

Метод описания топологической структуры вычислительных кластеров основанный на операциях произведений подграфов*

Э.Р. Хабилова¹, А.Н. Сальников^{2,3}

¹ООО "Открытая Мобильная Платформа"

²МГУ имени М.В. Ломоносова,

³ФИЦ ИУ РАН

Топологическая структура коммуникационных сетей суперкомпьютерных систем при увеличении размера и сложности суперкомпьютеров соответственно усложняется. Для ее описания существует множество методов, однако такие описания являются громоздкими, что усложняет манипулирование ими. В статье предложен подход к описанию коммуникационной среды суперкомпьютера, когда коммуникационная сеть описывается как конструктор, где элементами конструктора являются типовые топологические структуры, часто встречающиеся в различных вычислительных системах. С этой целью разработан язык описания топологической структуры, основанный на операции произведения подграфов. Язык идейно схож в своих принципах с языками NetML и OMNeT++. Отдельное внимание в работе уделяется исключениям в регулярности сетей реальных суперкомпьютеров; с целью добавления возможности описания данного факта в язык внесены специальные конструкции. Для поддержки работы с языком описания разработана библиотека на языке программирования Си и специальная оболочка над ней написанная на языке Python3, которая затем может использоваться для визуализации описываемых языком графов. Выразительная мощность языка была продемонстрирована на описании вычислительных кластеров: Tianhe-2A, AI Bridging Cloud Infrastructure и Ломоносов-2. Метод был проверен и сравнен с GraphViz DOT показано многократное сокращение необходимого объема записи для некоторых крупных систем из Top500.

Ключевые слова: вычислительный кластер, топология компьютерной сети, языки описания графов, произведения подграфов

1. Введение

Задачи, требующие огромных вычислительных ресурсов, выполняются на специальных вычислительных системах, позволяющих производить триллионы и квадриллионы операций с плавающей точкой в секунду — суперкомпьютерах. Современные суперкомпьютеры построены как вычислительный кластер серверов, объединённых сложной высокопроизводительной сетью. Размеры суперкомпьютерных систем увеличиваются: число узлов достигает десятков тысяч, а число вычислительных ядер — нескольких миллионов (с актуальным рейтингом TOP 500 самых мощных суперкомпьютерных систем в мире можно ознакомиться в [1]). Коммуникационные сети, являющиеся «бутылочным горлышком», становятся одним из ключевых факторов в определении производительности.

Кроме этого, топологии суперкомпьютерных коммуникационных сетей постоянно развиваются. Существует множество распространенных сетевых топологий — начиная с более простых топологий, таких как «звезда», «n-мерная решетка», «n-мерный тор», «гиперкуб» (описанных в классическом учебнике [2]), заканчивая более сложными, но более эффективными (имеющими лучшие характеристики, такие как диаметр, связность, ширина бисекции, а также являющимися более экономически выгодными), такими как «утолщенное

*Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2022».

дерево» («fat tree», хорошо исследованная топология, описанная, например, в [3, 4]), «стрекоза» («dragonfly», впервые описанная в [5]), «плоская бабочка» («flattened butterfly», представленная в [6]), и другими. Более того, каждая топология имеет множество параметров, сильно влияющих на размеры и характеристики итоговой сети.

При таком разнообразии возможных топологий коммуникационных сетей и их возможных конфигураций, а также при увеличении размеров этих сетей усложняется задача выбора конкретной топологии при построении новых суперкомпьютерных систем, особенно принимая во внимание критичность этого выбора для производительности всей будущей системы в целом. В таком случае можно прибегнуть к моделированию суперкомпьютеров и их отдельных компонентов на системах меньшего размера.

Имитационное моделирование коммуникационных сетей суперкомпьютерных систем позволяет оценить характеристики коммуникационной сети и всей системы в целом. С его помощью можно оценить, подходит ли конкретная сеть именно для задач, которые планируется запускать на данном суперкомпьютере. Оно также является более экономически эффективным методом, чем, например, построение макета; анализ с помощью моделирования перед переходом к этапу анализа на макете позволяет сократить этот сложный и дорогостоящий этап. Кроме того, к моделированию коммуникационных сетей суперкомпьютеров прибегают при анализе производительности уже существующих систем, что может помочь использовать их более эффективно; яркий пример такого использования представлен в [7]. Подробнее про моделирование коммуникационных сетей суперкомпьютеров описано в [8, 9].

При увеличении размеров усложняется задача описания коммуникационной сети. При наличии в системе тысяч узлов, имеющих между собой еще большее число связей, установленных по неочевидным правилам, описания становятся нетривиальными, их размеры растут, а адресация конкретных узлов усложняется. При этом подробное описание сети необходимо не только при администрировании уже существующего суперкомпьютера, но и при моделировании, и в последнем случае требуется оперировать тем количеством описаний, сколько вариантов архитектуры сети рассматривается. Кроме того, в случае моделирования описания должны составляться и редактироваться человеком.

Топологии коммуникационных сетей суперкомпьютерных систем чаще всего имеют регулярную структуру. Даже в сетях, построенных для иных целей, можно наблюдать закономерности, что показано, например, в [10]. Возможно, этот факт можно использовать для их описания. Это могло бы как уменьшить размеры описаний, так и сделать написание их человеком более удобным, а также упростить адресацию отдельных узлов — в случае, если именование узлов можно вывести из структуры сети.

Целью является создать метод описания топологической структуры коммуникационных сетей суперкомпьютерных систем, подходящий для обмена описаниями, такой, чтобы:

- описания были компактней, чем перечисление всех узлов и всех ребер связей;
- описания использовали факт регулярной структуры сетей;
- с помощью описания можно было автоматически дать имя каждому узлу сети.

Имеет смысл рассмотреть существующие методы описания графов и методы описания сетей, используемые в системах моделирования. Рассмотрим их в контексте описания топологической структуры суперкомпьютеров.

Статья организована следующим образом: в разделе 2 представлены возможные подходы к описанию графов, в том числе некоторые популярные языки описаний; в разделе 3 подходы, применяемые именно к описанию сетей; в разделе 4 обсуждаются операции произведения подграфов и их применение к описанию сетей, так же приводится описание разработанного языка; в разделе 5 приводятся примеры базовых описаний и описаний некоторых популярных топологий вычислительных сетей; в разделе 6 приводятся результаты

сравнения объёмов описаний; в разделе 7 приводится описание функций созданных библиотек, для работы с описаниями графов; в разделе 8 некоторые обобщения.

