Метод множественного использования графических процессоров в контейнерных средах

Н.Б. Лазарева

Тихоокеанский государственный университет, Хабаровск

Аннотация: В данной статье рассмотрена технология множественного использования графических процессоров и способы ее применения в контейнерных средах для запуска приложений, требующих наличия графического ускорителя вычислений.
Ключевые слова: графические процессоры, Kubernetes, контейнеризация, графический ускоритель, технология, сервер, машинное обучение.

В настоящее время возникает все больше задач, связанных с обработкой большого количества данных в целях проведения аналитики, а также задач, связанных с машинным обучением. Исторически так сложилось, что графические ускорители вычислений (graphics processing unit – GPU), которые изначально разрабатывались для задач отображения графики на дисплеях ЭВМ, архитектурно очень удачно подошли именно для таких задач. Экспериментальным путем был выявлен существенный прирост производительности при выполнении таких задач на GPU [1] по сравнению с традиционными центральными процессорами общего назначения (central processing unit – CPU). Использование GPU уже привело к существенному скачку в области искусственного интеллекта, начало которого было положено в 2006 году с созданием и применением технологии «Унифицированная архитектура вычислительных устройств» (Compute Unified Device Architecture – CUDA) [2]. В последующие годы GPU получали все более совершенные алгоритмы, а также совершенствовалась аппаратная реализация, особенности которой в итоге привели к существенной разнице в производительности между GPU и CPU [3].

Среди компаний, генерирующих вышеупомянутые специализированные задачи, требующие GPU, в последнее время становится все больше таких, которые стараются иметь в своих инфраструктурах

собственный парк серверов с GPU. При всех преимуществах такого подхода возникает проблема грамотной утилизации таких серверов, что особенно актуально с учетом высокой стоимости GPU. Возникают такие вопросы как:

1) Как использовать сервера таким образом, чтобы разные люди/подразделения/проекты могли обращаться к ним;

2) Как ускорить ввод таких северов в работу, как обновлять и менять конфигурации;

3) Как правильно утилизировать сервера, допуская минимальный простой;

4) Как запускать задачи на таких серверах;

5) Как автоматизировать все операции с такими серверами;

6) Как обеспечить безопасность данных при выполнении задач на серверах.

Большинство из этих вопросов покрывает ввод таких серверов в состав Kubernetes-кластеров [4]. Современные технологии и алгоритмы позволяют быстро вводить и выводить сервера с GPU, тем самым автоматизировать передачу таких серверов от одних потребителей к другим и существенно повысить утилизацию. Однако, в ряде случаев можно еще больше повысить утилизацию GPU и добиться большей гибкости при запуске тех или иных задач. Для этого можно воспользоваться технологией Nvidia «Множественное использование графических процессоров» (Multi-Instance GPU – MIG).

В основе технологии MIG лежит возможность дробления физических ресурсов на логические группы [5]. Например, для видеокарты Nvidia A100 предусмотрена возможность деления ресурсов GPU на 7 логических частей. Видеопамять видеокарты при этом так же может делиться на логические части.

Рассмотрим схему (рис.1), демонстрирующую общие принципы деления ресурсов видеокарты Nvidia A100:

