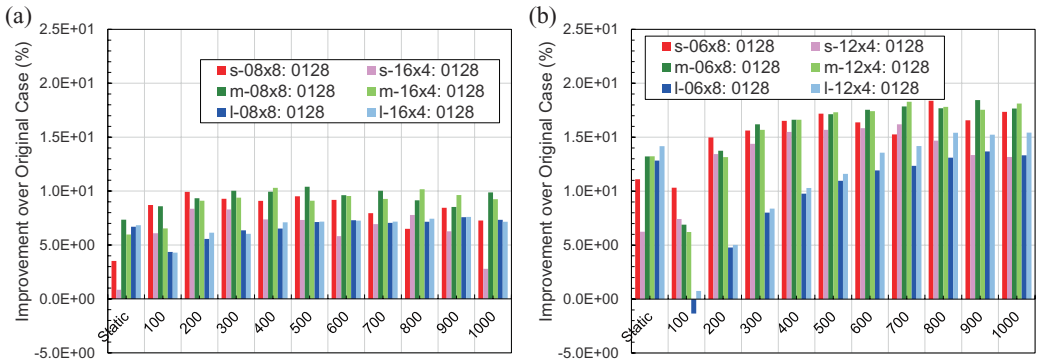


図 10　通信と計算のオーバーラップの効果：オリジナル GeoFEM/Cube に対する前処理付き反復法の速度向上率，Static：古典的オーバーラップ，数字：動的ループスケジューリングにおけるチャンクサイズ，128 ノード（a）OFP，（b）Odyssey

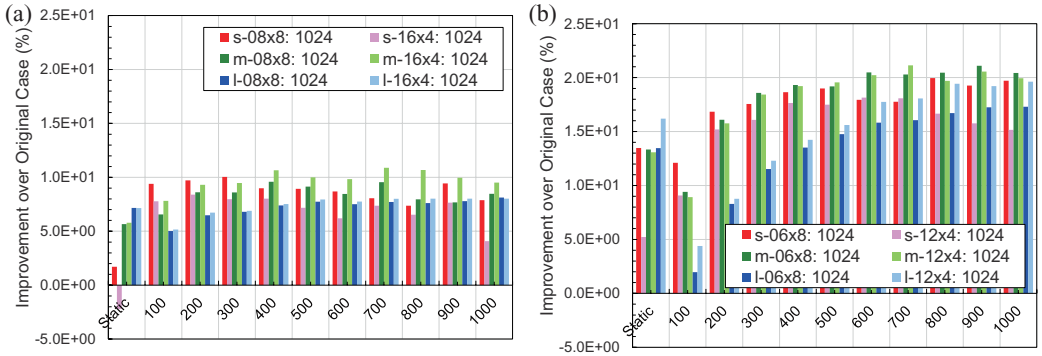


図 11　通信と計算のオーバーラップの効果：オリジナル GeoFEM/Cube に対する前処理付き反復法の速度向上率，Static：古典的オーバーラップ，数字：動的ループスケジューリングにおけるチャンクサイズ，1,024 ノード（a）OFP，（b）Odyssey

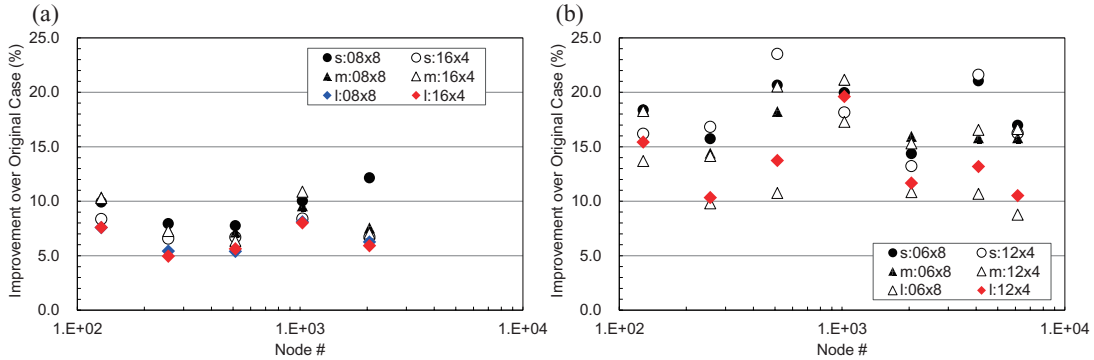


図 12　通信と計算のオーバーラップの効果：各ノード数におけるベストケースのオリジナル GeoFEM/Cube に対する前処理付き反復法の速度向上率，(a) OFP : HB 8×8 Original に対する比，最大 2,048 ノード（b）Odyssey：HB 6×8 Original に対する比，最大 6,144 ノード

OFP は最大 10%程度の速度向上率であるが，Odyssey は 20%程度である。OFP は 128 ノードと 1,024 ノードでは，やや 1,024 ノードの方が速度向上率が高いが，図 12 にも示すように，Odyssey において速度向上率とノード数の関係は必ずしも一様ではない。OFP と比較すると Static による速度向上は Odyssey においてより顕著である。また Odyssey においてチャンクサイズが小さい場合（700 以下）では，ノード当たり問題サイズが大きい場合の速度向上率が比較的低い。OFP，Odyssey ともにノード当たり問題サイズが小さい場合は，HB 8×8，HB 6×8 の性能が良く，大き

い場合は，HB 16×4，HB 12×4 の性能が良くなっている。

図 12 は OFP，Odyssey，各ノードにおけるオリジナルの HB 8×8（OFP），HB 6×8（Odyssey）に対するベストケースにおける速度向上率である。傾向は図 10，図 11 の場合と同じであるが，Odyssey についてはノード数と性能向上の関連が明かでなく，検討が必要である。

図 13 は，1,024 ノードにおける Odyssey（HB 6×8）と OFP（HB 8×8）の性能比である。ノード当たり問題規模が小さい場合，Odyssey は OFP の 2 倍以上，大きい場合でも 1.75 倍程度の計算性能であることがわかる。疎行列計算は memory-bound なプロセスであり，メモリ性能の影響が大きい。表 1 に示すように，Odyssey と OFP のメモリ性能比は STREAM Triad〔10〕で 1.71：1.00 であることを考慮すると，この性能比は妥当な数字である。

図 13 は M サイズ，128 ノード，HB 12×4 において，前処理付き共役勾配法部分の計算時間の内訳を Odyssey 上で詳細プロファイラ〔1〕を使用して測定したものである。動的ループスケジューリングによって，メモリスループットが 29.4%⇒37.8%と約 30%向上しており，計算時間から算定した速度向上率も 30%程度である。これは，メモリ性能の 40%程度しか活用できていない，ということでもあり，全体的な性能向上が必要である。

(a)　(b)