2. Методы описания графов

Множество методов описания графов довольно обширно. Статья [11] описывает 76 файловых форматов, созданных для хранения и обмена графами. Авторы предлагают несколько классификаций графовых форматов. По одной из классификаций, все описанные файловые форматы разделяются на группы в соответствии с принципом представления графа. В соответствии с изложенным в статье доступные форматы представления графов можно разделить на:

- перечисляющие представления:
 - представляющие граф в виде списка всех ребер;
 - представляющие граф в виде матрицы смежности;
 - представляющие граф в виде разреженной матрицы;
 - представляющие граф в виде списка смежности;
 - представляющие граф в виде списка путей;
- процедурные, определяющие граф не перечислением вершин и ребер, а с помощью набора процедур. Такие форматы зачастую по сути используют многие библиотеки по работе с графами, например, Boost Graph Library [12].
- конструктивные, определяющие граф через операции на меньших графах.

В соответствии со сформулированными ранее целями работы оказывается, что первые пять вариантов (т.е. методы, основанные на перечислении; не являющиеся процедурными или конструктивными) нас не интересуют, поскольку подобные методы недостаточно компактны.

Процедурные методы описания графов в подавляющем большинстве случаев используют существующие языки программирования. По этой причине такие методы плохо подходят для обмена описаниями структур, поскольку описание, написанное, возможно, на другом языке программирования, разбирать достаточно сложно. Следовательно, они не соответствуют нашей цели.

Конструктивные методы лучше всего подходят для описания коммуникационных сетей суперкомпьютерных систем, поскольку такие сети зачастую имеют регулярную иерархическую или рекурсивную структуру.

Широко используемые языки **GraphViz DOT** [13], **TGF** (Trivial Graph Format), **Pajek** [14], **GraphML** (Graph Markup Language [15]), **GML** (Graph Modelling Language [16]), **GXL** (Graph eXchange Language [17]) представляют графы в виде списка ребер, а следовательно для решения поставленной задачи не подходят.

Единственным конструктивным методом, представленном в обзоре [11], является **NetML** [18]. Авторы обзора [11] относят его к конструктивно-процедурным. С помощью NetML кроме явного перечисления вершин и ребер графа можно использовать графы-звезды как элементы в описании более крупного графа. Также возможно применять преобразования разбиения ребер, слияния смежных ребер, и объединения ранее определенных подграфов. Данный язык слишком ограничен для наших целей, поскольку отсутствие параметризации и скромный набор преобразований не позволяет достаточно эффективно укоротить описания рассматриваемых графов: отсутствие включения сразу нескольких копий какого-либо подграфа, а также конструкций, упрощающих создание сразу нескольких ребер по некоторому правилу делают его непригодным для описания суперкомпьютерных сетей. Кроме

того, при повторном включении некоего подграфа необходимо вручную переименовывать каждую включенную вершину для избежания конфликта имен. Также этот метод не предоставляет способа автоматического создания имен вершин.

3. Методы описания, используемые в системах моделирования сетей

Большая часть систем моделирования и симуляции сетей использует процедурные методы описания сетей. Рассмотрим некоторые популярные системы.

ns-3 [19] является набором библиотек для симулирования коммуникационных сетей. Сети описываются только процедурным методом.

BigNetSim (BigSimulator, [20]) является симулятором коммуникационных сетей. Поддерживается ограниченное число топологий, и для описания собственных топологий требуется модификация исходного кода, что является процедурным методом описания сети.

mininet [21] – эмулятор, часто используемый для моделирования программно-определяемых сетей. Он имеет ограниченное число встроенных топологий, и для определения собственной топологии предлагается использовать программный интерфейс на языке Python3, то есть процедурный метод.

INRFlow [22] имеет несколько встроенных топологий и позволяет загружать граф сети из внешнего файла в виде списка ребер.

OMNeT++ (Objective Modular Network Testbed in C++ [23, 24]) – библиотека и платформа для дискретно-событийного моделирования. Создана и используется для моделирования коммуникационных сетей, многопроцессорных систем и других распределенных вычислительных систем.

В OMNeT++ для описания сетей используется предметно-ориентированный язык NED (NEtwork Description).

Сеть состоит из вложенных модулей. Модули могут быть простыми (**simple**) или составными (**compound**). Модули соединяются друг с другом с помощью портов (**gates**) – абстрактных интерфейсов ввода-вывода модулей. Порты могут соответствовать портам коммутаторов, сетевых карт или коммутационных панелей. Как подмодули, так и порты могут быть указаны в виде векторов.

Связи между модулями описываются в секции **connections** модулей. Для указания связей можно использовать циклы и условия. Это помогает в описании в том числе сложных структур с повторяющимися элементами и, в некоторой степени, нерегулярных структур. Пример описания сети на языке NED приведен на рис. 1.

Можно заметить, что данный метод описания коммуникационной сети является конструктивным. Однако единственным преобразованием, реализованным в этом методе, является объединение графов.

4. Построение описания графов на основе топологических произведений

На основании обзора существующих методов был выбран конструктивный метод описания. За основу взята идея иерархично-модульной структуры, наподобие использующейся в OMNeT++. Такая структура также позволит создавать уникальное полное имя для каждой вершины графа. Но для более компактного описания метод должен поддерживать больше преобразований над подграфами, чем объединение.

Операции произведения графов могут использоваться при описании графов сетей (более подробно это обсуждается в [10]). Рассмотрим некоторые виды графовых произведений, которые могут использоваться с помощью предлагаемого метода.

Напомним, что декартовым произведением множеств называется множество, элемен-

```

1 simple Hub
2   gates:
3     out: outport[];
4 endsimple
5
6 simple Station
7   gates:
8     in: In;
9 endsimple
10
11 module Star
12   submodules:
13     hub: Hub
14     gatesizes: outport[4];
15     station: Station[4];
16   connections:
17     for i=0..3 do
18       Hub.outport[i] --> Station[i].In;
19     endfor
20 endmodule
21
22 network star: Star
23 endnetwork

```

Рис. 1: Описание сети на языке NED

тами которого являются все возможные упорядоченные пары элементов этих множеств. Пусть графы G и H имеют непересекающиеся множества вершин $V(G)$ и $V(H)$. Произведением графов G и H называется граф $G \times H$ такой, что $V(G \times H) = V(G) \times V(H)$ (множество вершин графа-произведения равно декартовому произведению множеств вершин графов-членов произведения). Множество ребер графа-произведения зависит от того, какой именно вид произведения рассматривается.

Было принято решение добавить поддержку декартового (или прямого, $G \square H$) [25], тензорного ($G \times H$) [26], лексикографического ($G \cdot H$) [27], сильного ($G \boxtimes H$) [25] и корневого ($G \circ H$) [28] произведений графов.