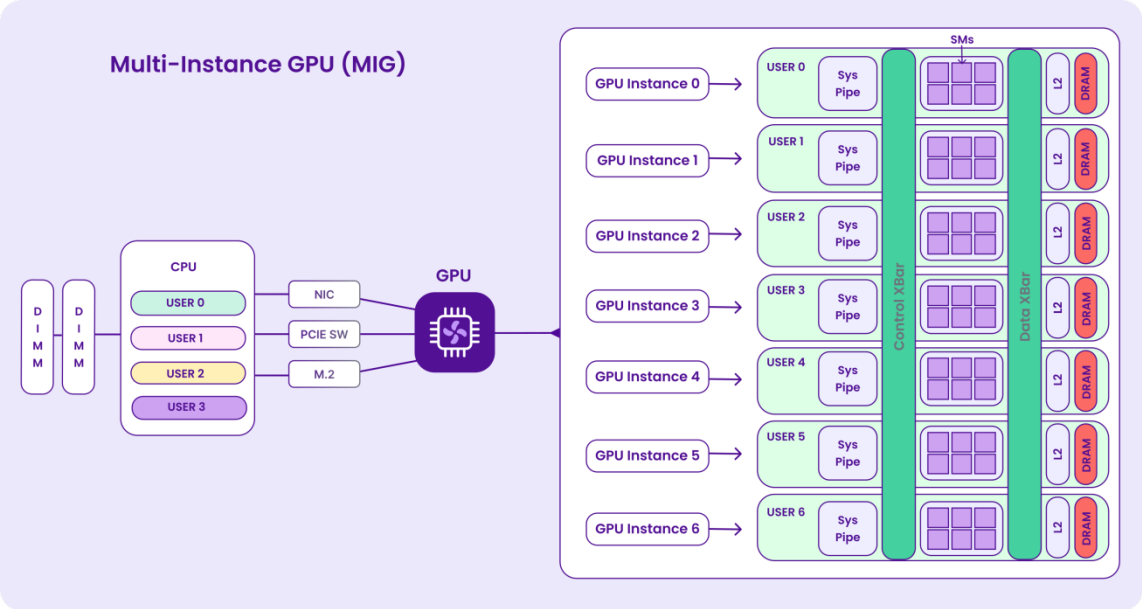

Рис. 1. – Общие принципы деления ресурсов видеокарты Nvidia A100 [6].

Как видно из данной схемы, каждая из семи логических групп обладает своей аппаратной реализацией и имеет свой выделенный объем видеопамяти.

Однако, технология MIG позволяет не только использовать эти логические группы по отдельности, но и объединять их вместе, образуя профили.

В таблице №1 продемонстрирована логика образования профилей на основе логических групп. Для видеокарты Nvidia A100 с объемом физической памяти 40Гб, каждая логическая группа будет обладать 5 Гб памяти. Если использовать логические группы без объединения в профили, то получится, что из общего объема 40 ГБ приложения получится аллоцировать лишь 35Гб, оставшиеся 5Гб не будут использоваться. Это архитектурное ограничение можно обойти. Например, можно сформировать профиль 3g.20gb. Расшифровка данного профиля проста: 3g – это количество логических групп GPU, 20gb – объем памяти, доступный этим трем группам.

Данный профиль можно использовать в комбинации с другими профилями, это отображено в строках 2, 5-11 таблицы №1. Таким образом, физические ресурсы видеокарты можно группировать различными способами в профили.

Таблица № 1

Профили на основе логических групп [7].

Config	GPC Slice #0	GPC Slice #1	GPC Slice #2	GPC Slice #3	GPC Slice #4	GPC Slice #5	GPC Slice #6	OFA	NVDEC	NVJPG	P2P	GPU Direct RDMA
1	7							1	5	1	No	Supported MemBW proportional to size of the instance
2	4				3			0	2+2	0	No	
3	4				2		1	0	2+1+0	0	No	
4	4				1	1	1	0	2+0+0+0	0	No	
5	3			3				0	2+2	0	No	
6	3			2		1		0	2+1+0	0	No	
7	3			1	1	1		0	2+0+0+0	0	No	
8	2		2		3			0	1+1+2	0	No	
9	2		1	1	3			0	1+0+0+2	0	No	
10	1	1	2		3			0	0+0+1+2	0	No	
11	1	1	1	1	3			0	0+0+0+0+2	0	No	
12	2		2		2		1	0	1+1+1+0	0	No	
13	2		1	1	2		1	0	1+0+0+1+0	0	No	
14	1	1	2		2		1	0	0+0+1+1+0	0	No	
15	2		1	1	1	1	1	0	1+0+0+0+0	0	No	
16	1	1	2		1	1	1	0	0+0+1+0+0+0	0	No	
17	1	1	1	1	2		1	0	0+0+0+0+1+0	0	No	
18	1	1	1	1	1	2		0	0+0+0+0+0+1	0	No	
19	1	1	1	1	1	1	1	0	0+0+0+0+0+0+0	0	No	

Приложение, которое необходимо запустить на сервере с GPU, может использовать как GPU целиком, так и работать на отдельном профиле MIG, имея ограниченный набор ресурсов.

