**УДК 004.051** **DOI: 10.14529/cmse190304**

# ОБЗОР МОДЕЛЕЙ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

**© 2019 Н.А. Ежова, Л.Б. Соколинский**
*Южно-Уральский государственный университет*
*(454080 Челябинск, пр. им. В.И. Ленина, д. 76)*
*E-mail: EzhovaNA@susu.ru, Leonid.Sokolinsky@susu.ru*
Поступила в редакцию: 01.06.2019

Цель данного обзора — дать максимально полное представление о достижениях и современном состоянии дел в разработке аналитических моделей параллельных вычислений, позволяющих предсказать время вычислений, ускорение, эффективность и масштабируемость параллельных алгоритмов применительно к различным целевым многопроцессорным платформам. Важность моделей параллельных вычислений вытекает из того, что они до реализации параллельного алгоритма в виде программы позволяют понять, насколько эффективно данный алгоритм может использовать конкретную многопроцессорную платформу, и при необходимости внести изменения в дизайн алгоритма, либо рассмотреть вариант замены целевой аппаратной платформы. В обзоре показывается эволюция моделей параллельных вычислений, происходившая одновременно с эволюцией многопроцессорных систем, от одноуровневых моделей с общей памятью до многоуровневых иерархических моделей с распределенной памятью, ориентированных на кластерные вычислительные системы с многоядерными ускорителями. В заключении обзора приводятся рекомендации по выбору возможных направлений дальнейших исследований в области разработки математических моделей параллельных вычислений.

*Ключевые слова: модель параллельных вычислений, обзор, параллельное программирование, многопроцессорные системы, оценка производительности, предсказание времени выполнения алгоритма.*

**ОБРАЗЕЦ ЦИТИРОВАНИЯ**

Ежова Н.А., Соколинский Л.Б. Обзор моделей параллельных вычислений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 3. С. 58–91. DOI: 10.14529/cmse190304.

## Введение

Модель вычислений представляет собой упрощенное, абстрактное описание компьютера. Проектировщики новых компьютеров, разработчики алгоритмов и программисты могут использовать модель вычислений для оценки своей работы, включая соответствие архитектуры компьютера различным приложениям, вычислительную сложность алгоритма и прогнозируемое быстродействие программы на различных вычислительных системах. Хорошая модель вычислений облегчает работу разработчиков компьютеров, алгоритмов и программ тем, что позволяет адекватно отобразить выбираемое решение на реальные компьютеры [1]. Подобную модель вычислений в литературе называют «*соединяющей моделью* (*bridging model*)» [2]. Универсальная соединяющая модель (см. рис. 1 а) может быть применима к любым алгоритмам и любым компьютерам. Такой универсальной соединяющей моделью вычислений для последовательных компьютеров и алгоритмов явились архитектура фон Неймана [3] и модель RAM (Random Access Machine) [4–8]. При появлении параллельных компьютеров были сделаны многочисленные попытки построить такую же универсальную соединяющую модель для параллельных алгоритмов [9], однако эти попытки потерпели неудачу. Это объясняется, главным образом, большим разнообразием многопроцессорных архитектур, появляющихся и развивающихся в настоящее время в погоне за повышением производительности.

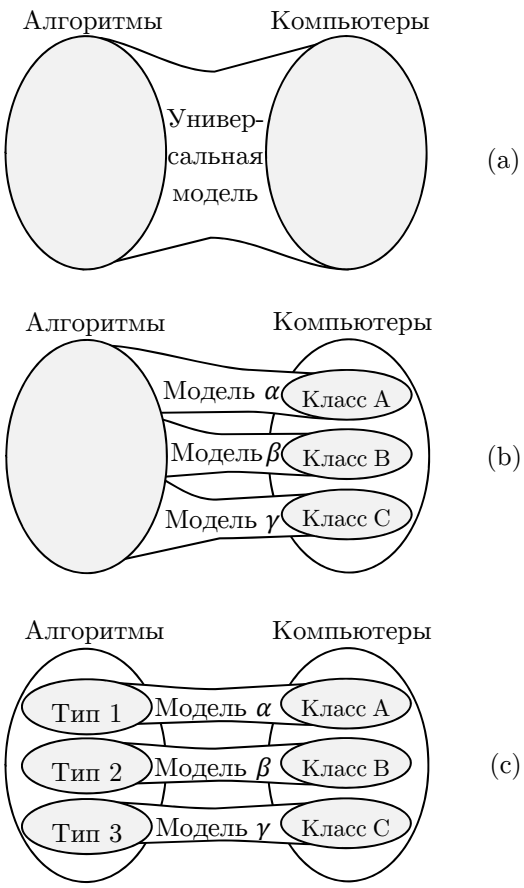


**Рис. 1**. Модели, соединяющие алгоритмы и компьютеры

В этих условиях создание простой, адекватной и универсальной модели параллельных вычислений является практически неразрешимой задачей. Для преодоления возникших трудностей был применен подход, схематично изображенный на рис. 1 b, в соответствии с которым многопроцессорные архитектуры были разделены на три класса: архитектуры с общей памятью, архитектуры с распределенной памятью и иерархические многопроцессорные архитектуры [1]. Для каждой из таких архитектур строились отдельные модели параллельных вычислений, которые, однако, оставались универсальными по отношению к множеству параллельных алгоритмов. Подобный подход позволил построить простые и универсальные модели с высоким уровнем абстракции, такие, как PRAM [10], BSP [2], LogP [11]. В целях адаптации таких моделей ко все более усложняющимся архитектурам многопроцессорных систем были предприняты многочисленные попытки их уточнения и расширения, что привело к появлению адекватных, но сложных для практического применения моделей параллельных вычислений (см., например, [12–17]). Исправить ситуацию в определенной мере позволяет подход, предполагающий разбиение всего множества алгоритмов по типам. Примерами различных типов алгоритмов могут быть итерационные численные алгоритмы, алгоритмы на графах, алгоритмы обработки больших данных и так далее. Для каждой пары (тип алгоритма, класс архитектуры) строится своя модель параллельных вычислений. Подобный подход позволяет достичь приемлемого компромисса между точностью оценок и простотой использования. В качестве примера можно привести модель параллельных вычислений BSF [18, 19], ориентированную на итерационные алгоритмы с высокой вычислительной сложностью и большие многопроцессорные системы кластерного типа.

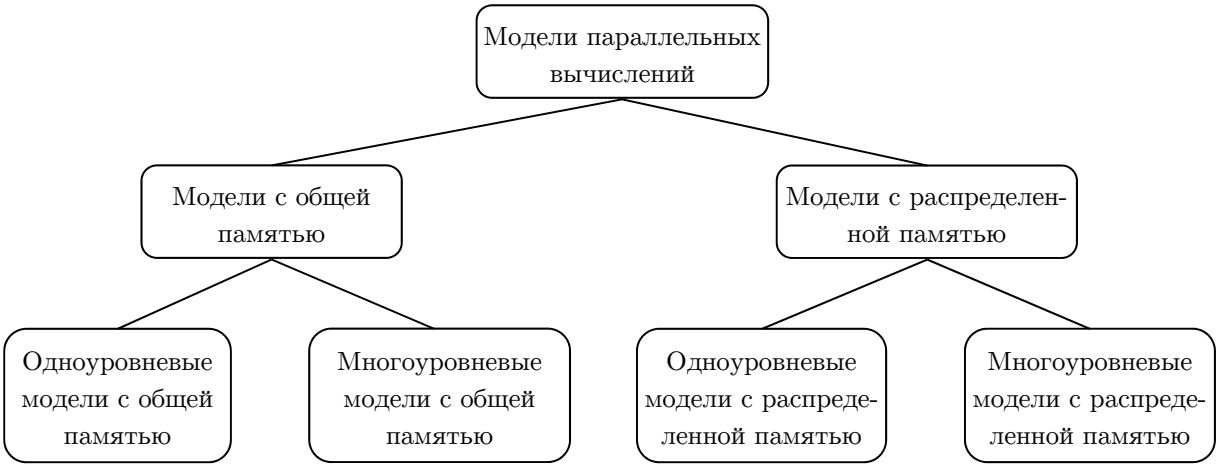


**Рис. 2**. Классификация моделей параллельных вычислений

Цель этой статьи — показать современное состояние дел в области разработки моделей параллельных вычислений и дать обзор наиболее важных и интересных моделей, разработанных к настоящему моменту. В обзоре используется классификация моделей, изображенная на рис. 2. Все модели параллельных вычислений делятся на два класса: модели для многопроцессорных систем с общей памятью и модели для многопроцессорных систем с распределенной памятью. Каждый из этих классов, в свою очередь, делится на два подкласса: одноуровневые модели и многоуровневые модели.

*Модели параллельных вычислений с общей памятью* ориентированы на многопроцессорные архитектуры класса SMP и отчасти класса NUMA. Архитектура SMP (Symmetric MultiProcessor) — объединяет в себе симметричные мультипроцессоры, являющиеся наиболее распространенным подклассом многопроцессорных систем с общей памятью [20]. Отличительными особенностями SMP систем является то, что все процессоры, входящие в систему, являются одинаковыми, и каждый процессор имеет одно и то же время доступа к любой ячейке оперативной памяти. Наиболее популярными представителями класса SMP являются современные серверные системы, построенные на многоядерных процессорах [21]. Многопроцессорные системы с архитектурой NUMA (Non-Uniform Memory Access) [22] также имеют общую оперативную память, но в отличие от SMP время доступа процессора к различным ячейкам памяти может существенно различаться. В настоящее время многопроцессорные системы класса NUMA практически исчезли с рынка.

*Модели параллельных вычислений с распределенной памятью* ориентированы на массово-параллельные [23] и кластерные архитектуры [24]. В простейшем случае многопроцессорная система с распределенной памятью представляет собой совокупность однопроцессорных серверов, объединенных высокоскоростной соединительной сетью. Именно такую структуру имели первые кластеры класса «беовульф» [25]. Однако, современные кластерные вычислительные системы имеют существенно более сложную структуру, включающую в себя SMP-узлы, внутри которых наряду с центральными многоядерными процессорами могут быть установлены дополнительные многоядерные ускорители, например, графические процессорные устройства (ГПУ) [26]. В этом случае может быть применено *двухуровневое* моделирование: сначала на уровне SMP-узла используется вычислительная модель для общей памяти, а затем на уровне кластерной системы в целом используется вычислительная модель для распределенной памяти.

*Модели параллельных вычислений с многоуровневой (иерархической) памятью* [27] ориентированы на современные SMP-системы, в которых кроме основной памяти имеются несколько уровней сверхоперативной кэш-памяти, встраиваемой в процессоры [28, 29]. Сюда же можно отнести и новый вид быстрой энергонезависимой памяти Intel Optane на базе технологии 3D XPoint [30], занимающей промежуточное положение между оперативной памятью и накопителями класса SSD. В общем случае модель многопроцессорной архитектуры с многоуровневой (иерархической) памятью можно определить как однородную систему, включающую в себя несколько уровней памяти с различным временем доступа. При этом определенные уровни памяти могут быть приватными для отдельных процессоров или их групп (в случае многоядерных сокетов под процессором понимается отдельное процессорное ядро). Модели параллельных вычислений для систем с многоуровневой памятью являются наиболее адекватными для приложений с интенсивным обменом данными между различными уровнями иерархической памяти. В качестве целевых аппаратных платформ для таких моделей часто фигурируют многоядерные кластеры с ГПУ, используемыми как ускорители, для которых характерно гибридное программирование с одновременным использованием технологий параллельных вычислений CUDA, OpenMP и MPI [31].

Обзор организован следующим образом. В разделе 1 дается определение модели вычислений и обсуждаются требования к качественной модели параллельных вычислений. В разделе 2 дается обзор одноуровневых моделей для многопроцессорных систем с общей памятью. Раздел 3 посвящен обзору многоуровневых моделей для многопроцессорных систем с общей памятью. В разделе 4 рассматриваются одноуровневые модели для многопроцессорных систем с распределенной памятью. В разделе 5 представлены многоуровневые модели для многопроцессорных систем с распределенной памятью. В разделе 6 обсуждаются близкие работы, являющиеся современными и более ранними обзорами моделей параллельных вычислений. Заключение содержит выводы и рекомендации относительно направления дальнейших исследований в области разработки новых моделей параллельных вычислений.

## 1. Требования к модели параллельных вычислении

Модель параллельных вычислений в общем случае должна включать в себя следующие пять компонентов, некоторые из которых в определенных случаях могут быть тривиальны [32]: *архитектурный компонент*, описываемый как помеченный граф, узлы которого соответствуют модулям с различной функциональностью, а дуги — межмодульным соединениям для передачи данных; *спецификационный компонент*, определяющий, что есть (синтаксически) корректный алгоритм/программа; *компонент выполнения*, задающий последовательность состояний архитектурных модулей, обеспечивающих корректное выполнение алгоритма/программы для определенных входных данных; *распараллеливающий компонент*, определяющий способы распределения вычислений, синхронизации и обмена данными между процессорными модулями, работающими независимо (параллельно); *стоимостный компонент*, определяющий одну или несколько метрик, позволяющих предсказать параметры выполнения алгоритма/программы (время выполнения, объем необходимой оперативной памяти и др.) в каждом конкретном случае.