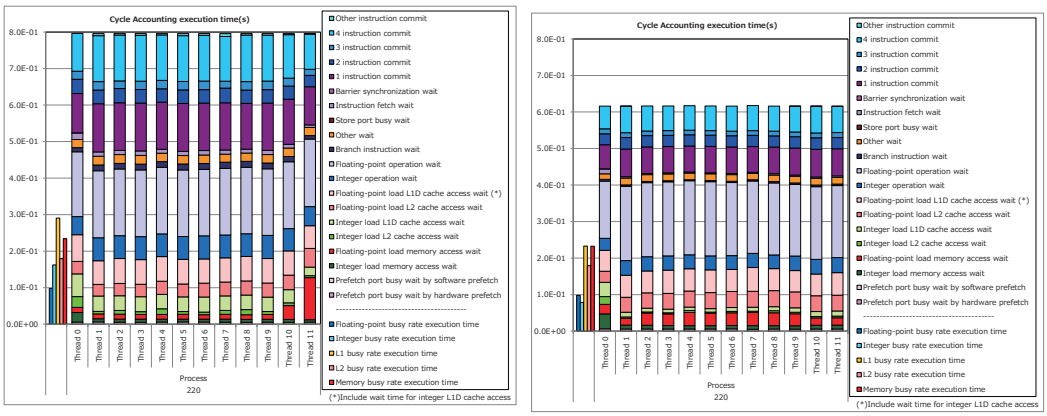


図 13　Odyssey 詳細プロファイラによる 0 番プロセス各スレッドにおける計算時間測定結果（前処理付き共役勾配法部分，サイズ M，128 ノード，HB 12×4），512 プロセスのうち＃220 プロセス，（a）オリジナル（メモリスループット：29.4%），（b）動的ループスケジューリング（チャンクサイズ＝700）（メモリスループット：37.8 %）

5. まとめ

Wisteria/BDEC-01（Oddysey）において，並列有限要素法から導かれる大規模線形方程式の，並列前処理付き反復法による求解プロセスのうち，疎行列ベクトル積演算部分に，通信と計算のオーバーラップ，動的ループスケジューリングを適用することによって，最大 20%以上の速度向上が得られることがわかった。今回は疎行列格納法が CRS であったため，性能は低く，メモリスループットは 40%未満であった。より高性能が期待される，ELL，SELL-C-σ等の手法を適用し，評価する必要がある。また，ノード数が 2,000 を超える場合については，ノード配置も含めた更なる詳細な検討が必要である。

参考文献

[1] Wisteria/BDEC-01: https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/system.php

[2] Nakajima, K., Hanawa, T., Communication-Computation Overlapping with Dynamic Loop Scheduling for Preconditioned Parallel Iterative Solvers on Multicore/Manycore Clusters, IEEE Proceedings of 10th International Workshop on Parallel Programming Models & Systems Software for High-End Computing (P2S2 2017) in conjunction with the 46th International Conference on Parallel Processing (ICPP 2017), Bristol, UK, 2017

[3] Nakajima, K. Okuda, H., Parallel Iterative Solvers for Unstructured Grids using an OpenMP/MPI Hybrid Programming Model for the GeoFEM Platform on SMP Cluster Architectures, Lecture Notes in Computer Science 2327, 437-448, 2002

[4] 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

[5] Saad, Y., Iterative Methods for Sparse Linear Systems (2nd Edition), SIAM, 2003

[6] Ghysels, P. and W. Vanroose, Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm, Parallel Computing 40-7, 224-238, 2014

[7] Shimokawabe, T., Aoki, T., Takaki, T., Yamanaka, A., Nukada, A., Endo, T., Maruyama, N., Matsuoka, S., Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer, ACM/IEEE Proceedings of SC'11, 2011

[8] 伊奈拓也，朝比 祐一，井戸村泰宏，テラフロップス級メニーコアアーキテクチャにおけるステンシル計算の最適化手法の開発，情報処理学会研究報告（2015-HPC-152-10），日本情報処理学会第 152 回 HPC 研究会，2015

[9] Oakforest-PACS: https://www.cc.u-tokyo.ac.jp/supercomputer/ofp/system.php

[10] STREAM: https://www.cs.virginia.edu/stream/

教育利用報告

「実践的シミュレーションソフトウェア開発演習」

居駒幹夫

東京大学大学院工学系研究科機械工学専攻非常勤講師

１．はじめに

大学院工学系研究科機械工学専攻向けの演習科目「実践的シミュレーションソフトウェア開発演習」の今年度実施内容および開発環境の動向について報告する。

本科目の目標は計算流体力学，分子動力学の本格的なシミュレーションソフトウェアを，信頼性，保守性，移植性などのソフトウェア工学の知識も考慮して開発できるスキルを身に付けることである。この目標達成のため，複数人の受講生がチームを組み，情報基盤センターのスーパーコンピュータや教育用のマシン環境を活用してシミュレーションソフトウェアを開発する。2009年に科目開始以来 13 年間継続，改善しており，受講者が保守性の高いシミュレーションソフトウェアを開発することを可能にしている。演習の内容，スケジュール等は本科目の教科書[1]および過去の報告 [1]を参照いただきたい。

２．今年度の実施報告

昨年度に引き続き、今年度も新型コロナウィルス感染拡大の防止のためリモート講義となった。使用した環境も、昨年度と同様に講義は Zoom を使用し、チーム開発でのチーム別の作業時間は、Zoom のブレイクアウトルーム機能を活用した。チャット環境として Slack を使い、チームごとに設定したチャンネルには講師も加わり、ソフトウェア開発中の質疑応答も Slack を活用して実施できるようにした。

本年度の科目登録者数は 14 名で、このうち 11 名が修了した。成果物は、これまでの平均程度だったが、昨年度と比べると完成度は低かった。昨年度のチーム数３から今年度は５に増えたことにより、Zoom 環境での各チームに対する講師側の指導が行き届かなかった点が反省事項である。受講生の評価としては、成果物としての満足度はあまり高くないが、科目としての満足度は高いという例年通りの結果であった。

３．2009 年からの開発環境の変遷

本科目は、2009 年のスタートから今年度まで継続しており、開発環境という観点でも多くの変更、改善を重ねている。12 年前というのは、「ほんのひと昔」という見方もできるが、コンピュータが誕生してから 100 年も経過していない現在において、「それなりに長い期間」ともいえる。実際、本科目演習の初年度の開発環境と現在の環境を比べてみると、ほぼ全部入れ替わったといっても過言ではない。本演習で使用している主要なマシン環境、開発ツールについて、その変遷を表１にまとめた。

[1] https://www.cc.u-tokyo.ac.jp/public/VOL14/No6/10_201211education-2.pdf

表 1　演習で活用している開発環境の変遷

開発環境		2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021現在
マシン	ターゲットスパコン	HA-8000-tc	→	→	Oakleaf-FX			→Reedbush	→	OakBridge-CX				**OakBridge-CX**
	開発マシン	演習室iMac	→	→	→	→	→	→	→	→	→	→	**OakBridge-CX**	
	ソース管理サーバー	自前サーバー	→	→	→	インターネットサービス						→	**インターネットのGitLab**	
開発ツール	バージョン管理	Subversion	→	→	→	→	→	Git	→	→	→	→	**Git+GitHubフロー**	
	チケット管理	Redmine	→	→	→	→	BitBucket		Trello	→	→	→	**Trello**	
	チャットツール	Twitter	→	→	→	→	→	Slack	→	→	→	→	**Slack**	
	作図ツール	PowerPoint	→	→	Astah	→	→	→	→	→	→	→	**Astah**	
	リモートエディタ	ssh+nano/vi/emac	→	→	→	→	→	→	→	→	Vscode		**Vscode**	
	ローカルエディタ	Eclipse		→	→	→	→	→	→	→	→	N/A		**N/A**
	知識管理	パワポテキスト		→	＋書籍出版			→	→	→	＋Wiki			**テキスト、書籍、Wiki**