Две вершины называются смежными ($u \sim v$), если они соединены ребром (u, v) .

Для всех видов произведений $P = G \times H$ графов G и H , где $e_g, f_g \in V(G)$ и $e_h, f_h \in V(H)$, две вершины $u = (e_g, e_h)$ и $v = (f_g, f_h)$ будут смежны при выполнении следующих условий:

- Для прямого произведения графов $(e_g, e_h) \sim (f_g, f_h) \Leftrightarrow e_g = f_g$ и $e_h \sim f_h$, либо $e_h = f_h$ и $e_g \sim f_g$.
- Для тензорного произведения $(e_g, e_h) \sim (f_g, f_h) \Leftrightarrow e_g \sim f_g$ и $e_h \sim f_h$.
- Для лексикографического произведения $(e_g, e_h) \sim (f_g, f_h) \Leftrightarrow e_g \sim f_g$ либо $e_g = f_g$ и $e_h \sim f_h$.
- Для сильного произведения — множество ребер равно объединению множеств ребер прямого и тензорного произведений.
- Для корневого произведения с заданным корнем $r_h \in H$ будет выполнено $(e_g, e_h) \sim (f_g, f_h) \Leftrightarrow e_g \sim f_g$, где $e_h = r_h$ и $f_h = r_h$, либо $e_g = f_g$ и $e_h \sim f_h$.

На рисунке 2 изображены примеры различных видов графовых произведений. Если принять изображенный на рисунке 2а граф за G , а изображенный на рисунке 2б — за H , то граф, изображенный на рисунке 2с, является прямым произведением G и H , изображенный на рисунке 2д — тензорным, а граф, изображенный на рисунке 2е, является одновременно

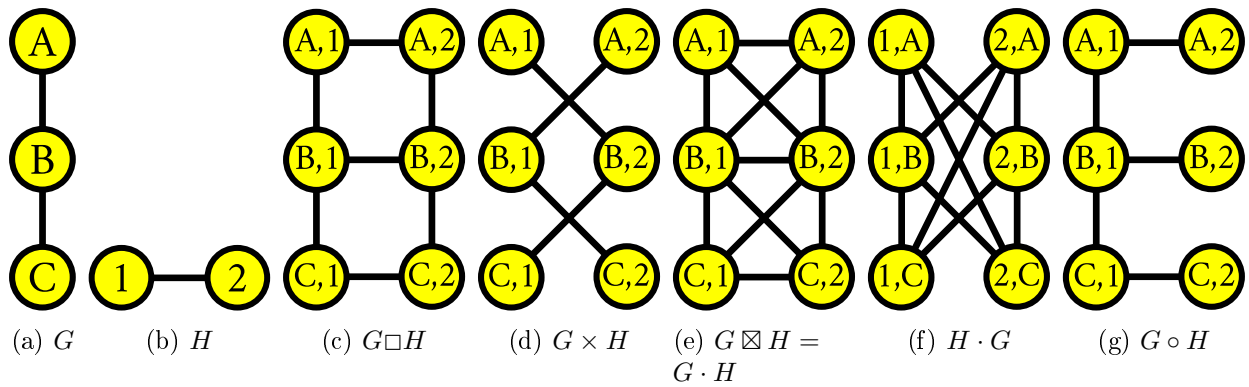


Рис. 2: Различные произведения графов

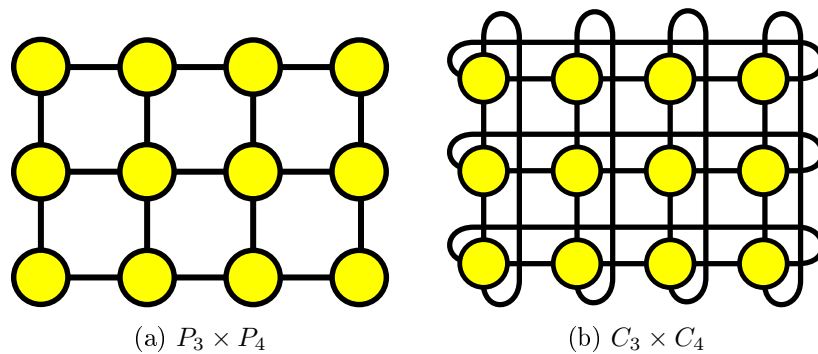


Рис. 3: Решетка и тор, выраженные через прямые произведения

и сильным, и лексикографическим произведением G и H . На рисунке 2f изображен граф, являющийся лексикографическим произведением H и G , что демонстрирует некоммутативность лексикографического произведения. На рисунке 2g изображен граф, являющийся корневым произведением G и H с выбранной в H корневой вершиной, помеченной цифрой 1.

Простыми для понимания примерами применения операций произведения для описания более сложных графов являются выражение решеток, торов и n -мерных кубов через прямое произведение графов-путей и графов-циклов. Подробнее про это написано в знаменитой книге [29]. На рисунке 3 изображены графы решетки и тора. Проиллюстрированная решетка может быть выражена как прямое произведение графов P_3 и P_4 , где P_n — граф-цепь длины n , а проиллюстрированный тор может быть выражен как прямое произведение C_3 и C_4 , где C_n — граф-кольцо длины n . Несмотря на простоту приведенных примеров, многомерные решетки и торы встречаются в топологиях суперкомпьютерных сетей (например, суперкомпьютерные системы серий IBM Blue Gene имеют сетевую топологию «многомерный тор» [30]), что демонстрирует возможное применение преобразования произведения подграфов в описании этих топологий.

Другим видом преобразования, которое может быть полезным для описания топологий суперкомпьютерных сетей, является замена одного подграфа на другой. Поскольку реальные системы зачастую имеют «дефекты», или «нерегулярности», связанные с тем, что либо суперкомпьютерную систему планируется расширить в будущем, либо существующую систему с регулярной топологией уже расширили — следует предусмотреть способ корректного описания таких случаев.

Наконец, можно упростить три простых, но часто встречающихся сюжета: соединение в некотором подмножестве вершин графа каждой вершины с каждой, и соединение вершин

некоторого подграфа в цепь или в кольцо.

4.1. Язык описания топологий

Было решено создать текстовый формат (для удобства чтения и написания человеком), основанный на JSON [31] (для удобства разбора в программных средствах). Пример описания полного графа в предлагаемом формате представлен на рис. 4.

```
1 [{ "simplemodule": { "name": "simple" } },
2 { "module": {
3   "name": "alltoall",
4   "submodules": [ { "name": "node",
5                     "module": "simple",
6                     "size": "10" } ],
7   "connections": [{ "all-match": "node" } ] },
8 { "network": { "module": "alltoall" } } ]
```

Рис. 4: Пример описания полного графа в предлагаемом формате

За базовую единицу построения был выбран *модуль* — некоторый подграф. Сеть (**network**) состоит из одного модуля. Модули разделяются на *простые* (**simplemodule**) и *составные* (**module**). Одному простому модулю соответствует одна вершина в графовом представлении.

