

TSUBAME4.0 の処理量担保のための計算ノード分割

野村 哲弘, 遠藤 敏夫

東京科学大学 情報基盤センター

Compute Node Partitioning to Ensure Job Throughput in TSUBAME4.0

Akihiro Nomura, Toshio Endo

Center for Information Infrastructure, Institute of Science Tokyo

概要

東京科学大学 (旧: 東京工業大学) のスーパーコンピュータ TSUBAME シリーズにおいては, 1 台の計算ノードに複数の GPU が搭載されているため, 全てのユーザが計算ノードの能力を十全に扱うことが困難であったため, 計算ノードを仮想的に分割してユーザのニーズに合う形の計算ノードを提供することに努めてきた. 計算ノードの仮想分割は, 限られた台数の計算ノードをより多くのユーザが利用できるようにする点においてもその必要性が年々増している. 本稿では TSUBAME4.0 において実施した計算リソース分割について報告する.

1 はじめに

東京科学大学情報基盤センターおよび前身の東京工業大学学術国際情報センター (以下, 「本センター」という) では, 「みんなのスパコン」を合言葉とし, 使いやすさと高性能を両立したスーパーコンピュータ TSUBAME シリーズを構築・運用しており, 2024 年 4 月からは現行の TSUBAME4.0 [5] を供用している. TSUBAME シリーズは 2006 年に導入された TSUBAME1.0 以来, 学内外で 1,300 名 (2023 年度 TSUBAME3.0 アクティブユーザ数) を超えるユーザに利用されている.

TSUBAME シリーズでは 2008 年に TSUBAME1.2 として NVIDIA Tesla S1070 を既存の TSUBAME1.0 計算ノードに後付けして以来, GPU を積極的に活用したスパコンとして知られ, 2010 年の TSUBAME2.0 では NVIDIA Tesla M2050(2013 年に NVIDIA Tesla K20X に交換 (TSUBAME2.5)) をノードあたり 3 基, 2017 年の TSUBAME3.0 では NVIDIA Tesla P100 をノードあたり 4 基, 2024 年の TSUBAME4.0 では NVIDIA H100 SXM5 HBM2e 94GB をノードあたり 4 基備え, いずれの世代においても (メモリ増強ノードなどの例外を除いて) 複数 GPU を備える単一構成の計算ノードを備えてきた.

TSUBAME シリーズにおける演算能力の大きな割合が GPU によることは明らかではあるが, TSUB-

AME の全てのユーザが計算ノードに備えられた複数の GPU を活用できるわけではない. また, TSUBAME シリーズでは代を経るごとに計算ノードの台数が半数以下へと減少しており, そのままではユーザに提供可能な資源量 (同時実行可能ジョブ数) も同様に激減せざるを得ない状況にあった. そのため, GPU を必要としていないユーザ等にむけて, その時代にそれぞれ利用可能であった技術を活用し, 計算ノードを仮想的に分割することで, それぞれのユーザに必要な十分な資源を見せるとともに, それらのジョブを 1 台の物理計算ノードに集約することによって, 遊休資源の最小化とともに, 計算ノードの台数以上の計算要求を満たすことに努めてきた.

2 TSUBAME4.0 の計算ノード概要

TSUBAME4.0 [7] の 240 台の計算ノードは, 2 台の AMD EPYC 9654(Genoa, 物理 96 コア) と 4 台の NVIDIA H100 SXM5 HBM2e 94GB を持ち, 768GiB のメインメモリを備えたノードである (図 1). TSUBAME4.0 の計算ノードは 240 台と, 前世代 TSUBAME3.0 の 540 台の半分弱, 前々世代の TSUBAME2.0/2.5 の 1408 台と比較すると 2 割弱のノード数しかなく, 増加するユーザ数・ジョブ数をさばくには, 計算ノードの分割を行って見せかけのノード数を確保する必要があった.