В среде Kubernetes [8] приложение нужно сконфигурировать определенным образом, чтобы при развертывании и запуске оно затребовало наличия GPU целиком или определенной свободной группы MIG. Согласно документации [9], необходимо добавить запрос на MIG-группу в блоке ресурсов контейнера, в нашем случае для 3g.20gb:

```
resources:
    limits:
        nvidia.com/mig-3g.20gb: "1"
    requests:
        nvidia.com/mig-3g.20gb: "1"
```

Данная секция определяет необходимость наличия свободного профиля 3g.20gb на сервере с GPU и ограничивает приложение только этим профилем.

Если на сервере такой профиль окажется свободен, приложение сможет запуститься.

Важно упомянуть, что профили MIG реализуют не только логическое разделение ресурсов, но и полную изоляцию друг от друга. Это означает, что задача, запущенная в рамках одного профиля MIG, не сможет получить доступ к данным, используемым задачей, запущенной в рамках другого профиля MIG. Это полезно и реализует важный сценарий в области информационной безопасности.

Проверить работоспособность профиля MIG можно имея доступ к приложению, которое его использует. Для теста запустим приложение Ollama [10], предназначенное для запуска моделей. Подключившись к консоли контейнера с приложением Ollama, выполним команду *nvidia-smi* для отображения информации о GPU (рис.2).

```
root@ollama-5ccb794f5c-9jcxc:/# nvidia-smi
Wed Nov  5 14:46:18 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.90.12              Driver Version: 550.90.12      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-PCIE-40GB          On  |   00000000:0B:00.0 Off |                   On |
| N/A   34C    P0             67W /  250W |                  N/A   |     N/A      Default |
|                                         |                        |              Enabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| MIG devices:                                                                            |
+------------------+--------------------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |                     Memory-Usage |        Vol|      Shared           |
|      ID  ID  Dev |                       BAR1-Usage | SM     Unc| CE ENC DEC OFA JPG    |
|                  |                                |        ECC|                       |
|==================+================================+===========+=======================|
|  0    1   0   0  |              38MiB / 19968MiB    | 42      0 | 3   0    2    0    0 |
|                  |                 0MiB / 32767MiB  |           |                       |
+------------------+--------------------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Рис. 2. – Информация о GPU.

Из вывода, генерируемого данной командой, можно увидеть, что контейнер распознал видеокарту Nvidia A100 с объемом видеопамяти 40Гб. При этом блок *MIG devices* не пустой, информация в нем говорит о том, что данному контейнеру выделено только 19968Мб (округляя, 20Гб) видеопамяти и 3 логических части GPU (Shared CE – 3). Это значит, что для этого приложения применился профиль 3g.20gb, который и был для него запрошен.

Таким образом, в данной статье была рассмотрена технология Nvidia MIG и пример ее использования для запуска приложения на сервере с GPU с ограничением по ресурсам в контейнерной среде Kubernetes [11]. Технология открывает возможность гибкого и безопасного одновременного запуска различных задач в рамках единого физического пула ресурсов [12], что существенно повышает утилизацию GPU, что в свою очередь приводит к экономии бюджета компаний, владеющих дорогостоящими серверами с GPU.