Задача составного модуля заключается в объединении подграфов, составляющих несколько модулей, в некоторый больший граф. Составные модули могут иметь внутри несколько составных подмодулей, которые в конечном итоге будут состоять из простых модулей.

Модули соединяются друг с другом с помощью *портов* (**gates**). Как было упомянуто ранее, «порт» является абстрактным понятием: порты простых модулей могут соответствовать портам коммутаторов либо сетевых карт вычислительных узлов, а порты составных модулей могут соответствовать портам коммутационных панелей. Порты простых модулей напрямую связаны с вершиной, соответствующей этому модулю и могут соединяться не более чем с одним другим портом. Порты составных модулей могут соединяться не более чем с двумя другими портами.

Кроме включения в модуль подмодулей, указанных перечислением, в модуль можно включить сразу несколько подмодулей одного типа, включив *вектор* подмодулей. Параметр **size** определяет размер вектора. В таком случае в модуль добавляется несколько подмодулей с именем **name[i]**, где **name** — название, с которым был добавлен данный подмодуль. Каждому подмодулю, являющемуся частью вектора, доступен параметр **index**, который равен индексу подмодуля в векторе (про параметры будет подробно рассказано далее). С помощью механизма векторов подмодулей значительно упрощается задача описания повторяющихся элементов сетевой топологии.

Аналогично в виде вектора могут описываться порты, принадлежащие какому-либо модулю. Применим тот же принцип именования результирующих портов — **name[i]**. Таким образом упрощаются описания, например, коммутаторов с большим числом портов.

Подмодули могут иметь *параметры* (**params**). Параметры используются при вычислении значения арифметических выражений. Арифметические выражения могут встречаться в нескольких местах:

- при описании векторов подмодулей или векторов портов размер данных векторов (**size**) задается арифметическим выражением;
- при описании соединения — если соединяемый модуль является частью вектора; таким образом, в описание некоего соединения может входить, например, не **node[3]**, а **node[i]**, если этого требует конкретная топология;
- также арифметическим выражением может задаваться индекс порта, если указанный

порт является частью вектора портов;

- при условном включении подмодулей и при условном описании ребер арифметическим выражением задается выражение, определяющее истинность условия (про условное включение подмодулей и описание ребер подробнее рассказано далее);
- при использовании циклов для описания ребер графа арифметическим выражением задаются начальное и конечное значения переменной цикла (про описание ребер в цикле подробнее рассказано далее).

Параметры указываются при определении сети, при определении модуля, и при включении подмодулей. При наличии нескольких определений параметра с одним и тем же именем более приоритетными считаются определения, находящиеся «глубже» (например, параметры, определенные в данном модуле, приоритетнее, чем параметры, определенные для всей сети).

Для построения составного модуля подграфы, соответствующие его подмодулям, могут либо объединяться, либо входить в виде произведения (**cartesian**, **tensor**, **lexicographical**, **strong**, **rooted**). При описании **rooted** – корневого произведения, требуется указать корень — **root**.

Подмодули также могут входить в модуль условно — с помощью условного оператора (**if: ... , then: ... , else: ...**). Таким образом можно, например, описывать граничные условия рекурсивных структур.

Модули соединяются друг с другом с помощью портов. Однако при желании порты можно опустить и при указании ребер соединять простые модули напрямую; в таком случае создаются порты **_auto[n]**, соединенные с указанными вместо портов простыми модулями, где *n* — следующий незанятый порт, начиная с 0. Такой вид соединения может быть удобен в случаях, когда вычислять номер следующего свободного порта в векторе портов, связанных с некоторым простым модулем, трудоемко, и конкретный номер порта для каждого соединения не представляет важности. Нумерация автоматически созданных портов для каждого конкретного описания всегда одна и та же; принимаем, что соединения создаются в рамках каждого модуля последовательно в порядке их указания.

Перед применением операций произведения над подграфами в обязательном порядке производится операция *сжатия* — удаления всех портов, кроме имеющих лишь одно инцидентное ребро. Эта операция производится потому, что цепи, проходящие от одного простого модуля к другому через некоторую последовательность портов, соответствуют одному ребру в результирующем графе, и потому порты в операциях произведения участвовать не должны. «Висячие» порты, связанные лишь с одним простым модулем, в графе-результате произведения присутствуют у каждой вершины, являющейся результатом произведения исходного простого модуля с какой-либо вершиной второго графа. На рисунке 5 проиллюстрирован результат тензорного произведения двух подмодулей. Порты, связывающие узлы в исходных подмодулях, не являются частью результирующего модуля, но оставшиеся свободными порты — являются.

При необходимости исключить порты из итогового графа можно также производить операцию сжатия.

Ребра могут указываться как простым перечислением (**from: ... , to: ...**), так и в цикле по некоторой переменной (**loop: ... , start: ... , end: ... , conn: ...**). Также ребра могут указываться условно (**if: ... , then: ... , else: ...**). Кроме того, перечислив в виде цикла список простых модулей, возможно соединить их в линию (**line**), кольцо (**ring**), или каждую со каждой (**all**). Также возможно соединить набор вершин по принципу «каждая с каждой», задав его расширенным регулярным выражением в формате POSIX [32], поскольку порядок перечисления вершин в полносвязном графе не важен (**all-match**).

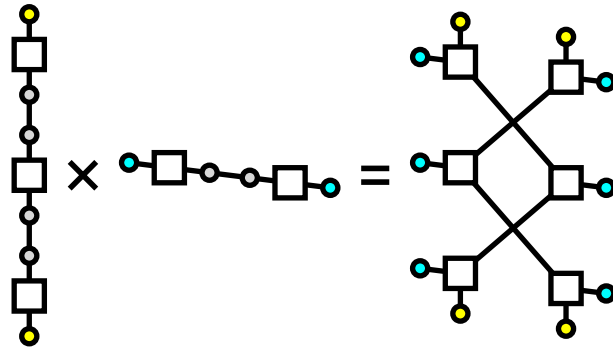


Рис. 5: Тензорное произведение двух подмодулей

Для описания «нерегулярностей» в описание модуля можно включить замещение части вложенных в него вершин и портов, описанных также расширенным регулярным выражением POSIX, другим подмодулем (`replace: nodes: ..., with: ...`). Заменой части вершин на пустой модуль можно отразить такую нерегулярность, как отсутствие узлов.