В ранней работе Скилликорна [33] к модели параллельных вычислений предъявляются следующие требования: архитектурная независимость, согласованность, дескриптивная простота. *Архитектурная независимость* модели означает, что модель применима к

широкому классу многопроцессорных архитектур. Программа, написанная согласно модели, не должна требовать переписывания при портировании на другую архитектуру, необходима только перекомпиляция. *Согласованность* предполагает, что время, предсказываемое стоимостными метриками модели для данной архитектуры, должно (асимптотически) соответствовать реальному времени выполнения программы. *Дескриптивная простота* предполагает, что модель абстрагируется от низкоуровневых механизмов параллельной обработки, межпроцессорных коммуникаций и синхронизации процессов, что обеспечивает легкость ее применения при практическом программировании.

В качестве наиболее важных свойств модели параллельных вычислений в современной литературе выделяют следующие [34, 35].

– *Юзабилити*, определяющая легкость описания алгоритма и анализа его стоимости средствами модели (модель должна быть легкой в использовании).
– *Переносимость*, характеризующая широту класса целевых платформ, для которых модель оказывается применимой.
– *Предиктивность*, выражающаяся в возможности с помощью стоимостных метрик модели предсказать *временные характеристики* выполнения алгоритмов/программ на целевой вычислительной системе.

Основными *временными характеристиками* являются следующие:

1) *реальное* (*астрономическое*) *время* выполнения программы;
2) *абстрактное время* выполнения программы, позволяющее предсказать, какая из двух программ будет выполняться быстрее;
3) *масштабируемость* алгоритма/программы, определяющая максимальное количество вычислительных модулей в многопроцессорной системе, после которого прирост ускорения становится нулевым или отрицательным.

## 2. Одноуровневые модели с общей памятью

***Модель параллельных вычислений PRAM (Parallel Random Access Machine)*** [36] предполагает, что все процессоры работают синхронно под управлением одного тактового генератора и имеют произвольный доступ к общему адресному пространству оперативной памяти. Каждый процессор может выполнить арифметическую операцию, логическую операцию или операцию доступа к памяти за один машинный такт. Модель PRAM появилась в конце 1970-х годов как естественное расширение модели RAM (Random Access Machine) и была описана сразу в нескольких работах, среди которых можно выделить [10, 37, 38]. Модель PRAM получила широкое распространение и интенсивно использовалась при проектировании алгоритмов [39].

*PRAM*-компьютер (см. рис. 3) состоит из $K$ процессоров, соединенных с общей памятью неограниченного размера. Каждый процессор имеет свой уникальный номер. Предполагается, что доступ к любой ячейке памяти происходит за один машинный такт. В общем случае каждый процессор может выполнять свою собственную программу, однако на практике большинство *PRAM*-алгоритмов реализуются по схеме SPMD (Single Program Multiple Data) [40, 41], предполагающей, что все процессоры выполняют одну и ту же программу. На каждом машинном такте конкретный процессор может находиться либо в состоянии простоя, либо в активном состоянии. При этом все процессоры, находящиеся в активном состоянии, выполняют одну и ту же команду, но над разными данными.

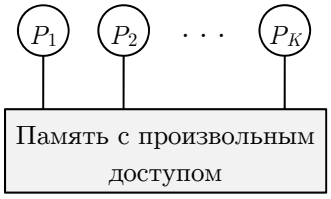


**Рис. 3.** *PRAM*-компьютер

Различают следующие три варианта модели PRAM в зависимости от политики разрешения конфликтов при параллельном доступе нескольких процессоров к одной и той же ячейке памяти в течении одного такта.

- EREW (Exclusive Read Exclusive Write) PRAM модель запрещает параллельный доступ к ячейке памяти и по чтению, и по записи.
- CREW (Concurrent Read Exclusive Write) PRAM модель разрешает параллельные чтения значения из ячейки, но запрещает параллельную запись в одну и ту же ячейку памяти.
- CRCW (Concurrent Read Concurrent Write) PRAM модель разрешает параллельный доступ к ячейке памяти как по чтению, так и по записи.

Указанные варианты модели PRAM характеризуются значительными различиями по времени выполнения параллельных алгоритмов. Рассмотрим следующую задачу: определить, являются ли все элементы двоичного массива $M[1 \dots n]$ нулями. Результатом вычислений в этом случае должно быть значение переменной $A$, являющееся результатом выполнения логической операции «ИЛИ» для всех элементов массива $M$. Параллельный CRCW PRAM алгоритм может решить эту задачу на $n$ процессорах следующим образом. Первоначально переменной $A$ присваивается значение 1. Затем все процессоры параллельно тестируют по одному элементу массива таким образом, чтобы индекс элемента совпадал с номером процессора. Если процессор обнаруживает в соответствующей ячейке значение 1, то он присваивает переменной $A$ значение 0. Очевидно, что описанный CRCW PRAM алгоритм корректно решит поставленную задачу на $n$ процессорах за $O(1)$ машинных тактов. В то же время, CREW PRAM алгоритм в общем случае потребует для решения этой задачи $\Omega(\log n)$ машинных тактов независимо от количества используемых процессоров [42]. С другой стороны, $p$-процессорная CREW PRAM модель может быть эмулирована с помощью $p$-процессорной EREW PRAM модели с замедлением не более, чем в $O(\log n)$ раз [43].

В теоретическом аспекте проектирование PRAM алгоритма состоит в разработке стратегии, позволяющей достигнуть максимального уровня параллелизма, при котором задача решается за минимальное количество параллельных шагов, используя разумное количество процессоров. В рамках модели PRAM задача считается хорошо распараллеливаемой, если она может быть решена за $O((\log n)^m)$ параллельных шагов при некотором фиксированном $m$ с использованием полиномиального количества процессоров. Данный класс задач получил название NC и был формально введен в работах [44, 45] применительно к логическим схемам.

Для описания PRAM-алгоритмов удобно использовать *фреймворк WT* (*Work-Time*). В соответствии с WT-фреймворком PRAM-алгоритм представляется в виде последовательности шагов, каждый из которых представлен оператором одного из двух классов:

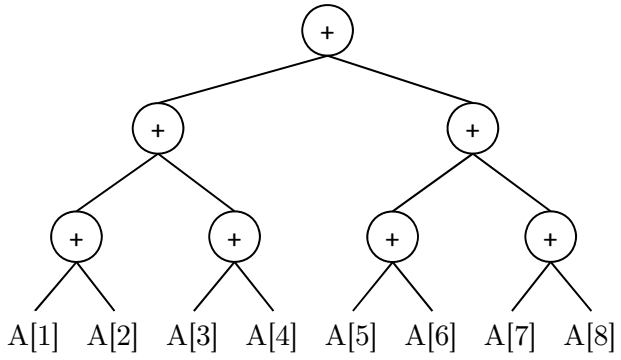


**Рис. 4**. *PRAM*-алгоритм, суммирующий элементы массива

последовательные операторы и параллельные операторы. Параллельные операторы специфицируются путем использования конструкций **pardo** и **for all**.

В качестве примера рассмотрим алгоритм PRAM SUM, суммирующий элементы числового массива $A[1..n]$, где $n = 2^l$ при некотором целом $l > 0$. Схема работы алгоритма PRAM SUM представлена на рис. 4. *PRAM*-алгоритм организуется как сбалансированное бинарное дерево, листьям которого соответствуют элементы массива $A$, а в качестве внутренних узлов фигурируют операции суммирования значений, представленных дочерними узлами. Алгоритм PRAM SUM выполняет вычисления снизу-вверх, осуществляя суммирование на каждом уровне за один параллельный шаг. При этом, в контексте рис. 4, на первом уровне задействуется четыре процессора, на втором — два, и на третьем — один. WT-реализация алгоритма PRAM SUM приведена в виде алгоритма 1. Из этой реализации видно, что решение состоит из $k$ параллельных шагов. Учитывая, что $n = 2^l$, отсюда получаем $l = \log n$. Следовательно, вычислительная сложность алгоритма составляет $O(\log n)$ машинных тактов. При этом задействуется $n \ / \ 2$ процессоров. Таким образом, задача суммирования элементов массива принадлежит классу NC и является хорошо распараллеливаемой в рамках модели PRAM. Очевидно, что границей масштабируемости алгоритма PRAM SUM является $K \leq n \ / \ 2$.

Отметим, что в общем случае временная сложность алгоритма в модели PRAM зависит от деталей его реализации, и в этом плане модель PRAM является низкоуровневой. Основным недостатком модели PRAM является то, что она не применима к многопроцессорным системам с распределенной памятью, так как не учитывает расходы на межпроцессорные коммуникации.

Модель PRAM позволяет строить чрезвычайно эффективные параллельные алгоритмы. Однако это достигается ценой следующих серьезных ограничений: все процессоры работают синхронно и межпроцессорные коммуникации выполняются мгновенно. На практике во многих случаях модель PRAM оказывается неадекватной [46]. В соответствии с этим был предложен ряд расширений и улучшений модели PRAM, которые более соответствуют реальным параллельным архитектурам.

Параллельные операции чтения и записи в память, допускаемые моделью PRAM, на реальных компьютерах могут приводить к конфликтам доступа к ячейкам памяти. Для предотвращения массовых конфликтов доступа к памяти в работе [47] была предложена ***модель модульно-параллельного компьютера*** (***Module Parallel Computer***), в соответствии с которой общее адресное пространство делится на $m$ модулей. Каждый

**Алгоритм 1** PRAM SUM — параллельное суммирование

**Input:** Числовой массив $A[1..n]$ размера $n = 2^l$ в оперативной памяти.

**Output:** Сумма элементов массива $A$.

```
begin
   d = n
   for j = 1 to l
      d = d/2
      for all 1 ≤ i ≤ d pardo
         A[i] = A[2i-1] + A[2i]
      end for
   end for
   return A[1]
end
```

модуль допускает только одно обращение в течении одного машинного такта. Указанная модель в определенной степени решает проблему конфликтов доступа к памяти, однако она игнорирует аспекты, связанные с пропускной способностью системной шины или соединительной сети. Другое, более реалистичное расширение модели PRAM, получившее название ***QRQW PRAM*** (***Queue-Read Queue-Write PRAM***), было предложено в [48]. Для разрешения конфликтов доступа к памяти в модели QRQW PRAM используются очереди доступа по чтению и записи для каждой ячейки памяти. Время доступа к ячейке памяти прямо пропорционально длине соответствующей очереди. Указанный подход был обобщен в ***модели QSM*** (***Queuing Shared Memory***) [49], которая была разработана с целью показать, что модель параллельных вычислений с общей памятью может играть роль соединяющей модели для параллельных алгоритмов. Для этого была выполнена прямая и обратная эмуляция модели QSM средствами BSP и других релевантных моделей.

С развитием архитектур параллельных компьютеров становилось все более очевидным, что доступ к нелокальной памяти оказывает серьезное влияние на производительность. Этот факт попытались учесть авторы ***модели BPRAM*** (*Block Parallel Random Access Machine*) [50]. В этой модели в качестве дополнительного параметра вводится латентность и предусматривается блочная передача сообщений между процессорами. При передаче блока данных между двумя процессорами на передачу первого машинного слова отводится $l$ тактов, а на передачу остальных машинных слов, входящих в блок, отводится $b$ тактов на каждое слово, причем $b$ много меньше $l$. Модель ***PRAM(m)*** [51] учитывает пропускную способность системной шины путем ограничения количества ячеек памяти, одновременно доступных для параллельных обращений в память. Эта модель базируется на модели CRCW PRAM с тем отличием, что на каждом такте могут быть выполнены не более $m$ обращений к памяти.

Стандартная модель PRAM предполагает, что параллельные процессы автоматически синхронизируются на каждом такте без дополнительных временных затрат. Это допущение не является реалистичным, так как на практике, с одной стороны, не требуется синхронизировать параллельные процессы на каждом машинном такте, и, с другой стороны, синхронизация процессов требует накладных расходов. Для учета первого фактора ***модели APRAM*** [52] и ***Asynchronous PRAM*** [53] допускают асинхронное выполнение процессов между двумя последовательными точками синхронизации. В отличие от этого ***модель XPRAM*** [54] предполагает периодическую глобальную синхронизацию асинхронных процессов. Однако, указанные модели не учитывают накладные расходы, связанные с синхронизацией процессов.

***Модель YPRAM*** [55] базируется на следующих трех характеристиках многопроцессорного компьютера, состоящего из $P$ процессоров: *латентность*, *пропускная неэффективность* и *рекурсивная декомпозиция*. Под *латентностью* $\delta(P)$ понимается максимальное время, необходимое для передачи одного машинного слова между любыми двумя процессорами или между процессором и общей памятью. *Пропускная неэффективность* определяется следующим образом. Пусть массив $M$ из $N$ чисел равномерно распределен между $P$ процессорами. Пусть $\tau(N)$ — максимальное время, требуемое для перестановки элементов массива $M$ среди всех возможных перестановок. Тогда пропускная неэффективность $\beta(P)$ вычисляется по формуле

$$\beta(P) = \lim_{N\to\infty} \frac{\tau(N)P}{N}. \tag{1}$$

*Рекурсивная декомпозиция* подразумевает, что многопроцессорный компьютер делится на два субкомпьютера с равным количеством процессоров и равным объемом разделяемой памяти, доступ к которой осуществляется на основе механизма EREW. Каждый субкомпьютер является самостоятельным компьютером и может быть рекурсивно разделен на две части. Для субкомпьютера из $S$ процессоров общее время, затрачиваемое на $M$ чтений из памяти, оценивается как

$$\Theta\left(\lambda(S) + m + \frac{M}{S}\,\beta(S)\right), \tag{2}$$

где $m$ — максимальное число чтений одного процессора.