このようにまとめてみると、12 年間に大きな節目が二つあることが分かる。一つ目は、自前の管理サーバーからインターネットベースの Web サービスに移行した 2016 年頃と、コロナ禍で、リモート講義が余儀なくされた 2020 年である。以下、ターゲットマシンと開発ツールに分けてこれまでの開発環境の変遷をまとめる。

ターゲットマシンであるスーパーコンピュータは、現在の OakBridge-CX で 4 代目である。これは教員側が用意しているソフトウェア資産のポータビリティの試金石にもなっており、ReedBush になってからは、ほぼ、ソフトウェア資産を変更せずに動作できるものとなっている。ソース管理や、チケット管理を実行する管理サーバーは、当初、生産技術研究所内の研究室に設置したサーバーを自前で管理していたが、2016 頃からインターネットベースのサービスを利用するように変更した。この変更により、サーバー管理に要する講師側の負荷が抜本的に低減し、受講者のビューでも毎年最新の機能が使用できるという利点を享受できるようになった。

開発ツールに目を向ける。まず、バージョン管理ツールのプレゼンスが大きく変わった。2009 年時点では、受講者の中にはこの類のツールを知っている人は皆無に近かったが、現在では、ツールの存在だけでなく、Git のベターユースまで経験済みという受講生が一定数いるようになった。このため、演習での使用方法も、単に開発者間でのソース管理だけではなく、開発者、保守者という役割を意識するような、「より実際的な Git の活用」を学べるように改善している。シェルやソースエディタは、コロナ禍の影響で演習室の統一された環境から受講者それぞれの環境に移行せざるをえなくなり、当初は、大問題だと考えられたが、Visual Source Code(Vscode)で、リモート編集、リモートシェルを使うことにより、従来よりも簡単かつ統一的にスーパーコンのエントリシステム上のソースコードを編集、アクセスできるようになった。

4．おわりに

今後も、ソフトウェア開発環境を継続的に改善し、受講生の学習目標達成および講師側の負荷低減を実現していく。

参 考 文 献

[1] 佐藤文俊, 加藤千幸編, “ソフトウェア開発入門: シミュレーションソフト設計理論からプロジェクト管理まで”, 東大出版, 2014 年 4 月, ISBN-4130624547

計算科学概論

（工学部物理工学科・理学部物理学科共通開講科目）

大 久 保 毅

東京大学大学院理学系研究科量子ソフトウェア寄付講座

本稿では，2021 年度 S セメスターに実施した，計算科学概論（工学部物理工学科・理学部物理学科 4 年生対象，月曜 3 限@オンライン講義）の内容について紹介する。東京大学では，計算物理学などの計算科学・工学から情報科学まで様々な学問領域の英知を結集した学際的研究教育プログラム「計算科学アライアンス[1]」が 2016 年 4 月より開始されているが，本講義は，計算科学アライアンスの認定講義として 2017 年度より開講されているものである。

本講義では，計算科学の様々な分野で行なわれている研究やシミュレーション手法の概要を複数の教員によるオムニバス形式で紹介することで，計算科学の現状を俯瞰し，最先端の研究と手法についての知識を得ることを目的としている。また，学生の理解を深める工夫として、座学と実習をセットで行うことで、計算科学の各手法について実際に手を動かして体験できる場を準備している点が一つの特色である。表 1 に講義日程と各回の担当者，内容を示している。計算科学の幅広い分野に加えて，計算機科学分野も扱い，計 7 テーマのオムニバス形式で実施した。座学 1 回と実習 1 回の計 2 回を基本とし、第 1 回のみ、内容から、座学 1 回のみとなっている。

昨年度に引き続き、新型コロナウイルスの感染拡大の影響により、座学・実習ともに Zoom を用いたオンライン講義となり、実習は基本的に学生所有の PC を用いて行った。また、本講義が最先端の計算科学の現状を知ることを目的としていることから、スーパーコンピュータの使用を体験することは重要であり、情報基盤センターの教育利用制度を利用してスーパーコンピュータ（Oakbridge-CX）を利用させて頂いた。ほとんどの学生にとって、スーパーコンピュータの利用は初めてであることが想定されるため、第 1 回で高性能計算機のアーキテクチャについて学んだのち、第 2、3 回で、スーパーコンピュータの特性や並列プログラミンの基礎の学習と、Oakbridge-CX の利用方法、及び、OpenMP, MPI プログラミングの実習を行った。第 4 回以降では、Oakbridge-CX も使用できることを前提として、各教員がテーマに沿った実習・座学を展開した。オンライン環境での実習では、テーマに応じて、Google Colaboratory や Amazon WorkSpace なども活用することで実習環境の統一と整備を行った。第 4，5 回では、並列化構造解析アプリケーションの利用、第 6，7 回では，量子多体問題を例に、大規模疎行列固有値問題の解法を扱い、第 8，9 回には、有限要素法を例とした大規模疎行列ソルバー、地震シミュレーションの紹介と有限要素法の SIMD 化を行った。続いて、第 10，13 回では、統計物理学に現れる格子スピン模型を例に、マルコフ連鎖モンテカルロ法とテンソルネットワーク形式による実空間繰り込み群を、第 11，12 回では、高性能プログラミングの実践として，ベクタ intrinsics 関数による演算の SIMD 化や，OpenMP によるマルチコア並列化を扱った。

講義の成績は，各テーマ担当の教員が，講義・実習内容に関連したレポート課題を設定し，学

[1] https://www.compsci-alliance.jp/

生は合計 7 つの課題のうちから 3 つを選んで提出することで評価した。また，4 つ以上の課題を提出した場合はその内容に応じて加点することとした。Oakbridge-CX のアカウントを取得し、毎回の講義に出席していた学生は 15 名程度であったが，3 つ以上のレポートを提出し，最終的に単位を取得した者は 4 名であった。今年度も新型コロナウイルス感染拡大の影響により、講義の大多数がオンラインとなったため、学生のレポート負担が大きかったことが想像される。これが、受講者数に比べて最終的な単位取得者が少数に留まってしまった原因の一つではと考えている。

本講義はオムニバスであるため、例年、担当教員間でのスケジュール調整が難しいが、本年度は比較的まとまり良く、前半に計算機ハードウェアとスーパーコンピュータ入門を行った後に、種々の計算科学の事例に入ることができ、また、講義と実習が連続して行えなかった例も 1 件のみに抑えることができた。そのため、参加者にとっては、講義内容が理解しやすい状況にできたと考えられる。一方で、昨年に引き続き、オンラインでの実施となったことで、特に実習において、実習スピードが異なる学生が混在している際の、細かなサポートが対面講義に比べて不足していた可能性もあり、今後、オンラインでの実習について、改善の検討も必要だと考えている。

本講義の座学と実習を通じて，計算科学・計算機科学の最先端の研究に関する知識を得るだけでなく，実際に手を動かして、アプリケーション利用やプログラミングの実践的な実習ができたことが，学生にとって貴重な経験になっていれば幸いである。