Каждый простой модуль и каждое соединение может иметь связанную с ними дополнительную информацию (`attributes`). С помощью дополнительной информации можно, например, указать пропускную способность соединений, или отличить маршрутизаторы от вычислительных узлов.

Имена всех модулей и портов создаются исходя из иерархической структуры сети. Именем каждого модуля является конкатенация через точку (.) имен всех модулей, в которые он входит, от внешнего к внутреннему, и имени самого модуля. Именем каждого порта является конкатенация через точку имени модуля, которому он принадлежит, и имени порта. Таким образом каждый узел и каждый порт имеет автоматически созданное имя, напрямую следующее из положения в структуре сети.

5. Примеры описаний

5.1. Описание решётки

Как было упомянуто ранее, многомерная решетка может быть представлена как прямое произведение нескольких простых цепей. Это свойство используется в данном описании для предоставления параметризации топологии не только по длине, но и по числу измерений. Решетка описана рекурсивно, и граничное условие рекурсии описано оператором `if`. Параметр `depth` на каждом шаге рекурсии уменьшается на единицу, пока не достигает значения 1. Базовая цепь описывается с помощью оператора `line`.

Пример описания многомерной решетки приведен в листинге 6, результирующий граф проиллюстрирован на рисунке 7¹. В иллюстративных целях каждой связи было придано свойство `style=dashed`, которое после конвертирования в формат GraphViz DOT было преобразовано в соответствующее свойство ребер и далее использовано программой визуализации графов для задания пунктирного вида ребер.

5.2. Пример описания жирного дерева (Fat tree)

Пример описания утолщенного дерева с нерегулярностями приведен в листинге 8, а результирующий граф проиллюстрирован на рисунке 9. Сама топология утолщенного дерева используется многими суперкомпьютерными системами, в том числе в находящихся на самом верху списка TOP 500 самых производительных вычислительных систем [1] — Summit

¹Узлы решетки на рисунке имеют имена вида `n.(a.s[1],p.(a.s[2],b.s[1]))`, в соответствии с указанными именами простого и составных модулей, и именем сети по умолчанию в используемой реализации

```

1 [{ "simplemodule": { "name": "grid_simple" }},
2 { "module": { "name": "grid_ring",
3   "submodules": [{ "name": "s", "module": "grid_simple", "size": "len"}],
4   "connections": [{
5     "line": "i",
6     "start": "0",
7     "end": "len",
8     "conn": "s[i]",
9     "attributes": "style=dashed" }]}},
10 { "module": { "name": "grid_prod",
11   "submodules": [ {
12     "if": "depth == 1",
13     "then": { "name": "a", "module": "grid_ring" },
14     "else": {
15       "cartesian": [
16         { "name": "a", "module": "grid_ring" },
17         { "if": "depth > 2", "then": {
18           "name": "p",
19           "module": "grid_prod",
20           "params": [ { "depth": "depth - 1" } ]
21         }, "else": {
22           "name": "b", "module": "grid_ring" }}}}]}},
23 { "network": {
24   "module": "grid_prod",
25   "params": [ { "len": "3" }, { "depth": "3" } ] }}]

```

Рис. 6: Пример описания многомерной решетки в предлагаемом формате

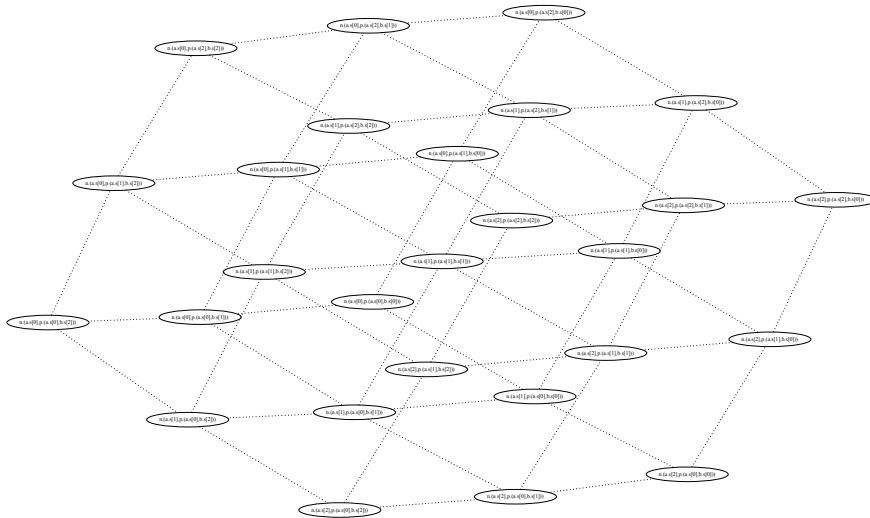


Рис. 7: Результирующий граф многомерной решетки

и Sierra [33], Tianhe-2A [34], AI Bridging Cloud Infrastructure [35], и многими другими.

В данном примере узлы, отражающие маршрутизаторы, и отражающие вычислительные узлы, описаны разными простыми модулями, что позволяет различить их с помощью указания дополнительной информации в поле **attributes**. Обобщенное утолщенное дерево описано с помощью модуля **gft_layer**. Ему передаются три параметра: **h** — высота дерева, **m** — число связей, уходящих «вверх», и **w** — число связей, уходящих «вниз». Более подробно про обобщенные утолщенные деревья описано в [4].

```

1 [{ "simplemodule": { "name": "gft_router" } },
2 { "simplemodule": { "name": "gft_endnode", "attributes": "shape=box" } },
3 { "module": { "name": "empty" } },
4 { "module": {
5     "name": "gft_layer",
6     "submodules": [{
7         "if": "h == 2", "then": {
8             "name": "node", "module": "gft_endnode", "size": "w^h"
9         }, "else": {
10            "name": "node", "module": "gft_router", "size": "w^h"
11        } }, { "if": "h > 0", "then": {
12            "name": "prev", "module": "gft_layer", "size": "m",
13            "params": [ { "h": "h-1" } ] } } ] },
14     "connections": [{
15         "if": "h > 0", "then": {
16             "loop": "i", "start": "0", "end": "w^h",
17             "conn": {
18                 "loop": "j", "start": "0", "end": "m",
19                 "conn": {
20                     "from": "prev[j].node[floor(i/w)]",
21                     "to": "node[i]" } } } } ] },
22 { "module": {
23     "name": "gft_alltoall",
24     "submodules": [{ "name": "node", "module": "gft_endnode", "size": "3" } ],
25     "connections": [{ "all-match": "gft.node\\[[0-2]\\]" } ] },
26 { "module": {
27     "name": "gft_m",
28     "submodules": [{
29         "name": "gft", "module": "gft_layer",
30         "params": [ { "h": "2" }, { "m": "2" }, { "w": "3" } ] },
31     "replace": [{
32         "nodes": "gft.node\\[[0-2]\\]",
33         "with": { "name": "gft", "module": "gft_alltoall" } }, {
34         "nodes": "gft.node\\[8\\]",
35         "with": { "name": "e", "module": "empty" } } ] },
36 { "network": { "module": "gft_m" } } ]