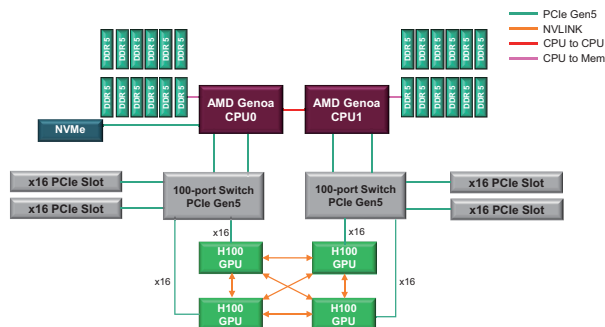


図1 TSUBAME4.0 計算ノードのブロックダイアグラム

3 Linux cgroup および NVIDIA MIG によるノード論理分割

TSUBAME4.0 のリソース分割の基本となった TSUBAME3.0 におけるリソース分割の設計時には以下のような要件を念頭に検討を行った。

- GPU や相互結合網がリソース分割機構のせいで遅くならない
- リソース分割しても今までできていたことができる
 - 計算ノードへの直接ログインによる対話的操作
- 計算ノードを共有する各ジョブが互いに相手を見ることができない
- 計算ノードの分割方法を動的に変更できる
- ジョブに応じて最適なユーザランドを選択できる (セキュリティパッチを容易に適用できる)

上述の要件のうち、最後の点についてはリソース分割とは直接は関係しないが、TSUBAME2.0/2.5 におけるリソース分割の実装が VM であったことから、リソース分割と実行環境仮想化の両方の概念を「仮想化」の語のもとに混同しており、あわせて要件として加えられたものである。

これらの要件と、導入当時の技術トレンドから、TSUBAME3.0 においては Docker を活用したリソース分割を行うこととした。しかしながら、Docker は上記要件の最後の点を除いたリソース分割の観点では本質的には Linux cgroup でしかなく、スケジューラである Univa(現 Altair) Grid Engine(以下、Altair のものも含めて AGE) [1] への組み込みの際には、cgroup による資源分割と Docker による実行環境仮想化が独立して実装された。

TSUBAME3.0 の時点では、CPU・メインメモリ・

SSD の論理分割および GPU のアクセス可否を cgroup で制御し、計算ノードを 1/4 までの比例分割と、1/4 ノードの CPU 部・GPU 部への分割を、スケジューラの設定を変えることなく計算需要に応じて動的に行うことを実現した。TSUBAME3.0 の計算ノードの分割パターンの模式図を図 2 に示す。実装には AGE の RSMAP 機能を用いて、CPU コア番号と GPU の対応付けを定義し、表 1 に示す資源タイプに沿った形で計算ノードが切り出されるように設計した。

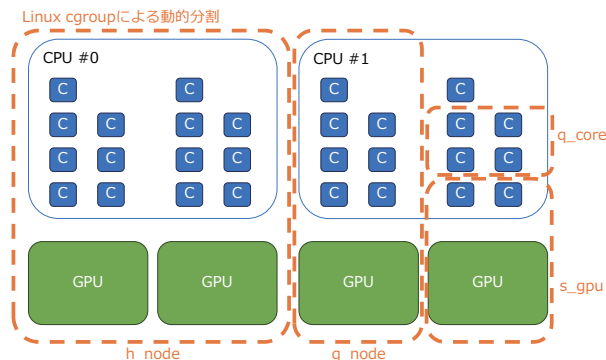


図2 TSUBAME3.0 計算ノードの資源分割イメージ

表1 TSUBAME3.0 における資源タイプ一覧

	資源タイプ	CPU コア数	GPU 台数	メモリ [GiB]
F	f_node	28	4	235
H	h_node	14	2	120
Q	q_node	7	1	60
GQ	s_gpu	2	1	15
CQ	q_core	4	0	30
CX	s_core	1	0	7.5

表の各行の記号は図 4,5 で用いる略号

TSUBAME4.0 では、TSUBAME3.0 におけるコンセプトを継承しつつも、CPU のコア数および GPU1 台の能力が格段に増加したことをうけて、リソースの分割単位を再検討した。TSUBAME4.0 に搭載された H100 では TSUBAME3.0 の P100 と違い、NVIDIA Multi-Instance GPU(MIG) [4] と呼ばれる GPU の仮想分割機構が導入されたため、GPU を仮想分割して、別ユーザのジョブに割り当てることを検討した。