表 1　講義日程・担当者・内容

回数	日付	担当者	内容
1	04 月 05 日（月）	中村 宏 [1]	高性能計算機のアーキテクチャ
2	04 月 19 日（月）	吉本 芳英 [2]	スーパーコンピュータと並列プログラミング
3	04 月 26 日（月）		Oakbridge-CX，OpenMP，MPI プログラミング
4	05 月 17 日（月）	奥田 洋司 [3]	連続体の並列有限要素法解析入門
5	05 月 24 日（月）		構造解析アプリケーションによる CAE 実践
6	05 月 31 日（月）	山地 洋平 [4]	大規模疎行列固有値問題と量子多体問題
7	06 月 07 日（月）		1D Heisenberg model のランチョス法
8	06 月 14 日（月）	市村 強 [5]	大規模疎行列ソルバー入門
9	06 月 21 日（月）	藤田 航平 [5]	有限要素法シミュレーションの SIMD 化
10	06 月 28 日（月）	大久保 毅	格子スピン模型の計算科学
10	07 月 05 日（月）	田浦 健次朗 [6]	高性能プログラミングと性能測定
11	07 月 12 日（月）		Oakbridge-CX での SIMD，マルチコア並列化
13	07 月 19 日（月）	大久保 毅	モンテカルロ法，テンソルネットワーク法

1. 大学院情報理工学系研究科システム情報学専攻
2. 大学院情報理工学系研究科コンピュータ科学専攻
3. 大学院新領域創成科学研究科人間環境学専攻
4. 物質・材料研究機構エネルギー・環境材料研究拠点
5. 地震研究所計算地球科学研究センター
6. 大学院情報理工学系研究科電子情報学専攻

第 155 回お試しアカウント付き並列プログラミング講習会「OpenFOAM 入門」実施報告

今野 雅

東京大学情報基盤センター客員研究員

2021 年 6 月 1 日（火），PC クラスタコンソーシアム（実用アプリケーション部会・HPC オープンソースソフトウェア普及部会)、オープン CAE 学会との共催で，第 155 回お試しアカウント付き並列プログラミング講習会「OpenFOAM 入門」がオンラインで開催された。本講習会は，センターに設置されたスーパーコンピューター(以降，スパコン)の利用促進とスパコンを用いた数値流体解析の普及を目的として実施されたものである。なお，本講習会はセンターのお試しアカウント付き並列プログラミング講習会として行われた 19 回目の OpenFOAM の講習会である。受講者は，大学・研究機関教職 2 名，学部学生 2 名，大学院学生 4 名，企業の方 4 名であり，事前申込者 13 名,受講者合計は 12 名であった。センターが運営するスパコン Wisteria/BDEC-01 を用い，Wisteria/BDEC-01 の概要，利用方法，OpenFOAM の演習が 1 日終日の日程で行われた。当日のプログラムを表 1 に掲載する。なお，講習会終了後約 1 ヶ月有効なお試しアカウント(Wisteria-O，最大ノード数 12，最大実行時間 15 分)が受講者に与えられた。

表 1 講習会プログラム

【2021 年 6 月 1 日（火)】
10：00 - 12：00　概要説明
イントロダクション
Wisteria/BDEC-01 概要
OpenFOAM 概要
Wisteria/BDEC-01 の OpenFOAM ベンチマークテスト
Wisteria/BDEC-01 へのログイン
Wisteria/BDEC-01 での OpenFOAM のビルド
Odyssey における module による OpenFOAM の環境設定
13：00 - 18：00　キャビティ流れ演習
解析対象
解析ケース
blockMesh による格子生成
格子の可視化
解析条件の設定
ソルバ実行
解析結果の可視化
解析結果の検証
並列計算
演習課題
チュートリアルの実行
質疑応答

講習会終了後のアンケート集計結果(回答数 12)を表 2 に示すが，参加した満足度の平均は 5 点満点中，4.33 と高かった。また，参加者から表 3〜5 に示すご意見を頂いた。今後の講習会の参考にしたい。

表 2 アンケート集計結果

評点	講習会の時間		講習会の講義内容(プレゼン)		配布資料の内容		サンプルプログラム内容		参加した満足度	
1	短い	0	簡単	0	簡単	1	簡単	1	不満	0
2		0		1		1		2		1
3	適切	7	適切	7	適切	7	適切	6	普通	1
4		2		3		2		2		3
5	長い	3	難	1	難	1	難	1	満足	7
	平均	3.67	平均	3.33	平均	3.08	平均	3.00	平均	4.33

表 3 Zoom によるオンライン講習会で良かったこと(原文ママ)

- 画面共有があり分かりやすかった 質問のハードルが低かった
- 社内でできるので、よかったです。
- 遠方からでも参加できること
- 業務時間でも参加できた
- 別室にてサポートいただき、無事実習ができました。
- ブレイクアウトルームを活用し、サポートしていただいことが非常に良かった。

表 4 Zoom によるオンライン講習会で悪かったこと(原文ママ)

- 休憩時間などに参加者間での会話をできないこと
- 特になし
- 当社 PC は、マイクが使えないです。
- ありません。
- 特になし。

表 5 本講習会に対するご意見(原文ママ)

- いつもお世話になっております。貴重な機会を無償でご提供いただき大変助かっております。今回も分かりやすくご教授いただきありがとうございました。
- 大変、ありがとうございました。　継続して勉強させていただきます。
- 前回中級編からいきなり参加しましたが、今回の入門編を受講したことで OpenFOAM の概要がつかめたような気がします。ありがとうございました。
- 以前、初級を受講した際、MobaXturm を入れて設定もしていたのですが、すっかり忘れていました。いずれにしても事前の確認を怠り、ご迷惑をかけ申し訳ありませんでした。
- 大変参考になりました。説明もわかりやすく、今後の講習会にも是非ともご参加させていただきたいと思いました。誠にありがとうございました。

第156回お試しアカウント付き並列プログラミング講習会「Wisteria実践」実施報告

塙　敏博
東京大学情報基盤センター

2021年6月4日（金）の午後、第156回お試しアカウント付き並列プログラミング講習会「Wisteria実践」が開催されました。新型コロナウイルス感染症対策のためにZoomを用いたオンライン講習会として実施されました。

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

受講者は、学部学生：1名、大学院学生：5名、大学・研究機関教職員：9名、企業の方：3名、計18名の方にご参加いただきました。

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

当日のプログラムを、以下に掲載します。

- 6月4日（金）
  13：00 - 14：00　Wisteria/BDEC-01システム紹介
  14：15 - 16：00　Odysseyノード：A64FXにおけるOpenMP最適化、MPI+OpenMPハイブリッド並列、性能分析(講義＋演習)
  16：15 - 17：45　Aquariusノード：A100 GPUの利用、OpenACC, MPI並列化、性能分析（講義＋演習）

16名の参加者について、講習会に関するアンケートをご提出いただきました。主要な項目の集計結果を以下に示します。

プログラミング経験については、5年未満が4名、5〜10年が5名、10〜20年が2名、20年を超える方も5名いらっしゃいました。並列プログラミングについては、知識を前提にしていましたが、経験なしの方も3名いらっしゃいました。使用しているプログラミング言語については、Fortranが12名と多いですが、Python, CとC++が同数の8名（複数回答可）となり、Pythonのユーザが引き続き増えてきています。