```

Рис. 8: Пример описания обобщенного утолщенного дерева с нерегулярностями в предлагаемом формате

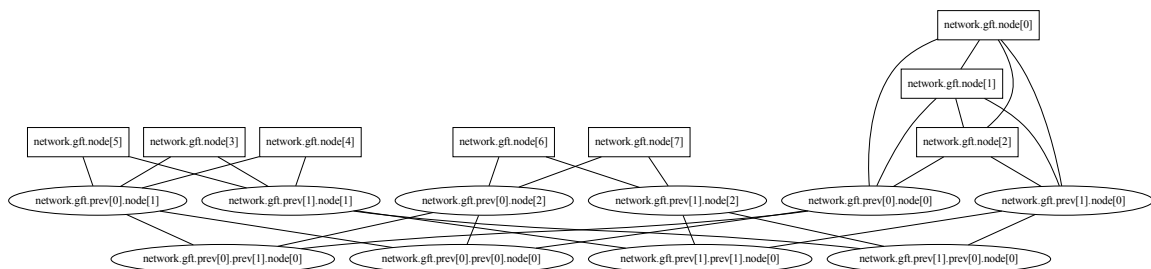


Рис. 9: Результирующий граф нерегулярного утолщенного дерева

В демонстрационных целях приведенная сеть имеет две нерегулярности: в одной из ветвей утолщенного дерева три узла связаны по принципу «все со всеми», а в другой ветви отсутствует один из узлов. Отсутствие одного из узлов описано с помощью его замены пустым модулем (в данном примере — **empty**).

5.3. Описание Ломоносов-2

```
1 [{ "simplemodule": { "name": "gft_router" } },
2 { "simplemodule": { "name": "gft_endnode", "attributes": "shape=box" } },
3 { "module": { "name": "empty" } },
4 { "module": {
5     "name": "gft_layer",
6     "submodules": [{
7         "if": "h == 2", "then": {
8             "name": "node", "module": "gft_endnode", "size": "w^h"
9         }, "else": {
10            "name": "node", "module": "gft_router", "size": "w^h"
11        } }, { "if": "h > 0", "then": {
12            "name": "prev", "module": "gft_layer", "size": "m",
13            "params": [ { "h": "h-1" } ] } } ],
14    "connections": [{
15        "if": "h > 0", "then": {
16            "loop": "i", "start": "0", "end": "w^h",
17            "conn": {
18                "loop": "j", "start": "0", "end": "m",
19                "conn": {
20                    "from": "prev[j].node[floor(i/w)]",
21                    "to": "node[i]" } } } } ] } },
22 { "module": {
23     "name": "gft_alltoall",
24     "submodules": [{ "name": "node", "module": "gft_endnode", "size": "3" } ],
25     "connections": [ { "all-match": "gft.node\\[[0-2]\\]$ " } ] },
26 { "module": {
27     "name": "gft_m",
28     "submodules": [{
29         "name": "gft", "module": "gft_layer",
30         "params": [ { "h": "2" }, { "m": "2" }, { "w": "3" } ] } ],
31     "replace": [{
32         "nodes": "gft.node\\[[0-2]\\]$ ",
33         "with": { "name": "gft", "module": "gft_alltoall" } }, {
34         "nodes": "gft.node\\[8\\]$ ",
35         "with": { "name": "e", "module": "empty" } } ] } },
36 { "network": { "module": "gft_m" } } ]
```

Рис. 10: Пример описания суперкомпьютера Ломоносов-2 в предлагаемом формате

Наконец, в листинге 10 приведено описание суперкомпьютера Ломоносов-2, установленного в МГУ им. М.В. Ломоносова. Подробнее система описана в [36] и [37].

В приведенном описании учтены нерегулярности, присутствующие в данном суперкомпьютере. Хотя топология суперкомпьютера основана на «плоской бабочке» размера $4 \times 8 \times 8$, к каждому маршрутизатору которой подключено по 8 узлов, на данный момент в системе лишь 1696 узлов; «плоская бабочка» имеет размеры $4 \times 8 \times 7$, и дополнительно отсутствуют 12 коммутаторов.

6. Сравнение размеров описаний графов

Для оценки компактности сравним описания некоторых топологий с помощью предложенного метода и с помощью GraphViz DOT (таблица 1).

Поскольку одно и то же описание, за исключением заданных параметров, может использоваться для нескольких частных случаев одной и той же топологии, значения в последней колонке повторяются.

Как видно, предлагаемый метод более краткий, чем GraphViz DOT, и разница более значительна для больших и регулярных сетей. Это объясняется тем, что после описания

Таблица 1: Сравнение с DOT

Топология		DOT, байт	Предлагаемый метод, байт	Отношение, %
Полный граф K_n	K_5	180	162	111%
	K_{10}	525	163	322%
	K_{20}	1 950	163	1.2%
Многомерная решетка	2x2x2x2	1 095	493	222%
	3x3x3	1 581	493	320%
	4x4x4	3 942	493	800%
Многомерный тор	2x2x2x2	1 095	493	222%
	3x3x3	1 803	493	366%
	4x4x4	4 356	493	884%
Утолщенное дерево $GFT(h, m, w)$	$GFT(2, 3, 1)$	554	421	132%
	$GFT(2, 2, 3)$	796	421	189%
	$GFT(3, 4, 3)$	11 058	421	2.6%
Плоская бабочка	2-ary 4-flat	246	382	64%
	4-ary 3-flat	698	382	183%
	4-ary 4-flat	3 836	382	1.0%
Tianhe-2A	Утолщенное дерево [34]	1 192 132	1 152	103.5%
AI Bridging Cloud Infrastructure	Утолщенное дерево [35]	71 749	1 002	7.2%
Ломоносов-2	Плоская бабочка $4 \times 8 \times 8$ [36]	143 539	1 258	11.4%

параметризованной топологии в предложенном формате размер описания не меняется при изменении его параметров, в то время как размер описания в формате GraphViz DOT неизбежно растет.