***Модель HPRAM (Hierarchical PRAM)*** [56] рассматривает многопроцессорную систему с общей памятью как динамически конфигурируемую иерархию синхронных *PRAM-подсистем*, работающих асинхронно по отношению друг к другу. Иерархия определяет политику синхронизации между независимыми PRAM-системами. Помимо инструкций, присущих классической модели PRAM, HPRAM включает в себя специальную команду *partition* (*разбиение*), добавляющую в модель управляемый асинхронный параллелизм. Команда partition логически разбивает PRAM-(под)систему из $P$ процессоров на непересекающиеся подмножества и назначает каждому из них свой синхронный PRAM-алгоритм. Предусматриваются два варианта HPRAM-системы: *private HPRAM* и *shared HPRAM*. В *private HPRAM* команда partition разбивает память на непересекающиеся сегменты пропорционально размерам создаваемых PRAM-подсистем таким образом, что каждая PRAM-подсистема имеет свое собственное адресное пространство, недоступное для других PRAM-подсистем. В отличие от этого в *shared HPRAM* весь объем памяти остается в равной мере доступным для каждой создаваемой PRAM-подсистемы. *HPRAM-алгоритм* представляет собой совокупность PRAM-алгоритмов, организованных в иерархию. Каждая точка в иерархии ассоциируется с соответствующей командой partition. Благодаря введению иерархического разделения общей памяти, модели YPRAM и HPRAM в определенной мере позволяют абстрагироваться от низкоуровневых механизмов параллельной обработки, межпроцессорных коммуникаций и синхронизации процессов, и в этом плане частично удовлетворяют требованиям Скилликорна к моделям параллельных вычислений [33], однако это достигается существенным усложнением стоимостных метрик.

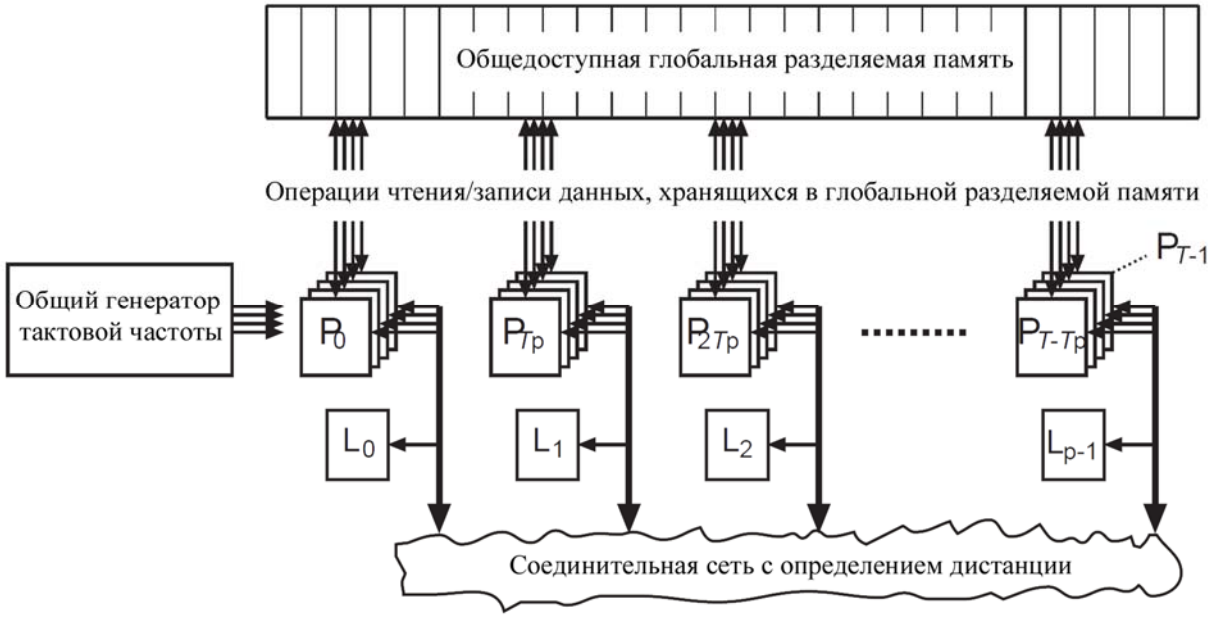


**Рис. 5.** PRAM-NUMA компьютер (P — процессор, L — локальная память)

В статье [57] предлагается ***модель PRAM-NUMA***, являющаяся расширением модели PRAM на многопроцессорные системы с распределенной памятью, работающие в режиме эмуляция общей памяти [58, 59]. Подобные многопроцессорные архитектуры получили название NUMA (Non-Uniform Memory Access) [24]. В отличие от модели PRAM, модель PRAM-NUMA вводит новую метрику, задающую *дистанцию* (относительное расстояние) между процессорами. PRAM-NUMA компьютер включает в себя следующие компоненты (см. рис. 5): $T$ процессоров, разбитых на $p$ групп по $T_P$ процессоров в группе; общедоступная глобальная разделяемая память; $p$ модулей локальной памяти; соединительная сеть, способная определять дистанцию между процессорными группами.

Каждый процессор соединен с глобальной разделяемой памятью, и каждая процессорная группа соединена со своим собственным модулем локальной памяти. Соединительная сеть обеспечивает доступ процессоров одной группы к локальной памяти остальных групп, при этом латентность маршрутизации прямо пропорциональна дистанции между процессором и модулем памяти, к которому он обращается. Пропускная способность каналов, соединяющих группу процессоров с общей и локальной памятью, является одинаковой. Каждый процессор может находиться либо в PRAM режиме, либо в NUMA режиме. В одной группе процессоров два или более процессора могут находиться в NUMA режиме. В этом случае они выполняют один поток команд над различными данными. Следует отметить, что NUMA-режим требует использования специального языка программирования «E» [60], сложного в использовании. В работе [61] предложено некоторое расширение модели PRAM-NUMA путем введения понятия «толстых» потоков управления, представляющих собой совокупность нитей управления, количество которых может меняться в ходе выполнения программы. Все нити одного «толстого» потока работают по схеме SIMD. Модель PRAM-NUMA ориентирована на многоядерные процессоры с нестандартной архитектурой и не может применяться для большинства современных многопроцессорных систем, поставляемых на рынке высокопроизводительной вычислительной техники.

## 3. Многоуровневые модели с общей памятью

Многоуровневые модели с общей памятью ориентированы на однородные многопроцессорные системы, в которых память делится на уровни с разным временем доступа [1]. Указанный класс моделей предназначен для более точной и реалистичной оценки времени доступа к памяти, организованной в виде иерархии уровней с монотонно возрастающими объемом памяти и временем доступа на каждом следующем уровне. Такие модели наиболее подходят для задач, связанных с перемещением больших объемов данных между различными уровнями иерархической памяти. В большинстве случаев модели этого класса вводят дополнительное множество измеряемых системных параметров, которые используются затем в качестве параметров стоимостных функций для повышения точности получаемых оценок.

Одной из первых последовательных моделей с многоуровневой памятью была ***модель HMM*** (***Hierarchical Memory Model***) [62]. HMM-компьютер содержит неограниченное количество регистров $R_1, R_2, R_3, \ldots$, каждый из которых может содержать целое число (или значение другого типа в зависимости от задачи). Операции выполняются так же как в модели RAM [8], за исключением того, что доступ к регистру $R_i$ требует $\lceil \log i \rceil$ единиц времени, в то время как выполнение любой операции занимает одну единицу времени. Очевидно, что P-сложная задача, выполняемая на RAM-компьютере за время $T(n)$, может быть выполнена на HMM-компьютере за время $O(T(n) \log n)$.

***Модель BT*** (***Hierarchical Memory with Block Transfer***) [63] расширяет модель HMM путем введения понятия блока, представляющего собой совокупность ячеек памяти с последовательными адресами. Время доступа к ячейке памяти с адресом $x$ определяется как $f(x)$. Однако, время доступа к блоку ячеек $[x - l, x]$ составляет $f(x) + l$. В качестве функции $f$ могут фигурировать $f(x) = \alpha$, $f(x) = x^{\alpha}$ и $f(x) = \log x$, где $\alpha$ — некоторая константа. Арифметические операции, как и в модели RAM, занимают одну единицу времени. В соответствии с этим сложение двух чисел, хранящихся в ячейках памяти с адресами $x$ и $y$, и сохранение результата в ячейке памяти с адресом $z$ занимает $f(x) + f(y) + 1 + f(z)$ единиц времени на BT-компьютере.

Еще одной ранней последовательной моделью вычислений для памяти с иерархическим доступом является *модель LPM* (*Logarithmic Pipelined Model*) [64]. Модель LPM предполагает, что доступ к $m$ последовательным ячейкам памяти занимает $\log m$ единиц времени. Это предположение согласуется с технологией VLSI (Very Large Scale Integration) [65], предназначенной для создания проблемно-ориентированных сверхбольших микросхем, включающих в себя ЦПУ, ПЗУ и ОЗУ. Логические вентили, с помощью которых осуществляется доступ к ячейкам памяти в VLSI системах, организуются обычно в виде древовидной структуры. В соответствии с этим доступ к $m$ последовательным ячейкам памяти не может быть осуществлен менее, чем за $\log m$ шагов.

Обобщением моделей HMM и BT является последовательная ***модель вычислений UMH*** (***Uniform Memory Hierarchy***) [66]. В рамках этой модели память представляется в виде иерархии модулей $\langle M_0, M_1, \ldots \rangle$. Например, $M_0$ может представлять регистровую память, $M_1$ — кэш-память, $M_2$ — оперативную память и так далее. Каждый

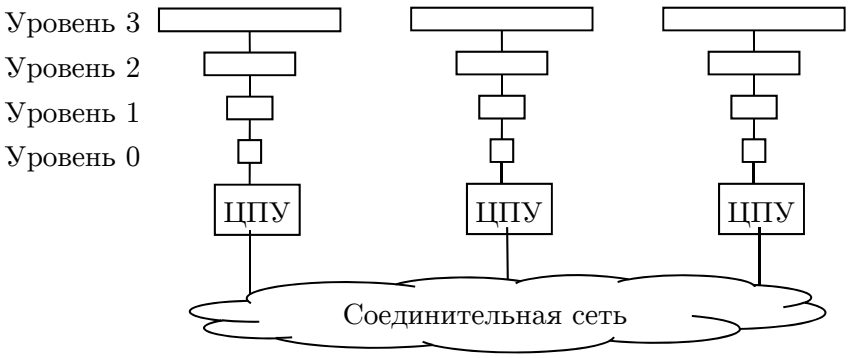


**Рис. 6**. Параллельная модель иерархической памяти

модуль памяти $M_u$ характеризуется тремя параметрами: $\langle s_u, n_u, l_u \rangle$. Параметр $s_u$ обозначает количество ячеек памяти в одном блоке; $n_u$ — количество блоков в модуле; $l_u$ — количество тактов, необходимых для копирования одного блока из $M_u$ в $M_{u+1}$ или обратно. Модель UMH обобщает модели HMM и BT в том смысле, что в модели UMH полагается: $n_u = \alpha s_u$, $s_{u+1} = \rho s_u$, $l_u = \rho f(u)$ для всех $u = 0,1,\ldots$. Здесь $\alpha$ и $\rho$ — положительные целочисленные константы, $f(u)$ — функция, определяющая количество тактов, необходимых для копирования одной ячейки памяти из уровня $M_u$ в уровень $M_{u+1}$. В качестве функции $f$ могут фигурировать, например, такие функции: $f(u) = 1$, $f(u) = u$ или $f(u) = \rho^u$. В общем случае предполагается, что всегда имеет место неравенство $f(u) \leq f(u+1)$.

Параллельные версии описанных многоуровневых моделей с общей памятью легко строятся путем репликации последовательной модели $p$ раз. ***Модели P-HMM*** **и** ***P-BT*** [67] обобщают для параллельного случая модели HMM и BT соответственно. Параллельная модель с иерархической памятью строится как совокупность HMM или BT-компьютеров, объединенных соединительной сетью (см. рис. 6).

При этом предполагается, что обмен данными по соединительной сети происходит через память уровня 0. Пропускная способность соединительной сети полагается такой, что сортировка массива, распределенного по модулям памяти уровня 0, выполняется за время $O(\log P)$, где $P$ — количество процессоров в параллельной машине. Отметим, что в рамках современных представлений указанная модель является адекватной, если уровни $1,2,\ldots$ представляют внешнюю память.

***Модель UPMH*** **(*Uniform Parallel Memory Hierarchy*)** [66] является параллельной версией модели UMH. UPMH — компьютер строится как совокупность UMH-машин, объединенных древовидной соединительной сетью. При этом для передачи сообщений используется память последнего уровня (самая медленная память).

## 4. Одноуровневые модели с распределенной памятью

Модели параллельных вычислений с распределенной памятью предполагают, что многопроцессорная вычислительная система строится как совокупность процессорных узлов, соединенных высокоскоростной коммуникационной сетью, обеспечивающей передачу сообщений от одного процессора другому. При этом каждый процессорный узел имеет

свою собственную приватную память, недоступную для других узлов. В этом разделе дается краткое описание моделей параллельных вычислений BSP, LogP, их модификаций, а также некоторых других одноуровневых моделей для многопроцессорных систем с распределенной памятью.