MIG では GPU を最大 7 分割することができるものの、分割しない場合と比較して 1 割程度の CUDA コアが利用できなくなるオーバーヘッドがあることと、GPU の一部でプログラムを実行中に GPU の他の部分の MIG 構成を変更することができるかが定かではなかったことから、MIG を利用しない状態と、MIG で GPU を 2 等分した状態のみを利用することとした。

また、AMD EPYC 9654 のコアは Chiplet と呼ばれるそれぞれ 8 コアの 12 個の塊に分割されており、チップレットを跨ぐプロセス配置を行った場合に、キャッシュアクセスなどの効率が低下するため、できるだけチップレット境界を跨がないような分割とすることが必要であった。

加えて、TSUBAME3.0 では CPU のみを利用するユーザが利用できる最大の資源タイプが `q_core` でメモリが 30GiB しか割り当てられておらず、メモリ量を必要とするタスクを実行するために、`f_node` などの GPU を含む資源タイプ指定を行い、GPU を余らせる結果となっている事例が観測およびユーザへのヒアリングで判明していた。

これらを踏まえて、図 3 に示すとおりノード分割を行った。各資源タイプの資源割当量は表 2 に示す通りである。

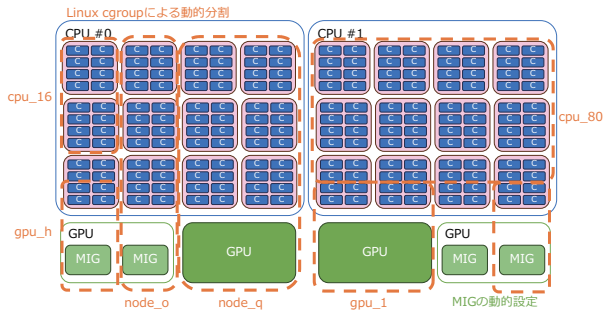


図 3 TSUBAME4.0 計算ノードの資源分割イメージ

表 2 TSUBAME4.0 における資源タイプ一覧

	資源タイプ	CPU コア数	GPU 台数	メモリ [GiB]
F	<code>node_f</code>	192	4	768
H	<code>node_h</code>	96	2	384
Q	<code>node_q</code>	48	1	192
O	<code>node_o</code>	24	1/2	96
GQ	<code>gpu_1</code>	8	1	96
GH	<code>gpu_h</code>	4	1/2	48
CF	<code>cpu_160</code>	160	0	368
CH	<code>cpu_80</code>	80	0	184
CQ	<code>cpu_40</code>	40	0	92
CO	<code>cpu_16</code>	16	0	36.8
CX	<code>cpu_8</code>	8	0	18.4
CY	<code>cpu_4</code>	4	0	9.2

表の各行の記号は図 4,5 で用いる略号

TSUBAME3.0 との大きな差異は、MIG による 1/2 GPU が割り当てられる資源タイプを追加した点および、CPU 資源タイプにおいて、`node_q` の境界を越えて 80 コアおよび 160 コアを利用する資源タイプが追加された点である。スケジューラは TSUBAME3.0 に引

き続き Altair Grid Engine(AGE) を使い、RSMAP による実装も流用している。後者の変更により、TSUB-AME3.0 と違い単純な階層的分割とはならず、リソース割り当ての複雑度が増加したが、現時点で特段の問題にはなっていない。

MIG については、計算ノードがアイドル状態のときには各 GPU の MIG を有効にして GPU を 2 分割した状態で待機する。この状態で `gpu_1` や `node_q` などの GPU を分割しないジョブが割り当てられたタイミングでジョブのプロローグ処理において MIG の無効化を、ジョブのエピローグ処理で MIG の再有効化を行う。これらの処理は `cgroups` によって対象の GPU のみにアクセスできるよう制限された状態で実行するため、ノード内の他の GPU に影響を与えることなく実行できる。