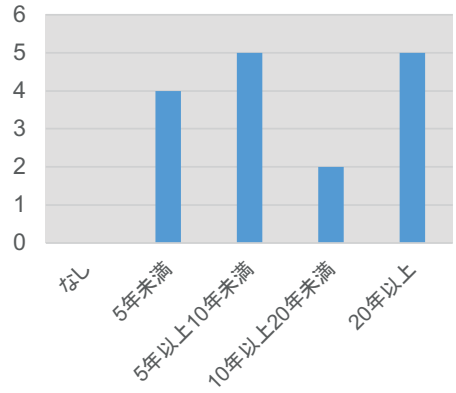


図 1 プログラミング経験

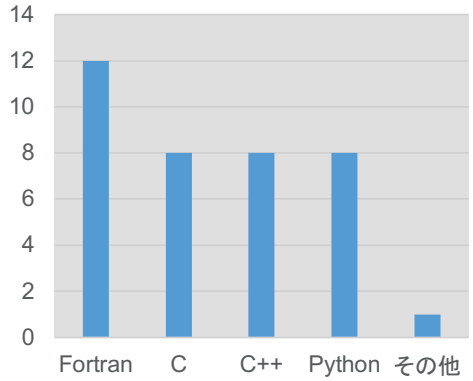


図 2 普段使用するプログラミング言語

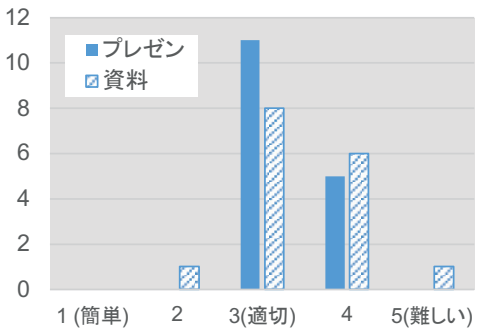


図 3 講習会の内容

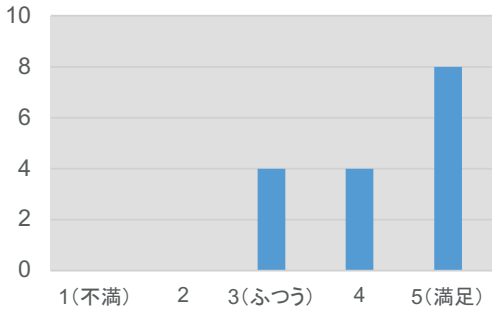


図 4 講習会参加の満足度

講習会の満足度は図 4 に示すようにおおむね満足度が高かったようです。平均値は 4.3 でした。ただ、図 3 に示すように、内容が難しいと感じた方もいらっしゃったようです。

今回は Zoom を用いての完全オンライン開催であったので、オンライン開催に関する回答をいただきました。オンライン開催で良かったことについての主な回答は

- 気軽に参加できる
- 予定がギリギリでも参加可能
- 浅野キャンパスで開催されるよりも参加しやすい
- じっくり考えながら、試していくことができて良かった（周りの雰囲気が気にならない）
- 往復の時間と交通費が節約できる（3 名）
- 持ち運びできない大きなディスプレイが利用できる
- 遠方に在住している者にとって、オンラインでの実施はとてもありがたいです。画面も非常に見やすいです。説明もわかりやすかったです。コロナ禍が済んだ後も、オンラインでの実施の継続を検討していただけると良いかと思います。

- 出張せずに、職場（或いは自宅）から簡単に参加できること。（2 名）
- 休憩の合間にも業務がこなせる点は教員にとてもありがたく，コロナ後も是非続けていただきたいです．この点は強く要望します．
- 遠く離れた場所から高品質な講義を受けることができた

一方悪かったことについては、

- あまり集中できない
- 一度わからなくなると大変そう
- Zoom の画面，PDF と，SSH を利用しているターミナルを同時に見比べるのが難しい
- 表示内容と、配付資料に齟齬があると、追いつきにくいと感じました。（これは、これまで、周りの雰囲気でフォローしていた部分。）
- スパコン実機を見学できないこと
- 受講者の進度確認がうまくいかないこと
- 演習スピードに付いていけないとき質問しづらいこと
- 特になし（7 名）

との意見をいただきました。遠隔地からの参加、使いやすい環境で講義を受けられることで、概ね好評だったようです。

また、以下のような感想をいただきました。

- 新しいスパコンの使い方がわかり、よかった
- 演習で進度がはやく追いつけませんでした
- Oakbridge-CX などの既存の大型計算機との相違点（コンパイラの特徴，同じアプリケーションを実行した時の比較など？）があるとうれしいです．
- これからも各種講習会を企画いただければと思います。大規模計算は情報基盤センターのスパコンが最適ですが中規模の計算などは研究室の PC クラスタ等でもできると思います。大規模スパコンの運用実績を活かして中規模 PC クラスタ構築法などのハードウエア構成法のノウハウ等の講習会もあると勉強したい方も多いと思います。本日はありがとうございました。
- どうもありがとうございました。Wisteria-o に、OSS をビルドする方法を学びたいです。
- 最新の情報について大変勉強になりました。どうもありがとうございました。
- 並列プログラミングのより実践的な内容があると助かります。
- 第 157 回講習会「GPU プログラミング入門」の申し込み時に本講習会の受講の推奨がありましたので、今回参加しました。東大情報基盤センターに新しいシステムが入ったので、その使い方、決まり事の説明が中心だと思っていました。本講習会での説明で、この趣旨の内容は十分理解できましたが、並列化処理についてかなり多くの説明がありましたので少し消化不良をおこしています。１か月無料で使用できますので、今後演習例題を実行し、新しいシステム、また並列化処理をマスターしようと思いま

す。

- プログラムの最適化に関する説明が少し駆け足だった印象です。

同様の講習会があれば、「また受けたい」という回答が 10 名、「どちらともいえない」が 5 名で、感想からもその他の講習会にも期待されていることが伺えます。

Oakforest-PACS においては「KNL 実践」講習会と「OFP 実践」講習会を合わせて 8 回開催し、好評を得ました。Wisteria/BDEC-01 についても同様に、今後も内容を見直しながら、「Wisteria 実践」を定期的に開催していきたいと考えております。

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

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

以上

第 158 回お試しアカウント付き並列プログラミング講習会

「第 3 回 GPU ミニキャンプ～HPC 編～」

下川辺 隆史

東京大学情報基盤センター

2021 年 6 月 22 日（火）、29 日（火）の 2 週にわたり、Zoom と Slack を用いてオンラインにて、第 158 回お試しアカウント付き並列プログラミング講習会「第 3 回 GPU ミニキャンプ～HPC 編～」が開催されました。

本講習会では、これから GPU を利用される方またはすでに利用されているが効率化を進めたい方、スパコンの GPU を利用したい方、を対象に、情報基盤センターに設置されたスーパーコンピュータ Wisteria/BDEC-01 を活用した実践を行いました。ミニキャンプは参加者がコードやデータセットを持ち込み、各自のペースで GPU 化や GPU 利用効率向上などを実践する形で進行します。情報基盤センター教員に加えて、GPU のスペシャリストがメンターとして参加し、受講者はコードの GPU 化や利用率向上の作業を進めるにあたり随時相談することができます。

HPC 編では、高性能計算（HPC）に焦点をあて、既存の CPU コードを CUDA、OpenACC、ライブラリで GPU 化したり、既存の単体 GPU コードを複数 GPU コードにすることなどに取り組みました。

本講習会は、東京大学情報基盤センター、エヌビディア合同会社、PC クラスタコンソーシアム（実用アプリケーション部会）の共催、プロメテック・ソフトウェア株式会社の後援で開催され、下記の皆様にメンターとしてご協力いただきました（敬称略）。