***Модель BSP (Bulk-Synchronous Parallelism)*** была предложена Валиантом (Valiant) в работе [2]. Данная модель широко используется при разработке и анализе параллельных алгоритмов и программ. BSP-*компьютер* представляет собой систему из $K$ процессоров $P_1,\ldots,P_K$, имеющих приватную память $M_1,\ldots,M_K$, и соединенных сетью, позволяющей передавать данные от одного процессора другому (см. рис. 7). Для соединительной сети вводятся следующие характеристики: $g$ — время, необходимое для передачи по сети одного машинного слова; $L$ — время, необходимое для выполнения глобальной синхронизации. Мы будем предполагать, что время измеряется в машинных тактах. Моделирование передачи сообщений в BSP-компьютере реализуется с помощью понятия $h$-сессии. *$h$-сессия* является абстракцией произвольной коммуникационной операции, в ходе которой каждый процессор передает не более $h$ машинных слов и получает не более $h$ машинных слов. Время на выполнение одной $h$-сессии в BSP-компьютере не может превышать $h \cdot g$.

*BSP-программа* состоит из $n$ потоков команд, каждый из которых назначается отдельному процессору, и делится на *супершаги*, которые выполняются последовательно относительно друг друга. Каждый *супершаг*, в свою очередь, включает в себя следующие четыре последовательных шага: 1) вычисления на каждом процессоре с использованием только локальных данных; 2) глобальная барьерная синхронизация; 3) пересылка данных от любого процессора любым другим процессорам путем выполнения одной $h$-сессии; 4) глобальная барьерная синхронизация. Переданные данные становятся доступными для использования только после барьерной синхронизации. Пример BSP-программы из двух супершагов приведен на рис. 8. Жирными линиями обозначены локальные вычисления, тонкими линями со стрелками — пересылка данных.

*Стоимостная функция* в модели BSP строится следующим образом [68, 69]. Пусть BSP-программа состоит из $S$ супершагов. Предположим, что каждый процессор на $i$-том супершаге выполняет максимум $w_i$ тактов в ходе локальных вычислений. Тогда общее время $t_i$, затрачиваемое системой на выполнение $i$-того супершага, вычисляется по формуле

$$t_i = w_i + h \cdot g + L. \tag{3}$$

Время $T$ выполнения всей программы определяется по формуле

$$T = W + h \cdot g \cdot S + L \cdot S, \tag{4}$$

где $W = \sum_{i=1}^{S} w_i$. Значения $W$ и $S$, как правило, зависят от количества процессоров $K$ и от размера задачи.

При проектировании BSP-программы значения $K$, $g$ и $L$ рассматриваются как конфигурационные параметры многопроцессорной системы. Эффективный BSP-алгоритм должен минимизировать количество локальных вычислений, объем передаваемых данных и число глобальных синхронизаций при некоторых реалистичных значениях указанных параметров. Основными принципами разработки BSP-программ являются следующие.

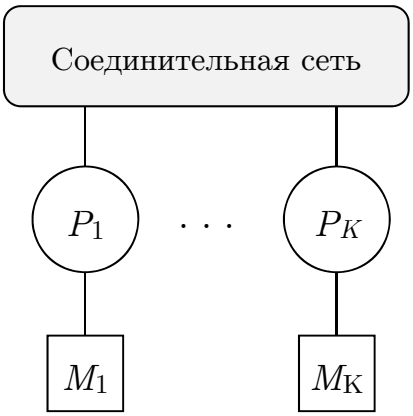


**Рис. 7**. BSP-компьютер

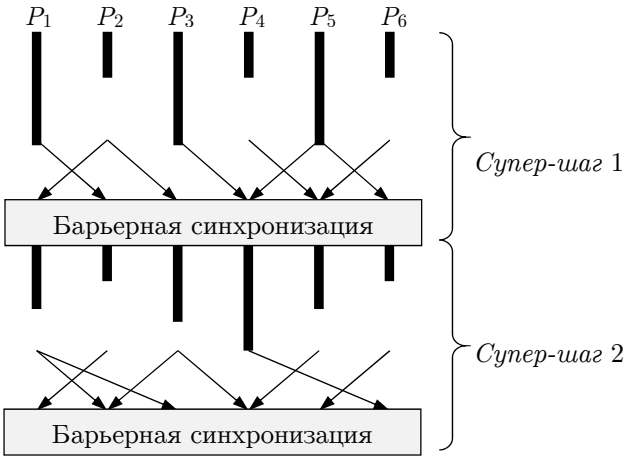


**Рис. 8.** Вычислительный процесс BSP

- Балансировка загрузки процессоров, позволяющая сократить максимальное время локальных вычислений.
- Локальность данных, позволяющая сократить затраты на коммуникации.
- Крупнозернистый параллелизм, позволяющая сократить затраты на коммуникации и глобальную синхронизацию.

Балансировка загрузки процессоров касается и равномерного распределения вычислительной нагрузки, и пропорционального распределения данных между процессорами. Принцип *локальности данных* говорит о том, что в локальной памяти процессора необходимо хранить данные, к которым этот процессор обращается чаще всего. *Крупнозернистый параллелизм* подразумевает разделение программы на крупные параллельные секции, каждая из которых выполняется на отдельном процессоре, содержит значительный объем вычислений и не предполагает обменов данными в ходе этих вычислений. Значения параметров $g$ и $L$ для соединительной сети конкретного компьютера могут быть получены путем бенчмаркинга (эталонного тестирования). Методика бенчмаркинга в контексте модели BSP описана в работе [70]. Там же можно найти полученные значения параметров для некоторых реальных многопроцессорных систем.

В качестве примера рассмотрим простейший параллельный алгоритм, вычисляющий произведение квадратной матрицы $A$ размера $n \times n$ на вектор $b$ размера $n$. Для решения задачи задействуем $K = n + 1$ процессоров: $P_0, P_1, \ldots, P_n$. Строки матрицы $A$ распределим по процессорам следующим образом: $i$-тая строка $a_i$ матрицы $A$ будет храниться в памяти процессора $P_i$ $(i = 1, \ldots, n)$. В память каждого процессора также поместим полный вектор $b$. Каждый процессор $P_i$ параллельно выполняет умножение своей строки $a_i$ на вектор $b$ и получает одно число. Это число пересылается процессору $P_0$, который формирует результирующий вектор. Таким образом, мы имеем BSP-алгоритм, состоящий из одного супер-шага. Умножение строки $a_i$ на вектор $b$ состоит из $n$ операций умноже-ния и $n-1$ операций сложения. Следовательно, временные затраты на вычисления составят $w = O(n) + O(n-1) \approx O(n)$. Коммуникационные затраты составят $gn$ в предположении, что $h = n$. Суммарные временные затраты, таким образом, составят $O(n) + gn + L$.

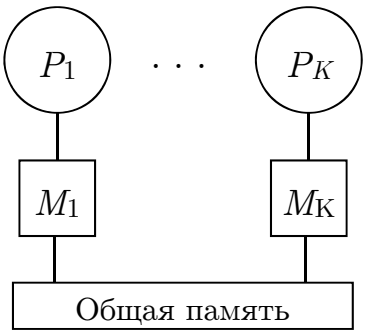


**Рис. 9**. *BSPRAM*-компьютер

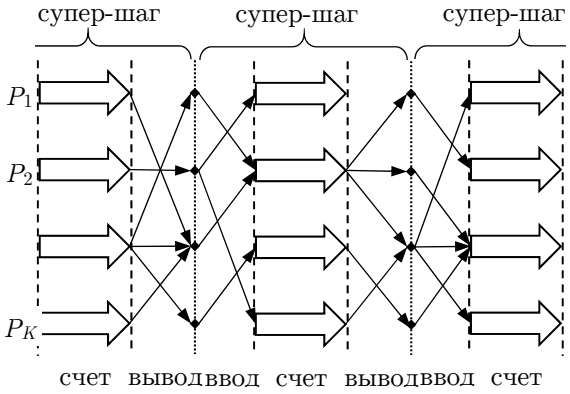


**Рис. 10**. Вычислительный процесс *BSPRAM*

В работе [71] предлагается более сложный BSP-алгоритм умножения плотной матрицы на вектор, вычислительная сложность которого при $K = n$ составляет $O(n) + g\sqrt{n} + L$. Основным недостатком модели BSP является то, что она предполагает передачу сообщений длиной в одно машинное слово и не учитывает тот факт, что передача $h$ машинных слов в виде одного сообщения может быть более эффективной, чем передача $h$ сообщений длиной в одно машинное слово.

***Модель BSPRAM*** (***Bulk Synchronous Parallel Random Access Machine***), предложенная в работах [72, 73], является вариацией моделей BSP и PRAM. BSPRAM-компьютер состоит из $K$ процессоров $P_1,\ldots,P_K$, каждый из которых имеет свою локальную память $M_1,\ldots,M_K$. В дополнение к этому имеется общая память, в равной мере доступная всем процессорам (см. рис. 9). Процессоры могут выполнять различные потоки команд. Вычислительный процесс в модели *BSPRAM* представляет собой последовательность супершагов (см. рис. 10). Супершаг включает в себя три фазы: *«ввод»*, *«счет»* и *«вывод»*. На фазе «ввод» процессоры читают данные из общей памяти; на фазе «вывод» процессоры пишут данные в общую память. Процессоры синхронизируются при переходе от текущего супершага к следующему; внутри супершага процессоры работают асинхронно. Указанный подход позволяет объединить преимущества моделей BSP и PRAM.

***Модель параллельных вычислений LogP*** была предложена группой авторов в работе [11] как некоторое усовершенствование модели BSP. В качестве критики модели BSP указывались следующие факторы. На каждом супершаге количество данных, обрабатываемых одним процессором, должно быть примерно равным количеству слов, получаемых в ходе $h$-сессии, то есть $h$, что ограничивает зернистость параллелизма снизу. Далее, сообщения, передаваемые в конце выполнения супершага, не могут быть использованы реципиентом до начала следующего супершага. И последнее, модель BSP предполагает аппаратную поддержку механизма глобальной синхронизации, однако большинство многопроцессорных систем с распределенной памятью не имеют подобных аппаратных средств.

По аналогии с моделью BSP модель LogP предполагает, что компьютер состоит из процессорных модулей с приватной памятью, соединенных коммуникационной сетью. Основными стоимостными параметрами модели являются следующие.

$L$ — (*latency*) верхняя граница латентности, представляющая собой время, необходимое для передачи сообщения длиной в одно машинное слово от одного процессорного модуля другому.

$o$ — (*overhead*) накладные расходы, представляемые как промежуток времени, в течении которого процессорный модуль занят приемом или передачей сообщения; в это время процессорный модуль не может выполнять никакую другую работу.

$g$ — (*gap*) задержка, представляющая собой минимальное время между двумя последовательными операциями чтения или передачи сообщений, выполняемыми процессорным модулем.

$P$ — количество процессорных модулей.

При этом предполагается, что коммуникационная сеть имеет ограниченную пропускную способность, позволяющую каждому процессорному модулю получать или посылать одновременно не более $\lceil L \,/\, g \rceil$ сообщений. Если процессорный модуль пытается передать сообщение, перекрывающее этот лимит, то он переводится в состояние ожидания до тех пор, пока сообщение можно будет послать, не выходя за границу пропускной способности сети. Базовый вариант модели LogP предполагает, что все сообщения имеют небольшой размер (одно или небольшое количество машинных слов). Большие сообщения необходимо фрагментировать.

Время выполнения алгоритма в модели LogP определяется как максимум временных затрат среди всех процессорных модулей, участвующих в вычислениях. Время передачи одного короткого сообщения от одного процессорного модуля другому составляет $o + L + o$. Время доступа к элементу данных, располагающемуся в памяти другого процессорного модуля, будет равно $2L + 4o$. Для последовательной передачи $n$ сообщений между процессорными модулями $P_1$ и $P_2$ может быть организован конвейер, как это показано на рис. 11. В этом случае передача $n$ последовательных сообщений займет время, равное $(n-1)g + o + L + o$ [74].

Сильной стороной модели LogP является ее простота. Однако, эта простота иногда может приводить к недостаточно точному предсказательному моделированию производительности алгоритмов и программ. Очевидным недостатком модели LogP является ограничение на размер сообщений. Попытка преодоления этого ограничения была сделана в ряде модификаций модели LogP, которые будут рассмотрены ниже.

В работе [75] было предложено расширение модели LogP путем добавления нового параметра $G$ (*Gap per Byte*), задающего время, необходимое для передачи одного байта в составе длинного сообщения. Новая модель получила название ***LogGP***. В модели LogGP время передачи сообщения длиной в $m$ байт вычисляется по следующей формуле:

$$T_n = o + (m-1)G + L + o. \tag{5}$$

Более радикальное расширение модели LogP предложено в статье [76], где описывается ***параметризованная модель PlogP***, расширяющая модель LogP путем введения дополнительных параметров. Модель PlogP лучше учитывает особенности коммуникационного программного обеспечения такого, как MPI [77, 78]. Среди этих особенностей выделяются следующие: 1) накладные расходы $o$ и задержка $g$ в значительной мере зависят от размеров сообщения; 2) накладные расходы, связанные с посылкой и приемом сообщения, могут существенно различаться, так как обработка асинхронно приходящих сообщений требует принципиально иной реализации, чем синхронные вызовы операции посылки.