Литература

1. Tazi N., Mom F., Zhao H., Nguyen P., Mekkouri M., Wolf T. The Ultra-Scale Playbook: Training LLMs on GPU Clusters. Creative Commons NonCommercial, ShareAlike (CC BY-NC-SA), 2025. 246 p.

2. Боресков А. В., Харламов А. А. Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. 232 с.

3. Кулакович А.Ю., Баранов Е.Ю. Оценка зависимостей времени работы алгоритма для восстановления расфокусированных изображений, выполняемого на CPU и GPU // Инженерный вестник Дона, 2019, №1. URL: ivdon.ru/magazine/archive/n1y2019/5518

4. Лазарева Н.Б. Автоматизация развертывания Kubernetes-кластеров на базе Ubuntu ОС в Rancher на инфраструктуре VMWare vSphere // Инженерный вестник Дона, 2023, №4. URL: ivdon.ru/magazine/archive/n4y2023/8325

5. Surati V. The Power of GPUs in AI and Machine Learning: Driving Modern Advancements. Independently published, 2024. 89 p.

6. Облачный провайдер ресурсов. URL: scaleway.com/en/docs/gpu/how-to/use-nvidia-mig-technology/

7. Документация Nvidia по профилям. URL: docs.nvidia.com/datacenter/tesla/mig-user-guide/supported-mig-profiles.html#supported-profiles

8. Burns B., Beda J., Hightower K., Evenson L. Kubernetes: Up & Running. O'Reilly Media, 2022. 202 p.

9. Документация Nvidia по работе с MIG в Kubernetes. URL: docs.nvidia.com/datacenter/cloud-native/kubernetes/latest/index.html

10. Работа с Ollama для запуска моделей. URL: kdnuggets.com/ollama-tutorial-running-llms-locally-made-super-simple

11. Masood F., Brigoli R. Machine Learning on Kubernetes. Packt Publishing, 2022. 384 p.

12. Raj E. Engineering MLOps. Packt Publishing, 2025. 370 p.

References

1. Tazi N., Mom F., Zhao H., Nguyen P., Mekkouri M., Wolf T. The Ultra-Scale Playbook: Training LLMs on GPU Clusters. Creative Commons NonCommercial, ShareAlike (CC BY-NC-SA), 2025. 246 p.

2. Boreskov A. V., Kharlamov A. A. Osnovy raboty s tekhnologiyey CUDA [Basics of working with CUDA technology]. M.: DMK Press, 2010. 232 p.

3. Kulakovich A.YU., Baranov Ye.YU. Inzhenernyj vestnik Dona, 2019, №1. URL: ivdon.ru/magazine/archive/n1y2019/5518

4. Lazareva N.B. Inzhenernyj vestnik Dona, 2023, №4. URL: ivdon.ru/magazine/archive/n4y2023/8325

5. Surati V. The Power of GPUs in AI and Machine Learning: Driving Modern Advancements. Independently published, 2024. 89 p.

6. Oblachnyy provayder resursov [Cloud resource provider]. URL: www.scaleway.com/en/docs/gpu/how-to/use-nvidia-mig-technology/

7. Dokumentatsiya Nvidia po profilyam [Nvidia Profile Documentation]. URL: docs.nvidia.com/datacenter/tesla/mig-user-guide/supported-mig-profiles.html#supported-profiles

8. Burns B., Beda J., Hightower K., Evenson L. Kubernetes: Up & Running. O'Reilly Media, 2022. 202 p.

9. Dokumentatsiya Nvidia po rabote s MIG v Kubernetes [Nvidia documentation on working with MIG in Kubernetes]. URL: docs.nvidia.com/datacenter/cloud-native/kubernetes/latest/index.html

10. Rabota s Ollama dlya zapuska modeley [Working with Ollama to launch models]. URL: kdnuggets.com/ollama-tutorial-running-llms-locally-made-super-simple

11. Masood F., Brigoli R. Machine Learning on Kubernetes. Packt Publishing, 2022. 384 p.

12. Raj E. Engineering MLOps. Packt Publishing, 2025. 370 p.

Дата поступления: 3.01.2026

Дата публикации: 22.02.2026