資源タイプは大別して 3 種類に分類でき、それぞれ 1) CPU・GPU をともに利用するグループ、2) 主に GPU で計算を行うグループ、3) CPU のみを利用するグループと特徴づけられる。このように資源タイプをお仕着せで定義せずに、ユーザに必要な CPU コア数、GPU 台数、メモリ容量をそれぞれ宣言させて `cgroup` を作る方式も検討したが、たとえばメモリの要求量が CPU・GPU に対して過剰となり、ジョブを割り当てられない CPU・GPU が出てくるなどの資源配分のインバランスが起こることが懸念されたため、それぞれの資源を比例で配分し、小さい資源タイプが大きい資源タイプに完全に包含されるよう資源タイプを定義して、ユーザには自分のジョブの要求に合った資源タイプを選択してもらう形を取った。また、資源タイプを事前定義することにより、確保した GPU と近い側の CPU ソケット内のコアが確保されることを保証し、CPU・GPU 間通信の性能がある程度担保できることとなった。

AGE における制約事項として、1 つのジョブが複数の仮想ノードを要求する MPI ジョブであるときに、ジョブを構成する仮想ノードを同一の物理ノードから複数とることができない（パッキングした配置ができない）という問題があった。これは、ジョブ内で各仮想ノードを指定する際に物理ノードと同様にホスト名で弁別していることに起因する。配置のパッキングを行うことで、MPI プロセス間の通信がローカル通信になることや、ジョブが詰め込まれるためにスケジューリング効率が向上するなどの利点を予期していたが、実運用上はこれらの点が問題になることはなかった。また、同様の理由により、ジョブを実行しているノー

ドに SSH で接続し X11 転送を利用したデバッガ GUI の利用などの作業を行うという TSUBAME2 までは制約なくできていた操作が `node.f` 以外の資源ではできなくなった。こちらは AGE の `qrsh` コマンドにてジョブの実行開始と同時にシェルアクセスを得るという限定された条件下ではあるが代替することができた。

一方で、ノードを `cgroups` を用いて論理分割することにより、複数のジョブが OS の名前空間を共有することとなった。このため、ユーザプログラムが原因でノードがダウンする事態が発生した場合に、同一ノードを共有している他のユーザのジョブが巻き込まれやすくなるとともに、OS 名前空間を共有することで、あるユーザのジョブが別のユーザのジョブに干渉しやすくなってしまった。計算資源に関しては、`cgroups` で分割することで他のジョブのためのプロセッサ・コアの利用を制限しており、性能上の問題はほぼ起こらないと考えている。他のユーザのジョブが見える点については、`/proc` ファイルシステムのマウントオプション `hidepid` を用いることで、他のユーザのプロセス情報を遮断することができている。しかしながら、通常の Linux 環境ではあまり用いられていない機能であり、一部のソフトウェアでは明示的に非サポートとしている設定であることから、この設定に起因するバグを誘発するなどの欠点も存在する [2,3]。

ノードのカレンダー予約については、AGE の Advance Reservation(AR) 機能を用いて実装した。ノード全体を単一のキューに入れ、キュー内に予約枠を作る形となるため、任意の時刻に開始する予約を作成することができるようになった。TSUBAME では予約用のユーザインタフェースの作成や、課金規則との兼ね合いを考慮して毎時 0 分に予約を切り替える 1 時間単位の予約として実装した。予約はノード単位であるが、予約後のキューには `node.q` などのノードを比例分割する資源タイプのジョブを投入することができる。

ノードを動的に分割することとしたため、資源タイプごとにスケジューラインスタンスを分けて計算ノードを固定割り付けする必要がなくなった。このことはフレキシビリティの面では有利ではあったが、1 台のスケジューラインスタンスで全てのジョブをさばくことを意味しており、スケジューラ負荷の面では不利な変更となった。

4 インタラクティブジョブ専用キューの分離

スケジューラインスタンスが 1 台しかないことにより、定常的にスケジューラが高負荷となり、応答性能が悪化した。通常のバッチジョブの場合、ジョブの実行開始のタイミングをユーザが待ち構えていることはなく、実行開始オーバーヘッドは体感できないものであるが、デバッグなどでリアルタイムに計算ノードにアクセスする際には、ノードが空いていてもジョブ投入から実行開始まで 1 分以上待たされるという状況は許容しがたいものであった。また、TSUBAME3.0 は運用期間全体を通して年度初めの期間以外は定常的に混雑しており、利用したいと思った時に空いている計算ノードがないという状況も定常的に発生していた。