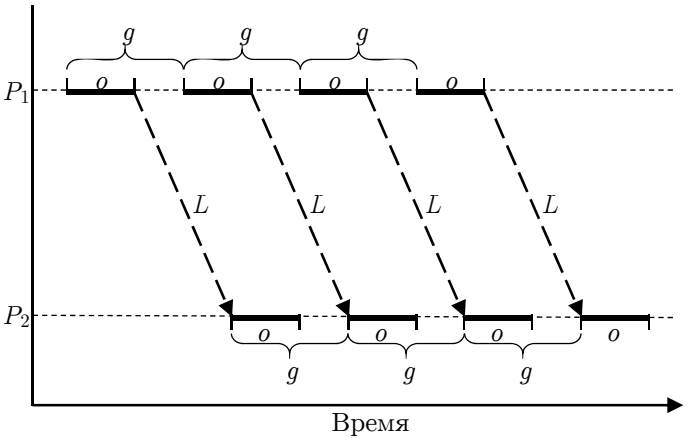


**Рис. 11**. Конвейерная передача сообщений в модели LogP

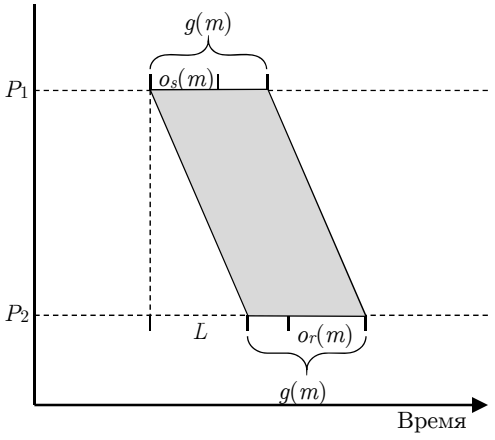


**Рис. 12**. Передача сообщения в модели PLogP

В соответствии с этим модель PLogP вместо параметров $o$ и $g$ вводит следующие параметры, зависящие от длины $m$ сообщения:

$o_s(m)$ — накладные расходы, связанные с посылкой сообщения длиной $n$.

$o_r(m)$ — накладные расходы, связанные с приемом сообщения длиной $n$.

$g(m)$ — задержка между двумя последовательными операциями чтения или передачи сообщений длиной $m$.

Семантика параметра $L$ (латентность) также несколько меняется. В модели PLogP латентность $L$ обозначает время, затрачиваемое на передачу первого бита сообщения. Параметр $P$ имеет тот же смысл, что и в модели LogP. Моделирование посылки сообщения длиной $m$ между процессорными модулями $P_1$ и $P_2$ в метрике модели PLogP проиллюстрировано на рис. 12. Таким образом, время, затрачиваемое на передачу одного сообщения длины $m$, составляет $T_1 = L + g(m)$.

Соотношения между параметрами моделей PLogP и LogP/LogGP приведены в табл. 1. При этом в качестве короткого сообщения берется один байт. Время передачи одного короткого сообщения в LogP/LogGP составляет $o + L + o$. В модели PLogP это время равно $L + g(1)$. Отсюда получается формула $o + L^{LogP/LogGP} + o = L^{PLogP} + g(1)$. Подставляя вместо $o$ значение $(o_s(1) + o_r(1)) / 2$, получаем итоговое соотношение $L^{LogP/LogGP} = L^{PLogP} + g(1) - o_s(1) - o_r(1)$.

**Таблица 1**

Соотношения между LogP/LogGP и PLogP

| **LogP/LogGP** | **PLogP** |
|---|---|
| $L$ | $L + g(1) - o_s(1) - o_r(1)$ |
| $o$ | $(o_s(1) + o_r(1)) / 2$ |
| $g$ | $g(1)$ |
| $G$ | $g(m) / m$ для $m \gg 1$ |
| $P$ | $P$ |

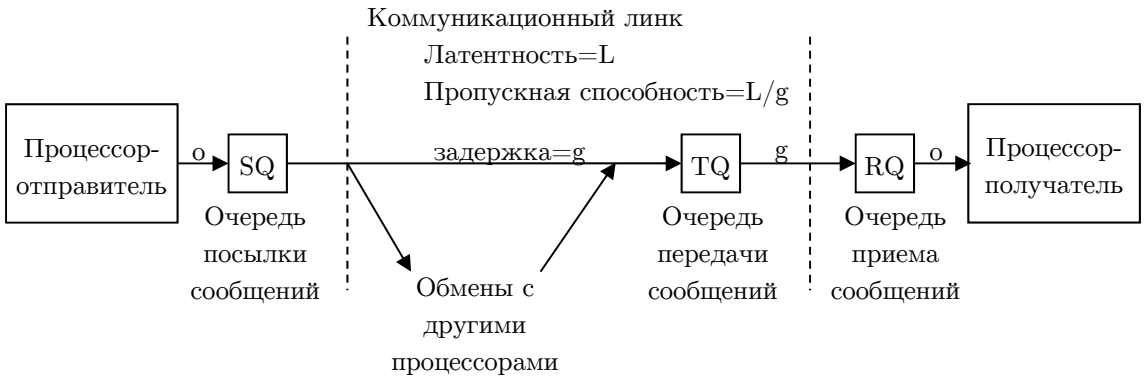


**Рис. 13.** Структура модели LogPQ

В работе [79] была предложена ***модель параллельных вычислений LogPQ***, расширяющая модель LogP путем введения дополнительных параметров, связанных с обслуживанием очередей при передачи сообщений: $SQ$ — размер очереди посылки сообщений, $RQ$ — размер очереди приема сообщений и $TQ$ — размер очереди передачи сообщений (см. рис. 13). Длина сообщений в модели LogPQ, также, как и в LogP, является постоянной и равна одному *коммуникационному* слову длиной $w_c$. Длинные сообщения представляются в виде последовательности сообщений длиной в одно *коммуникационное* слово. При этом длина $w_p$ машинного слова может не совпадать с $w_c$. Соотношения между параметрами моделей LogPQ и LogP приведены в табл. 2, где $r = w_p \,/\, w_c$; $L$ — латентность, представляющая собой время, необходимое для передачи сообщения длиной в одно коммуникационное слово; $o$ — накладные расходы, представляемые как промежуток времени, в течение которого процессорный модуль занят приемом или передачей сообщения; $g$ — временной интервал между двумя последовательными передачами сообщений от одного процессора другому.

Максимальная пропускная способность соединительной сети в модели LogPQ оценивается как $\lfloor L \,/\, g \rfloor + SQ + RQ$. Модель LogP, таким образом, может рассматриваться как частный случай модели LogP при $SQ = TQ = RQ = 1$. В работе [80] модель LogPQ была использована для оценки времени выполнения параллельного алгоритма перемножения матриц размера $n \times n$ на массивно-параллельном компьютере CM-5 [81]. Вычислительные эксперименты показали, что модель LogPQ более точно предсказывает реальное время выполнения программы по сравнению с LogP. Однако разница между двумя моделями оказалась существенной только для матриц небольшого размера ($n < 32$). Кроме этого, модель LogPQ оказалось сложно адаптировать к новым параллельным вычислительным архитектурам, и она не получила широкого распространения.

**Таблица 2**

Соответствие параметров LogPQ и LogP

| **LogPQ** | **LogP** |
|---|---|
| $L$ | $L + o + (r-1)g$ |
| $o$ | $o + (r-1)g$ |
| $g$ | $rg$ |
| $P$ | $P$ |

***Модель параллельных вычислений LogGPS***, представленная в работе [82], расширяет модель LogGP путем введения дополнительного параметра $S$, отражающего затраты на синхронизацию. Фактически модель LogGPS явилась адаптацией модели LogPQ к особенностям коммуникационных протоколов библиотеки MPICH [83], представляющей собой переносимую реализацию стандарта MPI. Еще одним расширением модели LogGP является *модель параллельных вычислений LoGPC* [84, 85], которая учитывает накладные расходы, связанные с конфликтами, возникающими при обмене сообщениями в мультипроцессоре с распределенной памятью, а также конвейерную передачу длинных сообщений с использованием интерфейса DMA [86, 87]. Модель LoGPC является более точной по сравнению с LogGP, однако это достигается ценой значительного усложнения анализа алгоритмов. Кроме этого, LoGPC существенно зависит от аппаратных особенностей соединительной сети.

## 5. Многоуровневые модели с распределенной памятью

***Модель параллельных вычислений $\log_n P$*** [88, 89] является расширением модели LogP. Модель $\log_n$P ориентирована на кластерные архитектуры, в которых передача сообщений между узлами с иерархической памятью осуществляется с помощью программного обеспечения промежуточного слоя такого, как MPI. Кроме явных затрат на передачу сообщений, учитываемых моделью LogP, модель $\log_n$P учитывает также неявные затраты, связанные с передачами данных между различными уровнями иерархической памяти, которые выполняются программным обеспечением промежуточного слоя при организации пересылок между процессорными узлами. Экспериментально было показано, что в некоторых приложениях при передаче сильно фрагментированных данных большого объема неявные затраты могут в несколько раз превосходить явные затраты на пересылку типа точка-точка. Стоимостная метрика модели $\log_n$P базируется на следующих пяти параметрах.

- $l$ — *эффективная латентность* (*effective latency*), определяемая как разница между временем, затрачиваемым процессором на передачу фрагментированных и нефрагментированных данных. Эффективная латентность может быть вычислена с помощью платформенно-зависимой функции $f(s,d) = l$, где $s$ — длина одного сообщения при передаче массива, $d$ — длина пропусков между единичными сообщениями в передаваемом массиве.
- $o$ — *эффективные накладные расходы* (*effective overhead*) определяются как время, затрачиваемое процессором на передачу единичного сообщения в случае нефрагментированных данных ($d=1$). Эффективные накладные расходы могут быть вычислены с помощью платформенно-зависимой функции $f(s,1) = o$.
- $g$ — (*gap*) задержка, представляющая собой минимальное время между двумя последовательными операциями передачи сообщений. В модели $\log_n$P предполагается, что $g=o$.
- $n$ — количество уровней иерархической памяти, используемых программным обеспечением промежуточного слоя при передаче сообщений. Фактически $n$ определяет количество неявных обменов данными между различными уровнями иерархической памяти.
- $P$ — количество процессорных модулей.

Время, затрачиваемое на передачу одного сообщения в модели $\log_n$P определяется по формуле

$$T = \sum_{i=0}^{n-1}(o_i + l_i) = \sum_{i=0}^{n-1}(f_i(s,1) + f_i(s,d)). \quad (6)$$

Частным случаем модели $\log_n$P является модель $\log_3$P ( $n = 3$ ), ориентированная на кластеры с SMP-узлами. В этом случае предыдущая формула преобразуется к виду

$$T = \sum_{i=0}^{2}(o_i + l_i).$$

Заметим, что при $n=1$ модель $\log_n$P эквивалентна модели *Memory logP*, представленной в более ранней работе [90].

***Модель параллельных вычислений HiHCoHP*** **(*Hierarchical Hyper-Clusters of Heterogeneous Processors*)** [91, 92] ориентирована на моделирование многоуровневых иерархических сетей из гетерогенных кластеров. На пути от отправителя $P_s$ к адресату $P_r$ сообщение пересекает несколько уровней в иерархии сетей, упорядоченных по возрастанию отношения латентности к пропускной способности. Коммуникационная стоимость вычисляется как функция, зависящая от стоимостей $\sigma_s^{(k)}$ и $\sigma_r^{(k)}$ обработки сообщения на уровне $k$, стоимостей $\pi_s^{(k)}$ и $\pi_r^{(k)}$ упаковки и распаковки каждого пакета сообщения, и следующих трех параметров: 1) латентности $\lambda^{(k)}$, которая включает в себя латентности уровней ниже $k$; 2) пропускной способности $\beta^{(k)}$, включающей в себя пропускную способность сетей более низкого уровня; 3) ширины канала $\kappa^{(k)}$, вычисляемой как максимальное число пакетов, передаваемых за один такт по сети уровня $k$. Время передачи сообщения длиной $m$ вычисляется по формуле $T(m) = \sigma_s^{(k)} + \pi_s^{(k)}p + \lambda^{(k)} + \frac{p-1}{\beta^{(k)}} + \sigma_r^{(k)} + \pi_r^{(k)}p$, где $p$ — количество процессов. Модель HiHCoHP также включает параметр $\rho_i$, обозначающий процессорное время, затрачиваемое узлом $P_i$ на выполнение единичного объема вычислений.