7. Библиотека для работы с топологиями в предложенном формате

Для работы с форматом была разработана библиотека, написанная на языке Си, не имеющая внешних зависимостей, за исключением стандартной библиотеки. Такое решение было принято для обеспечения большей переносимости между UNIX системами. Библиотека предоставляет следующие функции:

- `topologies_network_init` — производит инициализацию структуры, представляющей сеть;
- `topologies_network_read_file` — производит чтение из файла;
- `topologies_network_read_string` — производит чтение из строки;
- `topologies_definition_to_graph` — вычисляет графовое представление сети;
- `topologies_graph_compact` — производит операцию сжатия графа;
- `topologies_graph_string` — возвращает представление графа в формате GraphViz DOT;
- `topologies_graph_string_free` — освобождает память, выделенную под результирующую строку;

- `topologies_graph_print` — выводит представление графа в формате GraphViz DOT в файловый дескриптор;
- `topologies_graph_destroy` — освобождает память, выделенную под граф;
- `topologies_network_destroy` — освобождает память, выделенную под сеть.

Также было разработано программное средство на языке Python3 для визуализации графовой структуры описаний. Программе передается на вход один или несколько файлов, содержащих описание одной сети. Программа отображает интерактивное окно с визуализацией графа.

Для работы с описанной выше библиотекой, написанной на языке Си, из программного средства на языке Python3, был использован модуль `ctypes` из стандартной поставки Python3. С помощью этого модуля была написана обертка, предоставляющая интерфейс к библиотеке. Используя вышеописанную обертку основной код визуализатора конвертирует представление в формат GraphViz DOT, а затем передает полученный граф библиотеке `graph_tool`, которая и предоставляет функции интерактивной визуализации.

Оберткой для Python3 предоставляются следующие функции:

- `Topologies.__init__(self, definition)` — создает объект Python для работы с описанием сети. Параметр `definition` задает описание в виде строки.
- `Topologies.network_parse(self, compress, print_gates)` — производит работу с библиотекой и возвращает представление графа в формате GraphViz DOT в виде строки. При возникновении ошибки во время работы с библиотекой порождает исключение типа `ValueError`, содержащее строку с подробным описанием ошибки. Булевый параметр `compress` определяет, необходимо ли производить операцию сжатия (удаления всех портов, связанных с двумя вершинами графа). Булевый параметр `print_gates` определяет, включать ли порты, связанные менее чем с двумя вершинами графа, в итоговый граф.

8. Заключение

Итак, предложенный подход решает сложную задачу описания топологий сетей, которые носят в основном регулярную/симметричную структуру. Классические методы описания графов в данной ситуации не эффективны, поскольку приводят к громадным описаниям по сути одних и тех же элементов. Нами в статье был представлен метод описания графов топологий коммуникационных сетей суперкомпьютерных систем:

- являющийся компактным для регулярных структур, в том числе даже если эти структуры не совершенно идеальны;
- позволяющий именовать узлы сети исходя из описания топологии;
- предположительно, также подходящий для чтения и написания человеком (однако подтверждение этого факта требует дополнительного исследования и является возможным направлением развития данной работы).

Также были разработаны программные средства — библиотека для работы с разработанным форматом, и программа для визуализации описаний.

Метод был проверен и сравнен с GraphViz DOT на описаниях некоторых известных топологий и на описаниях существующих суперкомпьютерных систем, и оказался более кратким, особенно для больших регулярных сетей. При сравнении описаний систем Tianhe-2A, AI Bridging Cloud Infrastructure и Ломоносов-2 разница составила соответственно 103,484%, 7,160% и 11,410%.

Созданный программные коды библиотеки на языке Си и обёртки над ней на языке Python3 находится в открытом доступе по ссылке <https://github.com/asalnikov/topologies/>. В том же репозитории можно найти формальное описание грамматики предлагаемого формата.

Список литературы

1. June 2022 | TOP500 Supercomputer Sites. — <https://top500.org/lists/top500/2022/06/>. — Дата обращения 21 августа 2022.
2. В.В.Воеводин, Вл.В.Воеводин. Параллельные вычисления / Вл.В.Воеводин В.В.Воеводин. — БХВ-Петербург, 2002. — Рр. 67–71.
3. Leiserson, C. E. Fat-trees: Universal networks for hardware-efficient supercomputing / C. E. Leiserson // *IEEE Transactions on Computers*. — 1985. — Vol. C-34, no. 10. — Рр. 892–901.
4. Ohring, S. R. On generalized fat trees / S. R. Ohring, M. Ibel, S. K. Das, M. J. Kumar // *Proceedings of 9th International Parallel Processing Symposium*. — 1995. — Рр. 37–44.
5. Kim, J. Technology-Driven, Highly-Scalable Dragonfly Topology / J. Kim, W. J. Dally, S. Scott, D. Abts // *2008 International Symposium on Computer Architecture*. — 2008. — Рр. 77–88.
6. Kim, John. Flattened butterfly: a cost-efficient topology for high-radix networks / John Kim, William J. Dally, Dennis Abts // *ISCA '07*. — 2007.
7. Petrini, F. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q / F. Petrini, D. J. Kerbyson, S. Pakin // *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. — 2003. — Рр. 55–55.
8. Zheng, Gengbin. Simulation-Based Performance Prediction for Large Parallel Machines / Gengbin Zheng, Terry Wilmarth, Praveen Jagadishprasad, Laxmikant Kalé // *International Journal of Parallel Programming*. — 2005. — 06. — Vol. 33. — Рр. 183–207. DOI: 10.1007/s10766-005-3582-6.
9. Liu, N. Model and Simulation of Exascale Communication Networks / N. Liu, C. Carothers, J. Cope, P. Carns, Robert Ross // *Journal of Simulation*. — 2012. — 03. — Vol. 6. DOI: 10.1057/jos.2012.4.
10. Parsonage, E. Generalized graph products for network design and analysis / E. Parsonage, H. X. Nguyen, R. Bowden, S. Knight, N. Falkner, M. Roughan // *2011 19th IEEE International Conference on Network Protocols*. — 2011. — Рр. 79–88.
11. Roughan, Matthew. Unravelling Graph-Exchange File Formats / Matthew Roughan, Simon Jonathan Tuke // *CoRR*. — 2015. — Vol. abs/1503.02781. <http://arxiv.org/abs/1503.02781>.
12. Siek, Jeremy. The Boost Graph Library: User Guide and Reference Manual / Jeremy Siek, Lie-Quan Lee, Andrew Lumsdaine. — 2002. — 01.
13. Ellson, John. Graphviz - Open Source Graph Drawing Tools / John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, Gordon Woodhull // *Graph Drawing*. — 2001.
14. Batagelj, Vladimir. Pajek - Program for Large Network Analysis / Vladimir Batagelj, Andrej Mrvar. — 1999.