この問題に対処するために、TSUBAME3.0 運用中の 2019 年 11 月よりインタラクティブジョブ専用キューを導入した。[6] TSUBAME3.0 を構成する 540 台の計算ノードのうち 4 台、TSUBAME4.0 では 240 台中 2 台、いずれも通常のバッチキューの利用者への影響を最小限にとどめるべく 1% 未満となるノードを通常のキューから外し、インタラクティブジョブの待ち受けに専従させるものである。インタラクティブジョブに求められた主な要件は [6] にあるとおり、以下のようなものであった。

- インタラクティブジョブの実行要求は即時に実現されるか、失敗する
- インタラクティブジョブの実行要求には可能な限り応える
- インタラクティブジョブに対しては性能の保証を一切行わない
- 1 つのインタラクティブジョブがノードの全資源を使いつくさないよう、ジョブ当たりの利用可能資源量に制限をかける
- インタラクティブジョブ専用ノードでは多数のジョブが 1 ノード内で同時に実行されてよい (オーバーコミット状態)

スケジューラ上ではインタラクティブジョブ専用キューのジョブを 1CPU コアのみ利用する極小ジョブと定義し、`cgroup` としては `node_o` 相当の制限をかけることで 24CPU コアおよび GPU 1/2 台 (MIG による分割) への最大 24 ユーザによる共有アクセスを実現した。メモリはプロセッサと違い、複数ユーザで共有できるものではないため、計算ノード上のローカ

ル SSD の大部分をスワップ領域として利用し、提供可能なメモリ領域を拡大した。スワップ領域の利用により、アクセス性能が低下することはインタラクティブジョブ専用キューの設計上許容範囲と判断した。

上記から、計算ノード 2 台で、合計 384 名まで同時に利用可能となった。これは TSUBAME のインタラクティブジョブ専用キューを利用した実習を行った際に、受講者全員が問題なく接続できる容量であるとともに、システムの 1% 未満という本目的のために減らすことが許容されると思われるノード数とのバランスをとることができたと考えている。

TSUBAME3.0 における実装当初は、通常のバッチジョブと同じスケジューラインスタンスでインタラクティブジョブ専用キューのジョブも処理していたが、ジョブスケジューリングの遅延が 1 分以上と大きく、ユーザの快適性を損ねていたため、スケジューラインスタンスを分離し、インタラクティブジョブ専用キュー専用のスケジューラインスタンスを提供したところ、応答時間が数秒程度に減少し、快適に利用できるようになった。AGE の `qcrsh` コマンドはスケジューラインスタンス間の振り分けを想定していないため、インタラクティブジョブ専用キューに関するコマンドは `iqcrsh` のように接頭辞 `i` を用いて区別することとした。

5 システム利用率に関する考察

TSUBAME3.0 の運用期間中における資源タイプ別ノード時間積の推移を図 4,5 に示す。縦軸はノード時間積を各月の日数に単純に $24(\text{時間/日}) \times 540 \text{ or } 240(\text{ノード})$ をかけたもので、メンテナンス等を一切考慮しない提供可能なノード時間積の理論最大値に対する供給率を示す。実際にはメンテナンス以外にも、リソース分割による売れ残り部分や、ノード予約の開始間際で新規のジョブが開始できない時間など、さまざまな要因からシステム利用率 100% となることはない。

図 4 は各資源タイプの大きさを考慮せずに単純にノード時間積を足したものであり、理論最大値である 1.0 を超えている期間が多数発生している。このことは TSUBAME においてリソース分割を行わず全てのジョブを物理ノード上で実行した場合、その計算需要を全く捌ききれなかったことを意味する。図 5 は各資源タイプのノード時間積に、資源タイプごとに割り当てられる CPU コア数に応じた係数 (例: `q_node`: 1/4) をかけたものであり、こちらが TSUBAME のシステ