***Модель параллельных вычислений D-BSP*** **(*Decomposable Bulk Synchronous Parallel*)** [17] является расширением модели BSP. Модель D-BSP представляет многопроцессорную вычислительную систему как совокупность кластеров, каждый из которых имеет свою внутреннюю соединительную сеть, характеризующуюся определенной пропускной способностью и задержкой при передаче сообщений. Общее количество $n$ процессоров в системе предполагается равным степени числа два. Для фиксированного $i$, такого, что $0 \le i \le \log_2 n$, $n$ процессоров делятся на $2^i$ непересекающихся *i-кластеров* $C_0^{(i)}, \ldots, C_{2^i-1}^{(i)}$ по $n/2^i$ процессоров в каждом. Процессоры каждого *i*-кластера могут обмениваться сообщениями независимо от остальных. Структура D-BSP машины формируется как бинарное дерево высоты $\log_2 n$, узлами которого являются *i*-кластеры. Указанное бинарное дерево обладает следующими свойствами: кластер $C_j^{\log_2 n}$ содержит только один процессор $P_j$ ( $0 \le j < n$ ); $C_j^{(i)} = C_{2j}^{(i+1)} \cup C_{2j+1}^{(i+1)}$ для $0 \le i < \log_2 n$ и $0 \le j < 2^i$. Выполне-

ние программы в модели D-BSP состоит из последовательности *мегашагов*. Каждый мегашаг представляет собой последовательность $1+\log_2 n$ супершагов. В рамках $i$-того супершага процессоры выполняют некоторые вычисления и обмениваются данными исключительно в пределах их собственного $i$-кластера. В финале $i$-того супершага в каждом $i$-кластере выполняется барьерная синхронизация. Если на $i$-том супершаге каждый процессор выполняет не более $w$ операций и передает или получает не более $h$ сообщений, то время выполнения этого супершага оценивается по формуле $w+hg_i+l_i$, где $g_i$ — задержка, а $l_i$ — латентность для $i$-того уровня.

В работе [93] предлагается ***модель параллельных вычислений HLogGP (Heterogeneous LogGP)***, являющаяся расширением модели LogGP и ориентированная на гетерогенные вычислительные кластеры. Модель HLogGP допускает наличие в кластере вычислительных узлов, имеющих различающиеся процессоры, модули памяти и сетевые адаптеры. Основная идея — заменить скалярные параметры на векторы и матрицы для учета специфики разнородных узлов кластерной вычислительной системы. Для моделирования гетерогенной вычислительной системы с $M$ узлами вводится пять параметров: латентность $L$ и байтовая задержка $G$ представляют собой матрицы размера $M\times M$; величина накладных расходов $o$, задержка между сообщениями $g$ и вычислительная мощность $P$ являются векторами длины $M$. Если все эти параметры определены с достаточной точностью, модель HLogGP может достоверно предсказывать производительность параллельных алгоритмов, в которых коммуникационные затраты превалируют над вычислительными.

***Модель параллельных вычислений LogfP***, предложенная в [94], расширяет модель LogP на кластерные вычислительные системы, построенные с использованием соединительной сети InfiniBand [95]. Модель LogfP предполагает, что первые $f$ коротких сообщений передаются с нулевыми накладными расходами. Способность InfiniBand мгновенно пересылать сразу несколько коротких сообщений (в количестве, не превышающем $f$, после чего начинает наблюдаться замедление) обнуляет параметр $g$. Более того, использование в сети InfiniBand технологии RDMA (Remote Direct Memory Access) при передаче сообщений делает неадекватным параметр $o_R$, определяющий накладные расходы на стороне получателя. Параметры модели LogfP замеряются с помощью эталонного теста RTT (Round-Trip Time) по схеме $1:P$ — $P:1$. Модель LogfP позволяет повысить оценку производительности барьерной синхронизации на 40% по сравнению с другими известными моделями.

***Модель параллельных вычислений PLP*** [96] ориентирована на иерархические гетерогенные вычислительные системы с однопроцессорными узлами и соединительной сетью Ethernet. Время передачи сообщения длиной $m$ байт по схеме точка-точка вычисляется по формуле $T(m)=L+P_s+P_r$. Параметры $P_s$ и $P_r$ задают время, затрачиваемое процессорами отправителя и получателя на организацию передачи сообщения. Эти параметры в свою очередь зависят от физических параметров конкретных процессоров. Латентность $L$ определяется следующим образом: $L=(n-1)G+L_t$, где $G$ — межкадровая задержка при отправке $n$ последовательных Ethernet-кадров, на которые разбивается сообщение, и $L_t$ — латентность при кадровой передаче.

Модель PLP применима как для обменов типа точка-точка, так и для коллективных коммуникаций.

***Модель параллельных вычислений LogGOPS***, предложенная в работе [97], представляет собой расширение модели LogGPS и предназначена для оценки масштабируемости параллельных алгоритмов, ориентированных на большие вычислительные системы, работающие под управлением MPI. Согласно этой модели, стоимость передачи сообщения длиной $m$ байт составляет $2o + L + (m-1)O + (m-1)G$ . Параметры $o, L, G$ имеют тот же смысл, что и в модели LogGPS. Параметр $O$ является специфичным для модели LogGOPS и обозначает накладные расходы в пересчете на байт. Модель LogGOPS применяется для линейных коммуникаций типа scatter (один посылает всем) и gather (один получает от всех), для широковещательных коммуникаций на основе бинарного дерева и для распределенных коммуникаций, включающих в себя схему «всем от всех» и барьерную синхронизацию.

*Модель параллельных вычислений LogGPH* [12] является расширением модели LogGP и ориентирована на многопроцессорные системы, построенные на основе иерархии соединительных сетей. Для каждого уровня в сетевой иерархии вводится свой вектор параметров. Два процесса, запущенные на одном процессоре, взаимодействуют на уровне 1 и передают друг другу сообщения через общую память. Процессы, запущенные на разных процессорах одного узла, взаимодействуют на уровне 2. Процессы, запущенные на разных процессорных узлах, взаимодействуют на уровне 3, обмениваясь сообщениями через соединительную сеть. Время передачи сообщения длины $m$ на сетевом уровне $i$ вычисляется по формуле $T_i(m) = 2o_i + (m-1)G_i + L_i$.

В работе [98] предложена *модель параллельных вычислений Multi-BSP*, расширяющая модель BSP в двух направлениях. Первое, Multi-BSP является иерархической моделью с произвольным количеством уровней, отражающих реальные технические особенности иерархической памяти и различных уровней кэш-памяти современных многопроцессорных систем. Второе, в качестве дополнительного параметра Multi-BSP включает в себя объем памяти на каждом уровне. Для каждого уровня $i$ в иерархии вводится вектор параметров $(p_i, g_i, L_i, m_i)$, где $p_i$ — количество процессоров, $g_i$ — задержка при передаче сообщения, $L_i$ — затраты на синхронизацию, $m_i$ — объем памяти/кэша. Для системы, включающей в себя $d$ уровней, общее количество процессоров вычисляется по формуле $P_d = \prod_{i=1}^{d} p_i$, суммарный объем памяти вычисляется по формуле $M_d = m_d + \sum_{i=1}^{d-1} m_i \prod_{j=i+1}^{d} p_j$, суммарная задержка вычисляется по формуле $G_d = \sum_{i=1}^{d} g_i$.

***Модель mlog$_n$P*** [99, 100] является расширением модели log$_n$P, ориентированным на вычислительные кластеры с многоядерными процессорами. Иерархию $n$-уровневой памяти, введенную в модели log$_n$P, авторы модели mlog$_n$P называют вертикальной. Дополнительно они вводят горизонтальную $m$-уровневую иерархию каналов передачи данных. По каналу нулевого уровня происходит обмен данными между ядрами одного процессора, канал первого уровня используется для обмена данными между ядрами разных процессоров одного процессорного узла, канал второго уровня — для обмена данными между ядрами разных процессорных узлов, и так далее. В соответствии с этим стоимость передачи сообщения на горизонтальном уровне $i$ вычисляется по формуле

$$T_i = \sum_{j=0}^{n_i-1}(o_j^i + l_j^i) = \sum_{j=0}^{n_i-1}(f_j^i(s,1) + f_j^i(s,d)), \tag{7}$$

где семантика всех параметров с точностью до горизонтального уровня наследуется от модели $\log_n$P.

***Модель параллельных вычислений SLOWER***, предложенная в работе [101], ориентирована на экзамасштабные вычислительные системы. Главной целью модели SLOWER является оценка производительности $P$ вычислительной системы, которая вычисляется по формуле

$$P = e(L,O,W) \times s(S) \times \mu(E) \times a(R), \tag{8}$$

где $e$ обозначает эффективность $(0 < e < 1)$, $L$ — латентность, $O$ — накладные расходы, $W$ — задержка, вызванная ожиданием доступа к разделяемым ресурсам, $s$ — степень параллелизма, $S$ — мера недозагрузки, $\mu$ — скорость выполнения потока машинных команд, $E$ потребление энергии, $a$ — доступность, $R$ — надежность. Формула (8) включает в себя критические с точки зрения модели SLOWER параметры и показывает их взаимное влияние на общую производительность системы. По мнению авторов модели SLOWER эффективность $e$ и доступность $a$ не оказывают значительного влияния на общую производительность системы. Мера доступности $a$ отражает эксплуатационные проблемы, препятствующие способности системы выполнять реальную вычислительную работу. Она зависит от параметра $R$, определяющего среднее время между отказами системы. Пиковая скорость выполнения потока машинных команд $\mu$ зависит от тактовой частоты, которая в определенном диапазоне прямо пропорциональна энергопотреблению $E$. Степень параллелизма $s$ определяется количеством процессорных ядер, одновременно выделяемых для решения задачи. Мера недозагрузки $S$ вычисляется как отношение времени простоя процессорного ядра ко времени его работы при решении задачи. Недозагруженность возникает либо когда процессорное ядро простаивает в ожидании работы или необходимых данных, либо в результате дисбаланса в распределении вычислительной работы между процессорными ядрами. Эффективность $e$ вычислительной системы определяется как отношение скорости непрерывного выполнения машинных команд к пиковой скорости. Как следует из формулы (8), на эффективность влияет следующий ряд факторов. Латентность $L$ представляет собой время, необходимое для инициализации доступа к удаленным данным или сервисам в незагруженной системе. Задержка $W$ (ожидание отложенного доступа) возникает из-за конфликта доступа к разделяемым логическим или физическим ресурсам в загруженной системе. Накладные расходы $O$ включают в себя процессорное время, которое тратится на управление параллельными ресурсами и диспетчеризацию процессов. Модель SLOWER может применяться к широкому классу вычислительных систем экзафлопсного уровня производительности, однако в настоящее время отсутствует математическая формализация этой модели.

***Модель параллельных вычислений HLog$_n$GP (Heterogeneous Log$_n$GP)***, предложенная в работе [13], развивает идеи, заложенные в моделях *LogGP* и *log$_n$P*. Модель HLog$_n$GP ориентирована на кластерные вычислительные системы с графическими ускорителями [26]. Модель HLognGP предполагает, что передача данных в ГПУ-кластере может происходить на трех различных уровнях: через оперативную память, через шину PCI

и через соединительную сеть. В общем случае полагается, что количество таких *атомарных* коммуникационных уровней равно некоторой положительной константе $n$. В соответствии с этим стоимостная метрика модели $\mathrm{HLog_nGP}$ строится на основе следующих шести параметров:

- $n$ — количество атомарных уровней при передаче сообщения от одного процессорного узла другому;
- $L$ — латентность, представляющая собой вектор латентностей $(l_1,\ldots,l_n)$ для различных атомарных коммуникационных уровней;
- $o$ — накладные расходы, представляющие собой вектор накладных расходов $(o_1,\ldots,o_n)$ для каждого атомарного коммуникационного уровня;
- $g$ — (*gap*) задержка, представляющая собой вектор $(g_1,\ldots,g_n)$, определяющий для каждого атомарного коммуникационного уровня минимальное время между двумя последовательными операциями передачи сообщений;
- $G$ — (*gap per byte*) задержка на байт, представляющая собой вектор $(G_1,\ldots,G_n)$, определяющий для каждого атомарного коммуникационного уровня время, необходимое для передачи одного байта в составе длинного сообщения;
- $P$ — вычислительная производительность, представляющая собой пару $\left(P_C,P_G\right)$, определяющую производительность ЦПУ и ГПУ соответственно.

Частный вид модели *HLog₃GP* имеет три атомарных коммуникационных уровня и соответствует ГПУ- кластерам. В этом случае время, необходимое для передачи одного сообщения между процессорными узлами, оценивается по формуле

$$T_{comm} = \sum_{k=1}^{3} L_k + 2\max(o_k, g_k) + (s-1)G_k, \tag{9}$$

где $s$ обозначает длину сообщения в байтах.

***Модель параллельных вычислений MBSP* (*Multi-memory BSP*)** [102] является расширением модели BSP и ориентирована на многопроцессорные системы с многоуровневой иерархической памятью и многоядерными процессорами. Стоимостная метрика модели MBSP базируется на семи параметрах $(p,l,g,m,L,G,M)$, где параметры $(p,l,g)$ наследуются от модели BSP, а $(m,L,G,M)$ являются новыми параметрами, специфицирующими иерархическую память. Параметр $M$ задает объем приватной «быстрой» памяти, имеющейся у каждого процессора. Параметр $m$ задает объем «медленной» памяти, доступной всем ядрам процессорного узла. Стоимость доступа к медленной памяти определяется парой $(L,G)$. Параметр $G$ определяет время доступа к медленной памяти в пересчете на одно машинное слово. Параметр $L$ задает латентность при обменах с медленной памятью. Время выполнения супершага вычисляется следующим образом: $\max(l,L) + w + gh + Gb$, где $w$ — максимальное время вычислений, выполняемых одним процессорном, $h$ — максимальный размер сообщения, передаваемого или получаемого процессором, и $b$ — максимальный объем обменов с медленной памятью, выполняемых одним процессором в ходе супершага.