- 成瀬　彰：エヌビディア合同会社 シニアデベロッパーテクノロジーエンジニア
- 丹　愛彦：エヌビディア合同会社 HPC ソリューションアーキテクト
- 佐々木邦暢：エヌビディア合同会社 シニアソリューションアーキテクト
- 廣川祐太：プロメテック・ソフトウェア株式会社 HPC エバンジェリスト兼テクニカルサポート
- 大友広幸：東京工業大学 横田理央研究室 博士課程 2 年

本講習会のスケジュールは表 1 の通りです。本講習会では、オンラインで、まず座学を行い、その後はメンターと相談をしながら各チームで作業を進めていきました。本ミニキャンプでは、各チームでの実践時間を多く取るため、講習会 1 日目と 2 日目を 1 週間空けて開催しました。講習会開催日に挟まれた中日は原則各チームで実践し、ベストエフォートでメンターによる対応を行いました。図 1 は講習会最後に撮影した集合写真です。受講者には実習で使用した Wisteria/BDEC-01 を受講後 1 ヵ月間利用できるお試しアカウントが与えられます。

表１　スケジュール

日付	時間	内容
6 月 22 日	10:00 - 10:30	Wisteria/BDEC-01 使い方講座
	10:30 - 11:00	自己紹介と目標設定など
	11:00 - 16:50	実践（適宜自由に休憩）
	16:50 - 17:00	事務連絡・終了
6 月 23 日-28 日		各チームで実践
6 月 29 日	10:00 - 10:30	事務連絡・情報交換
	10:30 - 16:00	実施（適宜自由に休憩）
	16:00 - 16:50	実施内容の紹介
	16:50 - 17:00	事務連絡・終了

図１　講習会の集合写真

今回の講習会では、合計 11 チーム 26 名の事前申込があり、申込者全員が受講しました。受講者の内訳は、大学・研究機関教職員：14 名、大学院学生：8 名、学部学生：1 名、企業の方：3 名でした。講習会終了後に実施したアンケートの質問項目と回答の人数分布は表 2 の通りです。

オンライン開催について、下記の自由回答をいただきました。

オンラインの方が良いと回答した参加者の理由：

- 非同期と、同期をうまく混ぜてできてためです。参加する方々の状況によりかわるので、オンラインと現地とどちらが良いかは分かりません。少なくとも、オンラインでは、参加の機会が増えるので、私にとってはとても良かったです。

- 開催日以外の日にも質問ができるため。現地に集まる日があると、ほかの人の質問などが自然に耳に入ってきたりしてよいのかもしれません。
- 各自でテーマを持ち寄り、ほぼ個々人での作業となることから、現地に集まる必要はないため。Slack 上に様々な知識が集約され、非常に有効なテキストとして活用できるため（現地開催では、ここまで Slack は活用しないと予想）
- 移動の時間を節約できる。
- 現地だと遠隔地の人が困るため

現地開催の方が好ましいと回答した参加者の理由：

- 参加者同士の交流があるので。
- メンターにすぐに話しかけることができるとやはり嬉しいです。
- I find it easier to concentrate and focus when on site.
- 初日か最終日に現地で集まれるとメンターの方・参加者間で交流でき良いと感じました。中日は今回と同じオンライン形式が良いと思いました。
- オンラインでも進められるが、結局は同じ場所に集まり進めた方がより効率よく開発できると思うから。

Zoom と Slack の利用について、下記の自由回答をいただきました。

- slack の使い方の癖／マナーの一致が難しいのが、難しい気を予感しました。
- Slack でログを確認しながら他の方の質問も参考になりました。以外とオンラインも良かったです。
- オンラインの方が Slack での議論が活発で良かった。
- Slack は会社の PC だと許可されていなかった。
- Slack でのコミュニケーションは、スレッド化されており、アーカイブとして非常に有用でした。詳しいコメントも頂けて大変助かりました。Zoom でのコミュニケーションは、hands on で大変有用でした。
- スラックで質問ができるのは良かったです。ほかの人のやり取りもあとから見ることができて参考になりました。一方で、Zoom で質問することは、ハードルがやや高く感じてできなかったのがやや後悔しています。現地開催であれば、もうすこし気軽に聞けたかもしれません。
- オンラインではありますが、ブレイクアウトルームを使って個別の相談にのっていただけたので、深い議論もすることができたのが大変良かったです。
- slack で他の方の質問とその回答を見られるのは良かったです。
- Zoom、Slack ともに：他の受講者の問題解決の様子を共有させてもらえるため、「自身の気づいていなかった課題」や、「知らなかった機能」を知ることができた。Slack：過去のやり取りの内容を何度も見直せるため、勉強する上で非常に有用だった。
- なれていなかったこともあり，Slack に書き込むのが，最初ハードルが高かった。
- slack の質問に迅速にご対応いただけて、良かったと思います。
- Slack は記録として残り後から参照できることが有益でした。
- メンターの方々はそれぞれの業務が別であるにも関わらず非常に迅速かつ丁寧に返信していただきとてもありがたかったです。

下記の自由回答をいただきました。

- ぜひ、GPU 普及のために継続してください。
- slack channel をチームごとに立てて頂けると、自分のチームの問題点などがたどりやすくなりそうなので嬉しいです。メンターの方だけでなく参加者全体に可視な共有ディレクトリも用意して頂けると参加者同士でのインタラクションがやりやすくなりそうなので嬉しいです。
- プロファイリングをちゃんととったことがなかったのですが､プロファイリングをとって、律速になっている部分を高速化するという流れを習得できてとても良い経験になりました。ありがとうございました。
- GPU プログラミングもスパコン利用も未経験でしたが、非常にわかりやすい資料で、短期間で基礎的な部分を習得できた（今後ある程度は自学を進められるようになった）と感じています。ありがとうございました。
- 「GPU を社内に導入したい（あるいはスパコンを利用したい）が、社内提案できるほどの情報や技能を持ち合わせていない企業担当者」にとっては、今回のような講習会は非常に有用だと思うため、今後もぜひ継続していただきたいと思います。
- Thank you for organizing this nice workshop.
- 非常によい経験になりました。今後，もっと使っていこうと思います。
- どうもありがとうございました。

アンケート集計結果を見ると、多くの方に満足してもらえたようで、改善点を踏まえて、次回以降の開催を検討したいと考えております｡今回は､新型コロナウイルス感染症対策のため、前回に引き続きオンライン開催となりましたが、オンライン開催ならではのメリットも感じてもらえたようです。今後は状況をみながら、オンライン開催と現地開催の両方のメリットを生かした開催形態も考えていきたいと思います。

表２　アンケート集計結果の人数分布と平均

	あり			なし			
並列プログラミング経験	20			2			
GPU プログラミング経験	16			6			
	オンラインが良い		現地開催が良い		どちらでも良い		
オンラインと現地開催	5		5		12		
	評点	1	2	3	4	5	平均
講習会時間	短い <-> 長い		1	20	1		3.0
講習会講義内容（プレゼン）	簡単 <-> 難		5	15	2		2.9
配布資料内容	簡単 <-> 難		6	14	2		2.8
満足度	不満 <-> 満足				6	16	4.7