15. *Tamassia, Roberto*. Handbook of Graph Drawing and Visualization / Roberto Tamassia. — 1st edition. — Chapman & Hall/CRC, 2016.
16. *Himsolt, Michael*. GML: A portable Graph File Format / Michael Himsolt. — 2010.
17. *Holt, Richard C.* GXL: A graph-based standard exchange format for reengineering / Richard C. Holt, Andy Schürr, Susan Elliott Sim, Andreas Winter // *Science of Computer Programming*. — 2006. — Vol. 60, no. 2. — Pp. 149 – 170. — Special Issue on Software Analysis, Evolution and, Re-engineering. DOI: <https://doi.org/10.1016/j.scico.2005.10.003>. <http://www.sciencedirect.com/science/article/pii/S0167642305001176>.
18. *Batagelj, Vladimir*. Towards NetML Networks Markup Language / Vladimir Batagelj, Andrej Mrvar // International Social Network Conference, London. — 1995.
19. *Kumar, A. R. A.* NS3 Simulator for a Study of Data Center Networks / A. R. A. Kumar, S. V. Rao, D. Goswami // 2013 IEEE 12th International Symposium on Parallel and Distributed Computing. — 2013. — Pp. 224–231.
20. *Choudhury, Niles*. Scaling an Optimistic Parallel Simulation of Large-Scale Interconnection Networks / Niles Choudhury, Yogesh Mehta, Terry L. Wilmarth, Eric J. Bohm, Laxmikant V. Kalé // Proceedings of the 37th Conference on Winter Simulation. — WSC '05. — Winter Simulation Conference, 2005. — P. 591–600.
21. *Lantz, Bob*. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks / Bob Lantz, Brandon Heller, Nick McKeown. — 2010. — 01. — P. 19. DOI: 10.1145/1868447.1868466.
22. *Navaridas, Javier*. INRFlow: An interconnection networks research flow-level simulation framework / Javier Navaridas, Jose A. Pascual, Alejandro Erickson, Iain A. Stewart, Mikel Luján // *Journal of Parallel and Distributed Computing*. — 2019. — Vol. 130. — Pp. 140 – 152. DOI: <https://doi.org/10.1016/j.jpdc.2019.03.013>. <http://www.sciencedirect.com/science/article/pii/S0743731519302242>.
23. *Varga, András*. The OMNET++ discrete event simulation system / András Varga // *Proc. ESM'2001*. — 2001. — 01. — Vol. 9.
24. *Varga, András*. An overview of the OMNeT++ simulation environment / András Varga, Rudolf Hornig. — 2008. — 01. — P. 60. DOI: 10.1145/1416222.1416290.
25. *Sabidussi, Gert*. Graph multiplication / Gert Sabidussi // *Mathematische Zeitschrift*. — 1959. — Vol. 72, no. 1. — P. 446–457. DOI: 10.1007/bf01162967.
26. *Weichsel, Paul*. The Kronecker Product of Graphs / Paul Weichsel // *Proceedings of The American Mathematical Society - PROC AMER MATH SOC*. — 1962. — 02. — Vol. 13. DOI: 10.2307/2033769.
27. *Geller, Dennis*. The chromatic number and other functions of the lexicographic product / Dennis Geller, Saul Stahl // *Journal of Combinatorial Theory, Series B*. — 1975. — Vol. 19, no. 1. — P. 87–95. DOI: 10.1016/0095-8956(75)90076-3.
28. *Godsil, Chris*. A new graph product and its spectrum / Chris Godsil, B. McKay // *Bulletin of The Australian Mathematical Society - BULL AUST MATH SOC*. — 1978. — 02. — Vol. 18. DOI: 10.1017/S0004972700007760.
29. *Д.Кнут*. Искусство программирования, том 4, А. Комбинаторные алгоритмы, часть 1 / Д.Кнут. — Вильямс, 2013. — Pp. 48–50, 65–66.

30. *Chen, Dong*. The IBM Blue Gene/Q interconnection network and message unit / Dong Chen, Noel Easley, Philip Heidelberger, Robert Senger, Yutaka Sugawara, Valentina Salapura, David Satterfield, Burkhard Steinmacher-Burow, Jeffrey Parker. — 2011. — 12. — Pp. 1–10. DOI: 10.1145/2063384.2063419.
31. *T. Bray, Ed.* The JavaScript Object Notation (JSON) Data Interchange Format: RFC 8259 / Ed. T. Bray: RFC Editor, 2017. — 12. <https://www.rfc-editor.org/rfc/rfc8259.txt>.
32. IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7 // *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*. — 2018. — Pp. 1–3951.
33. *Stunkel, Craig B.* The high-speed networks of the Summit and Sierra supercomputers / Craig B. Stunkel, Richard L. Graham, Gilad Shainer, Michael Kagan, Sameh Sharkawi, Bryan S. Rosenberg, George Chochia // *IBM J. Res. Dev.* — 2020. — Vol. 64. — Pp. 3:1–3:10.
34. *Liao, Xiang-Ke*. High Performance Interconnect Network for Tianhe System / Xiang-Ke Liao, Zheng-Bin Pang, Ke-Fei Wang, Yu-Tong Lu, Min Xie, Jun Xia, Dezun Dong, Guang Suo // *Journal of Computer Science and Technology*. — 2015. — 03. — Vol. 30. — Pp. 259–272. DOI: 10.1007/s11390-015-1520-7.
35. *Takizawa, Shinichiro*. AI Bridging Cloud Infrastructure (ABCI) and its Communication Performance / Shinichiro Takizawa. — 7th Annual MVAPICH User Group Meeting. <http://mug.mvapich.cse.ohio-state.edu/mug/19/>.
36. *Voevodin, Vladimir*. Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community / Vladimir Voevodin, Alexander Antonov, Dmitry Nikitenko, Pavel Shvets, Sergey Sobolev, Igor Sidorov, Konstantin Stefanov, Vadim Voevodin, Sergey Zhumatiy // *Supercomputing Frontiers and Innovations*. — 2019. — Vol. 6, no. 2. <https://superfri.org/superfri/article/view/278>.
37. PARALLEL.RU. Суперкомпьютер “ЛОМОНОСОВ-2” | PARALLEL.RU - Информационно-аналитический центр по параллельным вычислениям. — <https://parallel.ru/cluster/lomonosov2.html>. — Дата обращения 15 мая 2020.