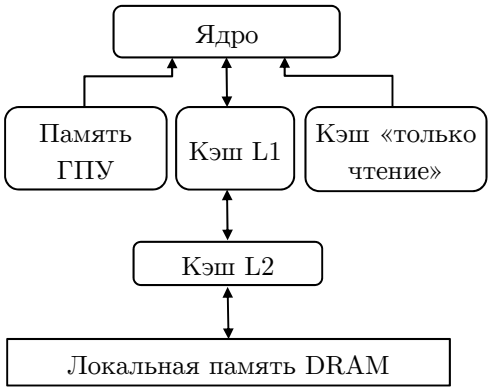


**Рис. 14**. Иерархия памяти в ГПУ с архитектурой Kepler

В работе [103] предложено ***расширение модели BSP для ГПУ под управлением CUDA*** (далее для краткости — CUDABSP). Модель CUDABSP ориентирована на предсказание времени выполнения CUDA-программы на кластерных вычислительных системах с графическими ускорителями архитектуры Kepler (см. рис. 14) на основе оценок затрат на коммуникации и вычисления, которые выполняются независимо. Время, расходуемое на вычисления, в свою очередь, оценивается на основе работы одного процессорного ядра и вычисляется по формуле

$$T_k = \frac{t \cdot (Comp + Comm_{DRAM} + Comm_{GPUM})}{R \cdot P \cdot \lambda}, \tag{10}$$

где $t$ — количество нитей, выполняемых на ядре, $Comp$ — время, расходуемое нитью на вычисления, $Comm_{DRAM}$ — затраты на обмены с локальной памятью DRAM процессорного узла, $Comm_{GPUM}$ — затраты на обмены с памятью ГПУ, $R$ — тактовая частота, $P$ — количество ядер в ГПУ, $\lambda$ — положительное число, больше либо равное единицы, моделирующее эффект аппаратной оптимизации. Затраты на глобальные коммуникации оцениваются также, как в модели BSP.

## 6. Другие обзоры

Одним из первых обзоров моделей параллельных вычислений была работа [104], опубликованная Маггсом (Maggs), Матесоном (Matheson) и Тарьяном (Tarjan) в 1995 г. В данном обзоре рассматриваются модели как для общей, так и для распределенной памяти, включая модели PRAM, BSP, P-HMM, P-BT и некоторые другие.

Скилликорн (Skillicorn) и Талья (Talia) в обзоре [9] 1998 года рассматривают средства параллельного программирования в целом, включая сюда, помимо моделей, языки, фреймворки и технологии параллельного программирования. Они формулируют требования к инструментам параллельных вычислений (легкость программирования, легкость понимания, архитектурная независимость, гарантированная производительность, наличие стоимостных метрик и др.) и затем в контексте этих требований рассматривают широкий спектр известных инструментов, включая формализм Бирда—Мертенса, языки программирования Haskell, параллельный Prolog, Modula 3*, Occam, Linda, математические модели параллельных вычислений BSP, LogP, PRAM, библиотеки PVM, MPI и многие другие инструменты.

Грама (Grama), Кумар (Kumar), Ранка (Ranka) и Сингх (Singh) в 2001 году опубликовали обзор [35], в котором предложили свою иерархическую таксономию моделей параллельных вычислений в виде бинарного дерева. На верхнем уровне они поставили следующие два класса: «p-ported global memory» (общая память с параллельным многоканальным доступом) и «p-single ported memories» (распределенная память, состоящая из блоков с одноканальным доступом). Каждый из этих классов делится на два подкласса: «Bulk Access Locality» (блочный доступ к участку памяти) и «No Bulk Access Locality» (отсутствие блочного доступа к участку памяти). Блочный доступ к участку памяти предполагает наличие механизма передачи длинной последовательности байт, характеризующегося наличием латентности, предшествующей реальной пересылке данных. На третьем уровне иерархии вводится еще два субкласса: «No Data Volume Locality» (отсутствие разницы во времени доступа к различным областям памяти) и «Data Volume Locality» (наличие разницы во времени доступа к различным областям памяти). Далее авторы рассматривают и классифицируют известные модели параллельных вычислений. К настоящему времени этот обзор, однако, устарел, так как в нем отсутствуют модели, ориентированные на современные кластерные вычислительные системы с многоядерными ускорителями.

Обзор [1], опубликованный Жангом (Zhang), Ченом (Chen), Суном (Sun) и Миао (Miao) в 2007 г., является обновленной версией обзора [104]. Авторы разбивают все модели параллельных вычислений на три класса: модели с общей памятью, модели с распределенной памятью и модели с иерархической памятью. К моделям с общей памятью они относят PRAM, YPRAM и HPRAM. В класс моделей с распределенной памятью попадают BSP, BSPRAM, LogP, LogGP и некоторые другие. Под системами с иерархической памятью авторы понимают однопроцессорные и многопроцессорные системы с многоуровневой памятью, в которых доступ к каждому уровню различается по времени. В качестве таких уровней могут фигурировать регистровая память, кэш первого уровня, кэш второго уровня, память DRAM и так далее. К моделям с иерархической памятью авторы обзора отнесли такие разные модели, как P-HMM, UPMH, $\log_n$P, LogP-HMM и др. Данный подход к классификации моделей параллельных вычислений представляется не вполне корректным, так как измерения «общая — разделяемая память» и «одноуровневая — многоуровневая память» являются ортогональными.

Наиболее свежим является обзор [105] группы авторов (Rico-Gallego, Díaz-Martín, Manumachu, Lastovetsky), опубликованный в «ACM Computing Surveys» в 2019 г. Указанный обзор включает в себя 25 различных моделей параллельных вычислений, которые классифицируются по целевым аппаратным платформам (кластеры с однопроцессорными узлами, кластеры с гетерогенными узлами и гомогенной соединительной сетью, гетерогенные кластеры, системы на основе двухмерной соединительной сети, Myrinet-кластеры, многоуровневые гетерогенные кластеризованные грид-системы, многопроцессорные системы с сетью lnfiniband, иерархические кластеры) и по следующим типам: оригинальность (основополагающие модели; производные модели), параметризация (модели, учитывающие параметры аппаратной платформы; модели, учитывающие параметры программного обеспечения промежуточного слоя), универсальность (платформенно-зависимые модели; платформенно-независимые модели). К недостаткам данного обзора следует отнести то, что авторы концентрируются на аспектах, связанных с затратами на коммуникации, отдавая предпочтение моделям, ориентированным на приложения, в которых затраты на коммуникации превалируют над затратами на вычисления.

## Заключение

В данном обзоре сделана попытка рассмотреть и классифицировать все значимые модели параллельных вычислений, известные к настоящему моменту. Рассмотрены аспекты, связанные как с оценкой времени вычислений, так и с оценкой затрат на межпроцессорные коммуникации. Вместе с непрерывным усложнением архитектур современных многопроцессорных систем неизбежно растет сложность параллельных вычислительных моделей, стремящихся как можно более точно предсказать время выполнения параллельного алгоритма на целевой аппаратной платформе. Однако сложные модели трудно использовать на практике. С другой стороны, простые классические модели типа BSP или LogP уже не могут обеспечить достаточную адекватность применительно к современным вычислительным кластерам и другим многопроцессорным платформам. Возможный выход из сложившейся ситуации заключается в том, что для создания современных простых и адекватных моделей параллельных вычислений необходимо ограничивать не только класс аппаратной платформы, но и класс алгоритмов, для которых предназначена модель.

*Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00352 а, Правительства РФ в соответствии с Постановлением №211 от 16.03.2013 г. (соглашение № 02.A03.21.0011) и Министерства образования и науки РФ (государственное задание 2.7905.2017/8.9).*

## Литература/References

1. Zhang Y. et al. Models of Parallel Computation: a Survey and Classification // Frontiers of Computer Science in China. Higher Education Press, 2007. Vol. 1, No. 2. P. 156–165. DOI: 10.1007/s11704-007-0016-1.
2. Valiant L.G. A Bridging Model for Parallel Computation // Communications of the ACM. 1990. Vol. 33, No. 8. P. 103–111. DOI: 10.1145/79173.79181.
3. Campbell D.K.G. A Survey of Models of Parallel Computation. Technical Report No.YCS-97-278. 1997. 37 p.
4. Shepherdson J.C., Sturgis H.E. Computability of Recursive Functions // Journal of the ACM. ACM, 1963. Vol. 10, No. 2. P. 217–255. DOI: 10.1145/321160.321170.
5. Elgot C.C., Robinson A. Random-Access Stored-Program Machines, an Approach to Programming Languages // Journal of the ACM. ACM, 1964. Vol. 11, No. 4. P. 365–399. DOI: 10.1145/321239.321240.
6. Hartmanis J. Computational Complexity of Random Access Stored Program Machines // Mathematical Systems Theory. Springer-Verlag, 1971. Vol. 5, No. 3. P. 232–245. DOI: 10.1007/BF01694180.
7. Cook S.A., Reckhow R.A. Time Bounded Random Access Machines // Journal of Computer and System Sciences. Academic Press, 1973. Vol. 7, No. 4. P. 354–375. DOI: 10.1016/S0022-0000(73)80029-7.
8. Aho A. V., Hopcroft J.E., Ullman J.D. The Design and Analysis of Computer Algorithms. London, Amsterdam, Don Mills, Ontario, Sydney: Addison-Wesley, 1974. 470 p.
9. Skillicorn D.B., Talia D. Models and Languages for Parallel Computation // ACM Computing Surveys. 1998. Vol. 30, No. 2. P. 123–169. DOI: 10.1145/280277.280278.

10. Fortune S., Wyllie J. Parallelism in Random Access Machines // Proceedings of the Tenth Annual ACM Symposium on Theory of Computing — STOC'78. New York, New York, USA: ACM Press, 1978. P. 114–118. DOI: 10.1145/800133.804339.
11. Culler D. et al. LogP: Towards a Realistic Model of Parallel Computation // Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming — PPOPP'93. New York, New York, USA: ACM Press, 1993. P. 1–12. DOI: 10.1145/155332.155333.
12. Yuan L. et al. LogGPH: A Parallel Computational Model with Hierarchical Communication Awareness // Proceedings of the 2010 13th IEEE International Conference on Computational Science and Engineering — CSE'10. Washington, DC, US: IEEE Computer Society, 2010. P. 268–274. DOI: 10.1109/CSE.2010.40.
13. Lu F., Song J., Pang Y. HLognGP: A Parallel Computation Model for GPU clusters // Concurrency and Computation: Practice and Experience. 2015. Vol. 27, No. 17. P. 4880-4896. DOI: 10.1002/cpe.3475.
14. Qiao X., Chen S., Yang L.T. HPM: a Hierarchical Model for Parallel Computations // International Journal of High Performance Computing and Networking. 2004. Vol. 1, No. 1–3. P. 117–127. DOI: 10.1504/IJHPCN.2004.007571.
15. Rico-Gallego J.-A., Díaz-Martín J.-C. τ-Lop: Modeling Performance of Shared Memory MPI // Parallel Computing. North-Holland, 2015. Vol. 46. P. 14–31. DOI: 10.1016/J.PARCO.2015.02.006.
16. Rico-Gallego J.-A., Lastovetsky A.L., Diaz-Martin J.-C. Model-Based Estimation of the Communication Cost of Hybrid Data-Parallel Applications on Heterogeneous Clusters // IEEE Transactions on Parallel and Distributed Systems. 2017. Vol. 28, No. 11. P. 3215-3228. DOI: 10.1109/TPDS.2017.2715809.
17. Bilardi G. et al. On the Effectiveness of D-BSP as a Bridging Model of Parallel Computation // Proceedings of the International Conference on Computational Science — ICCS'01. Part II. Lecture Notes in Computer Science, Vol. 2074. Berlin, Heidelberg: Springer, 2001. P. 579–588. DOI: 10.1007/3-540-45718-6_63.
18. Ежова Н.А., Соколинский Л.Б. Модель параллельных вычислений для многопроцессорных систем с распределенной памятью // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. Том 7, № 2. С. 32–49. DOI: 10.14529/cmse180203. [Ezhova N.A., Sokolinsky L.B. Parallel Computational Model for multiprocessor Systems With Distributed Memory. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2018. vol. 7, no. 2. pp. 32–49. (in Russian) DOI: 10.14529/cmse180203.]
19. Ежова Н.А., Соколинский Л.Б. Исследование масштабируемости итерационных алгоритмов при суперкомпьютерном моделировании физических процессов // Вычислительные методы и программирование. 2018. Том 19, № 4. С. 416–430. DOI: 10.26089/NumMet.v19r437. [Ezhova N.A., Sokolinsky L.B. Scalability Evaluation of Iterative Algorithms for Supercomputer Simulation of Physical Processes. *Numerical methods and programming.* 2018. vol. 19, no. 4. pp. 416–430. (in Russian) DOI: 10.26089/NumMet.v19r437.]
20. Ceze L.H. Shared-Memory Multiprocessors // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 1810–1812. DOI: 10.1007/978-0-387-09766-4_142.
21. Nayfeh B.A., Olukotun K. A Single-chip Multiprocessor // Computer. 1997. Vol. 30, No. 9. P. 79–85. DOI: 10.1109/2.612253.
22. Bardine A. et al. NUMA Caches // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 1329–1338. DOI: 10.1007/978-0-387-09766-4_16.