第 159 回お試しアカウント付き並列プログラミング講習会

「第 4 回 GPU ミニキャンプ～DL 編～」

下川辺 隆史

東京大学情報基盤センター

2021 年 6 月 23 日（水）、30 日（水）の 2 週にわたり、Zoom と Slack を用いてオンラインにて、第 159 回お試しアカウント付き並列プログラミング講習会「第 4 回 GPU ミニキャンプ～DL 編～」が開催されました。

本講習会では、これから GPU を利用される方またはすでに利用されているが効率化を進めたい方、スパコンの GPU を利用したい方、を対象に、情報基盤センターに設置されたスーパーコンピュータ Wisteria/BDEC-01 を活用した実践を行いました。ミニキャンプは参加者がコードやデータセットを持ち込み、各自のペースで GPU 化や GPU 利用効率向上などを実践する形で進行します。情報基盤センター教員に加えて、GPU のスペシャリストがメンターとして参加し、受講者はコードの GPU 化や利用率向上の作業を進めるにあたり随時相談することができます。

DL 編では、深層学習（Deep Learning）に焦点をあて、既存の PyTorch や TensorFlow コードを最新の GPU で高速化したり、複数 GPU による学習などに取り組みます。

本講習会は、東京大学情報基盤センター、エヌビディア合同会社、PC クラスタコンソーシアム（実用アプリケーション部会）の共催、株式会社ディー・エヌ・エー、株式会社ブレインパッドの後援で開催され、下記の皆様にメンターとしてご協力いただきました（敬称略）。

- 山崎和博：エヌビディア合同会社 ディープラーニングソリューションアーキテクト
- 阮　佩穎：エヌビディア合同会社 ディープラーニングソリューションアーキテクト
- 呉　先超：エヌビディア合同会社 シニアソリューションアーキテクト
- 佐々木邦暢：エヌビディア合同会社 シニアソリューションアーキテクト
- 藤原秀平：株式会社ディー・エヌ・エー ソフトウェアエンジニア / TensorFlow User Group
- 太田満久：株式会社ブレインパッド チーフデータテクノロジーオフィサー / TensorFlow User Group
- 大友広幸：東京工業大学 横田理央研究室 博士課程 2 年
- 星野　華：東京工業大学 横田理央研究室 修士課程 2 年

本講習会のスケジュールは表 1 の通りです。本講習会では、オンラインで、まず座学を行い、その後はメンターと相談をしながら各チームで作業を進めていきました。本ミニキャンプでは、各チームでの実践時間を多く取るため、講習会 1 日目と 2 日目を 1 週間空けて開催しました。講習会開催日に挟まれた中日は原則各チームで実践し、ベストエフォートでメンターによる対応を行いました。図 1 は講習会最後に撮影した集合写真です。受講者には実習で使用した Wisteria/BDEC-01 を受講後 1 ヵ月間利用できるお試しアカウントが与えられます。

表１　スケジュール

日付	時間	内容
6 月 23 日	10:00 - 10:30	Wisteria/BDEC-01 使い方講座
	10:30 - 11:00	自己紹介と目標設定など
	11:00 - 16:50	実践（適宜自由に休憩）
	16:50 - 17:00	事務連絡・終了
6 月 24 日-29 日		各チームで実践
6 月 30 日	10:00 - 10:30	事務連絡・情報交換
	10:30 - 16:00	実施（適宜自由に休憩）
	16:00 - 16:50	実施内容の紹介
	16:50 - 17:00	事務連絡・終了

図１　講習会の集合写真

今回の講習会では、合計 8 チーム 16 名の事前申込があり、申込者全員が受講しました。受講者の内訳は、大学・研究機関教職員：10 名、大学院学生：4 名、企業の方：2 名でした。講習会終了後に実施したアンケートの質問項目と回答の人数分布は表 2 の通りです。

オンライン開催について、下記の自由回答をいただきました。

オンラインの方が良いと回答した参加者の理由：

- オンラインの方がよく聞き取れ、質問もし易い様に感じているため。
- 今回、午後に授業があり一部参加できない時間ありましたが、オンラインであったため、授業前後に参加できました。
- 参加の機会が広くなるため。

- 出張が必要ないので、すごく便利になります。
- ブロックタイムが少ない
- 移動の時間を節約できる

現地開催の方が好ましいと回答した参加者の理由：

- もちろん自分が計算科学に疎いのが悪いのですが，slack でどう質問すれば十分な情報になるのかわからず，雑な質問を投げてしまったと感じました.
- 現地の方が質問しやすいと思いました。オンラインですと、自分の質問が全員に見えてしまうので少し恥ずかしかったです。多くの人からご助言を頂けるという利点もありますが。
- I find it easier to focus / concentrate on the workshop if I am attending in person.

Zoom と Slack の利用について、下記の自由回答をいただきました。

- どの方がメンターなのかわかりづらかったです。対応していただいている方がメンターの方なのか、参加者の方なのかを判断したかったです。
- Slack は会社環境では使えない場合もある。
- Slack に情報が残るのでよかったと思います。
- Slack 上のやり取りがアーカイブ化され、色々と参考になりかなり有益でした。Zoom では、hands on 的なセッションをして頂き効率的にやりたかったことをご教示頂けよかったです。
- リラックス の方式で質問ができます。イベント後でも、Slack による質疑応答のプロセスを確認できます。
- slack は記録が残って後から参照できることが良い。

下記の自由回答をいただきました。

- 講習会が始まってからスパコンの動作環境を作るのに時間がとられてしまうため、余裕を持ってログインアカウントを通知して欲しい。また同様に、講習会前にプログラムを実行できるようにしていただけると助かります。
- ぜひ続けてください。
- 今回、初めて参加したので勝手がわからず、様子を眺めているだけで終わりました。またもし機会があればメンターの方々にいろいろご相談にのっていただけたらと思いました。
- DL では、簡単なモデル+データセットの minimum working example 的なものがあるとありがたかったです。何か新しいものを試す際にとりあえず動くものから出発出来ると上手くいかなかった時に立ち戻る原点となるので。
- 貴重な機会いただき、どうもありがとうございました。″
- 日時が指定されていたのですべて参加することは出来なかったが、チームメンバーと情報交換することにより良い知見が得られた
- 通常は MPI を使った並列計算をしておりますが，今回集中的に GPU を使った並列計算をする機会ができ，理解がかなり進んだと思います。また，プロの方の助言が比較的気楽に聴けることがよかったです。

アンケート集計結果を見ると、多くの方に満足してもらえたようで、改善点を踏まえて、次回以降の開催を検討したいと考えております。今回は、新型コロナウイルス感染症対策のため、

前々回、前回に引き続きオンライン開催となりましたが、オンライン開催のメリットも確認できました。一方で、サンプルの準備状況など、いくつか課題もありましたので、これらを解決していきたいと思います。今後は状況をみながら、オンライン開催と現地開催の両方のメリットを生かした開催形態も考えていきたいと思います。

表２　アンケート集計結果の人数分布と平均

	あり			なし			
並列プログラミング経験	11			3			
GPU プログラミング経験	7			7			
	オンラインが良い	現地開催が良い		どちらでも良い			
オンラインと現地開催	6	3		5			
	評点	1	2	3	4	5	平均
講習会時間	短い <-> 長い			14			3.0
講習会講義内容（プレゼン）	簡単 <-> 難		1	11	2		3.1
配布資料内容	簡単 <-> 難		2	11	1		2.9
満足度	不満 <-> 満足			5	3	6	4.1