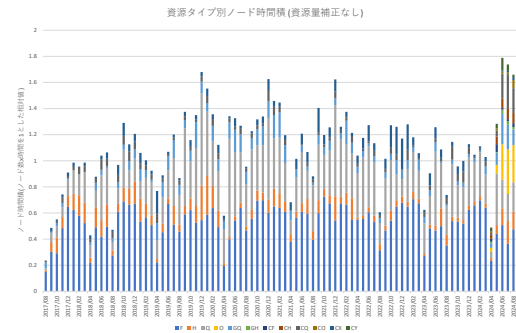


図 4 TSUBAME3.0/4.0 における資源タイプ別ノード時間積の推移 (資源量補正前)

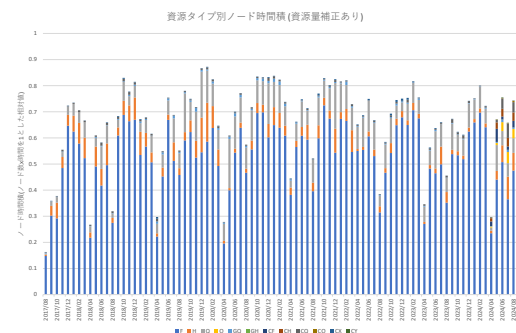


図 5 TSUBAME3.0/4.0 における資源タイプ別ノード時間積の推移 (資源量補正後)

ム充填率に近い指標となる。この指標で一番重点率が高かった月は 2019 年 12 月で、86.7% となっている。他方、この月のノード使用率 (ノードの一部だけでも使われていた時間を 1 とカウントしたノード単位の充填率) は 96.5% であった。これらの値から、リソース分割を行っても 89.8% もの効率でジョブを充填することができていたことが示されている。

6 おわりに

スーパーコンピュータを構成する部品の高度化と高価格化のトレンドは今後も続き、各センターにおいて同等の予算規模で調達可能な計算機のノード数は減少の一途を辿っている一方で、利用者の計算需要は高まり続けている。そのような状況下において、東京工業大学・東京科学大学ではその時々技術に基づき、計算ノードを論理的に分割することで、同時に処理できる計算ジョブの数を増やし、もしくは維持してきた。TSUBAME4.0 の資源分割は TSUBAME3.0 で導入

した Linux cgroups ベースの動的リソース分割を踏襲し、NVIDIA MIG の利用による GPU の論理分割や、CPU のみかつサイズが大きい資源タイプの導入により、さらに柔軟に計算資源の要求に応えることができるようになった。

本センターでは、システムの利用状況や利用者からの問い合わせ、フィードバックをもとに、リソース分割方法に限らずシステムの運用に改善できる点がないか常に検討を続けている。大規模な設計変更はシステムの代替わりのタイミングでしか実行できないが、小規模な改善や機能追加の機会は常にうかがっており、そのためにも利用者から率直な意見をお寄せいただけると大変ありがたい。

参考文献

- [1] Altair. Altair grid engine. <https://altair.com/grid-engine>.
- [2] GitHub psutil. [linux] ppid_map get permissionerror if /proc is mounted with hidepid=1. <https://github.com/giampaolo/psutil/issues/2026>.
- [3] GitHub singularity. fakeroot option does not work with hidepid procfs. <https://github.com/apptainer/singularity/issues/4633>.
- [4] NVIDIA. Nvidia multi-instance gpu user guide. <https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html>.
- [5] 東京科学大学情報基盤センター. TSUBAME 計算サービス. <https://www.t4.gsic.titech.ac.jp/>.
- [6] 野村哲弘, 遠藤敏夫, 三浦信一, 朝倉博紀, 越野俊充, 草間俊博. TSUBAME3 のインタラクティブ利用の利便性向上にむけた取り組み. 情報処理学会研究報告, 第 2020-HPC-175 巻, pp. 1–6, Jul 2020.
- [7] 遠藤敏夫, 野村哲弘, 渡邊寿雄, 安良岡由規, 鶴見慶. HPC-AI 時代に向けたもっとみんなのスパコン TSUBAME4.0. 情報処理学会研究報告, 第 2024-HPC-195 巻, Aug 2024.