第 160 回お試しアカウント付き並列プログラミング講習会
一日速習：有限要素法プログラミング徹底入門（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 7 月 26 日（月）にオンライン開催した第 160 回お試しアカウント付き並列プログラミング講習会「一日速習：有限要素法プログラミング徹底入門[1]」（共催：東京大学情報基盤センター、PC クラスタコンソーシアム（実用アプリケーション部会・HPC オープンソースソフトウェア普及部会））について紹介する。

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

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

有限要素法の理論，手法は大学教養程度の数学，物理の知識があれば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　日に同様の内容で第 145 回講習会[4]を実施し，今回はそれらののフィードバックから得られた知見も踏まえ，更に改良した詳細な教材を提供した。

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

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

[1] https://www.cc.u-tokyo.ac.jp/events/lectures/160/
[2] https://www.cc.u-tokyo.ac.jp/events/lectures/116
[3] https://www.cc.u-tokyo.ac.jp/events/lectures/134/
[4] https://www.cc.u-tokyo.ac.jp/events/lectures/145/
[5] http://nkl.cc.u-tokyo.ac.jp/20w
[6] http://nkl.cc.u-tokyo.ac.jp/FEM/01-FEMintro.pdf

ム（Fortran・C）をダウンロードしてもらい，数学的・理論的な背景から，実問題（定常熱伝導）まで，実習も交え，大規模連立一次方程式の反復解法（前処理付き共役勾配法）も含めて解説した。詳細はホームページ（https://www.cc.u-tokyo.ac.jp/events/lectures/160/）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 16 名，出席者は 11 名（学生：3 名，大学・研究機関：2 名，企業：6 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：9）。表 2 は質問項目と回答（5 段階評価）の人数分布である。全体的な満足度の平均値は 5 点満点で 4.44 と第 134 回（4.32）よりは高かったものの，第 145 回（4.57）と比較して低くなった。講義や教材の難易度については「4：やや難しい」が多かったことも関係しているようだ。アンケートの自由記述欄には：

- 1 日で実施する内容としては分量が多いかもしれないと感じた
- プログラミングの解説があり、実際に有限要素法がどのように使われているのか良くわかりました。ありがとうございました。
- 懇切丁寧な内容でした。ここまで親切な内容の講習会はなかなかないと思います。
- 全て満足です。特に実装・プログラミングの観点での講義。計算力学の基礎面はある程度ですが知識があったので，少なくとも私にはうってつけの講義でした。

というようなコメントが記載されていた。

今後も「オンライン講習会」を中心に考えていく必要があり，本センターとしても講習会，プログラミング教育の今後のあり方を継続して検討していく予定である

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			7	2	
(b) 講習会講義内容（プレゼン）	簡単⇔難		1	1	6	1
(c) 配布資料内容	簡単⇔難		1	3	5	
(d) サンプルプログラム内容	簡単⇔難		1	4	4	
(e) 満足度（平均 4.44）	不満⇔満足			1	3	5

科学技術計算Ⅰ／計算科学アライアンス特別講義Ⅰ／スレッド並列コンピューティング　「科学技術計算のためのマルチコアプログラミング入門」（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 S1・S2 学期に実施した，科学技術計算 I（大学院情報理工学系研究科数理情報学専攻）／計算科学アライアンス特別講義 I（同 コンピュータ科学専攻）／スレッド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）「科学技術計算のためのマルチコアプログラミング入門」[1]について紹介する。

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。本講義では，「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（大規模超並列スーパーコンピュータシステム（Oakbridge-CX，OBCX）[2]）を使用した実習を実施した。

講義内容の詳細については，ウェブページから資料をダウンロードできるのでそちらを参照いただきたい。本講義では，受講者の多様なバックグラウンドを考慮して，ほぼ全講義内容について Fortran，C 両方による教材を準備している。

本年度は昨年に引き続き「新型コロナウイルス感染症」のため，全ての講義を Zoom によるオンラインで実施した。本講義は 4 月 7 日に開講したが，受講者の環境準備状況等を考慮して，4 月 7 日と 14 日は同じ講義を実施した（図 1）。

登録者は 56 名（科学技術計算Ⅰ：14 名，計算科学アライアンス特別講義Ⅰ：18 名，スレッド並列コンピューティング：24 名）で，昨年度（54 名）よりやや増加した。出席者は 35 名～40 名程度である。毎回の講義は録画して，クラウド上の動画のありかを ITC-LMS 経由で受講者に連絡しており，これを利用して復習している学生もいた様子である。

2018～2020 年度はやや難しいプログラミング（Sequential Reordering の実装と評価）をレポート課題としていたが，本年度は First Touch Data Placement に関するやや容易な課題としたため，単位取得者は 13 名から 21 名に増加した（参考：7 名（2018 年），10 名（2019 年））

2013 年度以降，資料は英語版のみ用意していたが，講義そのものは日本語で実施していた。2017 年度から英語で実施することとしたため，留学生の受講は増加しており，2021 年度は登録者の半数をやや上回る 56 名中 31 名であった。

[1] http://nkl.cc.u-tokyo.ac.jp/21s/
[2] https://www.cc.u-tokyo.ac.jp/en/supercomputer/obcx/system.php

Date	ID	Title
Apr-07 (W)	CS-01a	Introduction-a
Apr-14 (W)	CS-01b	Introduction-b (Introduction-a and –b are same)
Apr-21 (W)	CS-02	FVM (1/3)
Apr-28 (W)	CS-03	FVM (2/3)
May-05 (W)	**(no class)**	**(National Holiday)**
May-12 (W)	CS-04	FVM (3/3) , OpenMP (1/3)
May-19 (W)	CS-05	OpenMP (2/3), Login to OBCX
May-26 (W)	CS-06	OpenMP (3/3)
Jun-02 (W)	CS-07	Reordering (1/3)
Jun-09 (W)	CS-08	Reordering (2/3)
Jun-16 (W)	CS-09	Reordering (3/3)
Jun-23 (W)	CS-10	Tuning
Jun-30 (W)	CS-11	Parallel Code by OpenMP (1/3)
Jul-07 (W)	CS-12	Parallel Code by OpenMP (2/3)
Jul-14 (W)	CS-13	Parallel Code by OpenMP (3/3), Q/A

図 1　講義スケジュール

原 稿 募 集

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

執 筆 要 項

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

【スーパーコンピュータシステム利用案内】

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

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

【IP ネットワーク経由時のホスト名】

シ ス テ ム	ホ ス ト 名
Wisteria/BDEC-01 スーパーコンピュータシステム (Wisteria-O/A)	wisteria.cc.u-tokyo.ac.jp 以下のホストの何れかに接続します※ wisteria0{1-6}.cc.u-tokyo.ac.jp
Oakbridge-CX スーパーコンピュータシステム	obcx.cc.u-tokyo.ac.jp 以下のホストの何れかに接続します※ obcx0{1-6}.cc.u-tokyo.ac.jp
Oakforest-PACS スーパーコンピュータシステム	ofp.jcahpc.jp 以下のホストの何れかに接続します※ ofp0{1-6}.jcahpc.jp
Reedbush スーパーコンピュータシステム (Reedbush-H/L)	reedbush.cc.u-tokyo.ac.jp 以下のホストの何れかに接続します※ reedbush-{u{1-4},h1}.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.23 No.5(2021.9)

目　　次