23. Snir M. Distributed-Memory Multiprocessor // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 574–578.
24. Pfister G.F. In Search of Clusters. 2nd Edition. Upper Saddle River, NJ: Prentice Hall, 1998. 575 p.
25. Beowulf Cluster Computing with Linux / ed. Sterling T.L. Cambridge, London: MIT Press, 2002. 496 p.
26. Owens J.D. et al. GPU Computing // Proceedings of the IEEE. 2008. Vol. 96, No. 5. P. 879–899. DOI: 10.1109/JPROC.2008.917757.
27. Rochange C., Uhrig S., Sainrat P. Memory Hierarchy // Time-Predictable Architectures. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2014. P. 69–104. DOI: 10.1002/9781118790229.ch4.
28. Hennessy J.L., Patterson D.A. Computer Architecture: A Quantitative Approach // Computer. Fifth Edit. Morgan Kaufmann, 2011. 856 p.
29. Bottomley J. Understanding Caching // Linux Journal. 2004. No. 117. P. 58–62.
30. Wu K. et al. Early Evaluation of Intel Optane Non-Volatile Memory with HPC I/O Workloads // arXiv:1708.02199v2 [cs.DC]. 2017. 6 p.
31. Yang C.-T., Huang C.-L., Lin C.-F. Hybrid CUDA, OpenMP, and MPI Parallel Programming on Multicore GPU Clusters // Computer Physics Communications. North-Holland, 2011. Vol. 182, No. 1. P. 266–269. DOI: 10.1016/J.CPC.2010.06.035.
32. Bilardi G., Pietracaprina A. Models of Computation, Theoretical // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 1150–1158. DOI: 10.1007/978-0-387-09766-4_218.
33. Skillicorn D.B. Parallelism and the Bird-Meertens Formalism. Kingston, Canada, 1992. 16 p.
34. Bilardi G., Pietracaprina A., Pucci G. A Quantitative Measure of Portability with Application to Bandwidth-Latency Models for Parallel Computing // Euro-Par'99 Parallel Processing. Euro-Par 1999. Lecture Notes in Computer Science, Vol 1685. Springer, Berlin, Heidelberg, 1999. P. 543–551. DOI: 10.1007/3-540-48311-X_76.
35. Grama A. et al. Architecture Independent Analysis of Parallel Programs // Proceedings of the International Conference on Computational Science — ICCS'01. Part II. Lecture Notes in Computer Science, Vol. 2074. Berlin, Heidelberg: Springer, 2001. P. 599–608. DOI: 10.1007/3-540-45718-6_65.
36. JaJa J.F. PRAM (Parallel Random Access Machines) // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 1608–1615. DOI: 10.1007/978-0-387-09766-4_23.
37. Goldschlager L.M. A Unified Approach to Models of Synchronous Parallel Machines // Proceedings of the Tenth Annual ACM Symposium on Theory of Computing — STOC'78. New York, New York, USA: ACM Press, 1978. P. 89–94. DOI: 10.1145/800133.804336.
38. Ladner R.E., Fischer M.J. Parallel Prefix Computation // Journal of the ACM. 1980. Vol. 27, No. 4. P. 831–838. DOI: 10.1145/322217.322232.
39. JaJa J.F. An Introduction to Parallel Algorithms. Redwood City, CA, USA: Addison Wesley Publishing Co., Reading, 1992. 576 p.
40. Darema F. et al. A Single-Program-Multiple-Data Computational Model for EPEX/FORTRAN // Parallel Computing. 1988. Vol. 7, No. 1. P. 11–24. DOI: 10.1016/0167-8191(88)90094-4.
41. Darema F. SPMD Computational Model // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 1933–1943. DOI: 10.1007/978-0-387-09766-4_26.
42. Cook S., Dwork C., Reischuk R. Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes // SIAM Journal on Computing. Society

for Industrial and Applied Mathematics, 1986. Vol. 15, No. 1. P. 87–97. DOI: 10.1137/0215006.

43. Karp R.M., Ramachandran V. Parallel Algorithms for Shared-Memory Machines // Handbook of theoretical computer science. Volume A: Algorithms and Complexity / ed. Van Leeuwen J. Amsterdam, New York, Oxford, Tokyo: Elsevier, 1990. P. 871–941.
44. Pippenger N. On Simultaneous Resource Bounds // 20th Annual Symposium on Foundations of Computer Science (SFCS 1979). San Juan, Puerto Rico: IEEE, 1979. P. 307–311. DOI: 10.1109/SFCS.1979.29.
45. Pippenger N. Pebbling with an Auxiliary Pushdown // Journal of Computer and System Sciences. Academic Press, 1981. Vol. 23, No. 2. P. 151–165. DOI: 10.1016/0022-0000(81)90011-8.
46. Snyder L. Type Architectures, Shared Memory, and the Corollary of Modest Potential // Annual Review of Computer Science. 1986. Vol. 1, No. 1. P. 289–317. DOI: 10.1146/annurev.cs.01.060186.001445.
47. Mehlhorn K., Vishkin U. Randomized and Deterministic Simulations of PRAMs by Parallel Machines with Restricted Granularity of Parallel Memories // Acta Informatica. Springer-Verlag, 1984. Vol. 21, No. 4. P. 339–374. DOI: 10.1007/BF00264615.
48. Gibbons P.B., Matias Y., Ramachandran V. The Queue-Read Queue-Write PRAM Model: Accounting for Contention in Parallel Algorithms // SIAM Journal on Computing. 1998. Vol. 28, No. 2. P. 733–769. DOI: 10.1137/S009753979427491.
49. Gibbons P.B., Matias Y. Can a Shared-Memory Model Serve as a Bridging Model for Parallel Computation? // Theory of Computing Systems. 1999. Vol. 32, No. 3. P. 327-359. DOI: 10.1007/s002240000121.
50. Aggarwal A., Chandra A.K., Snir M. On Communication Latency in PRAM Computations // Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures — SPAA'89. New York, New York, USA: ACM Press, 1989. P. 11–21. DOI: 10.1145/72935.72937.
51. Mansour Y., Nisan N., Vishkin U. Trade-offs between Communication Throughput and Parallel Time // Journal of Complexity. Academic Press, 1999. Vol. 15, No. 1. P. 148-166. DOI: 10.1006/JCOM.1998.0498.
52. Cole R., Zajicek O. The APRAM: Incorporating Asynchrony into the PRAM Model // Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures — SPAA'89. New York, New York, USA: ACM Press, 1989. P. 169–178. DOI: 10.1145/72935.72954.
53. Gibbons P.B. A More Practical PRAM Model // Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures — SPAA'89. New York, New York, USA: ACM Press, 1989. P. 158–168. DOI: 10.1145/72935.72953.
54. VALIANT L.G. General Purpose Parallel Architectures // Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity. Elsevier, 1990. P. 943–971. DOI: 10.1016/B978-0-444-88071-0.50023-0.
55. de la Torre P., Kruskal C.P. Towards a Single Model of Efficient Computation in Real Parallel Machines // Future Generation Computer Systems. North-Holland, 1992. Vol. 8, No. 4. P. 395–408. DOI: 10.1016/0167-739X(92)90071-I.
56. Heywood T., Ranka S. A Practical Hierarchical Model of Parallel Computation I. The model // Journal of Parallel and Distributed Computing. Academic Press, 1992. Vol. 16, No. 3. P. 212–232. DOI: 10.1016/0743-7315(92)90034-K.

57. Forsell M. A PRAM-NUMA Model of Computation for Addressing Low-TLP Workloads // International Journal of Networking and Computing. [Hiroshima University], 2011. Vol. 1, No. 1. P. 21–35.
58. Ranade A.G. How to Emulate Shared Memory // Journal of Computer and System Sciences. Academic Press, 1991. Vol. 42, No. 3. P. 307–326. DOI: 10.1016/0022-0000(91)90005-P.
59. Forsell M. et al. Hardware and Software Support for NUMA Computing on Configurable Emulated Shared Memory Architectures // 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum. IEEE, 2013. P. 640–648. DOI: 10.1109/IPDPSW.2013.146.
60. Forsell M. E — A Language for Thread-Level Parallel Programming on Synchronous Shared Memory NOCs // WSEAS Transactions on Computers. 2004. Vol. 3, No. 3. P. 807–812.
61. Forsell M., Leppanen V. An Extended PRAM-NUMA Model of Computation for TCF Programming // International Journal of Networking and Computing. 2013. Vol. 3, No. 1. P. 98–115.
62. Aggarwal A. et al. A Model for Hierarchical Memory // Proceedings of the Nineteenth annual ACM Conference on Theory of Computing — STOC'87. New York, New York, USA: ACM Press, 1987. P. 305–314. DOI: 10.1145/28395.28428.
63. Aggarwal A., Chandra A.K., Snir M. Hierarchical Memory with Block Transfer // 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). IEEE, 1987. P. 204–216. DOI: 10.1109/SFCS.1987.31.
64. Luccio F., Pagli L. A Model of Sequential Computation with Pipelined Access to Memory // Mathematical Systems Theory. Springer-Verlag, 1993. Vol. 26, No. 4. P. 343–356. DOI: 10.1007/BF01189854.
65. Mead C.A., Conway L.A. Introduction to VLSI systems. Boston, MA, USA: Addison-Wesley, 1980. 396 p.
66. Alpern B. et al. The Uniform Memory Hierarchy Model of Computation // Algorithmica. Springer-Verlag, 1994. Vol. 12, No. 2–3. P. 72–109. DOI: 10.1007/BF01185206.
67. Vitter J.S., Shriver E.A.M. Algorithms for parallel memory, II: Hierarchical multilevel memories // Algorithmica. Springer-Verlag, 1994. Vol. 12, No. 2–3. P. 148–169. DOI: 10.1007/BF01185208.
68. Tiskin A. BSP (Bulk Synchronous Parallelism) // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 192–199. DOI: 10.1007/978-0-387-09766-4_311.
69. Goudreau M. et al. Towards Efficiency and Portability: Programming with the BSP Model // Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures — SPAA'96. New York, NY, USA: ACM Press, 1996. P. 1–12. DOI: 10.1145/237502.237503.
70. Bisseling R.H. Parallel Scientific Computation: A Structured Approach using BSP and MPI. New York: Oxford University Press, 2004. 325 P.
71. McColl W.F. Scalable Computing // J. van Leeuwen (eds). Computer Science Today: Recent Trends and Developments. Lecture Notes in Computer Science, Vol. 1000. Berlin, Heidelberg: Springer, 1995. P. 46–61. DOI: 10.1007/BFb0015236.
72. Tiskin A. The Bulk-synchronous Parallel Random Access Machine // Theoretical Computer Science. 1998. Vol. 196, No. 1–2. P. 109–130. DOI: 10.1016/S0304-3975(97)00197-7.
73. McColl W.F., Tiskin A. Memory-Efficient Matrix Multiplication in the BSP Model // Algorithmica. Springer-Verlag, 1999. Vol. 24, No. 3–4. P. 287–297. DOI: 10.1007/PL00008264.

74. Kielmann T., Gorlatch S. Bandwidth-Latency Models (BSP, LogP) // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 107–112. DOI: 10.1007/978-0-387-09766-4_189.
75. Alexandrov A. et al. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation // Journal of Parallel and Distributed Computing. 1997. Vol. 44, No. 1. P. 71–79. DOI: 10.1006/jpdc.1997.1346.
76. Kielmann T., Bal H.E., Verstoep K. Fast Measurement of LogP Parameters for Message Passing Platforms // Parallel and Distributed Processing. IPDPS 2000. Lecture Notes in Computer Science, Vol. 1800. Berlin, Heidelberg: Springer, 2000. P. 1176–1183. DOI: 10.1007/3-540-45591-4_162.
77. Gropp W., Lusk E., Skjellum A. Using MPI: Portable Parallel Programming with the Message-Passing Interface. Second Ed. MIT Press, 1999.
78. Gropp W. MPI 3 and Beyond: Why MPI Is Successful and What Challenges It Faces // Recent Advances in the Message Passing Interface. EuroMPI 2012. Lecture Notes in Computer Science, Vol. 7490 / ed. Träff J.L., Benkner S., Dongarra J.J. Berlin, Heidelberg: Springer, 2012. P. 1–9. DOI: 10.1007/978-3-642-33518-1_1.
79. Touyama T., Horiguchi S. Parallel Computation Model LogPQ // High Performance Computing. ISHPC 1997. Lecture Notes in Computer Science, vol 1336 / ed. Polychronopoulos C., Joe K., Araki K. A.M. Berlin, Heidelberg: Springer, 1997. P. 327–334. DOI: 10.1007/BFb0024227.
80. Touyama T., Horiguchi S. Performance Evaluation of Practical Parallel Computation Model LogPQ // Proceedings of the Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99). Washington, DC, USA: IEEE Computer Society, 1999. P. 216–221. DOI: 10.1109/ISPAN.1999.778942.
81. Palmer J., Steele G.L. Connection Machine model CM-5 System Overview // Frontiers'92, the Fourth Symposium on the Frontiers of Massive Parallel Computation, October 19-21, 1992, McLean, Virginia. IEEE Computer Society Press, 1992. P. 474–483. DOI: 10.1109/FMPC.1992.234877.
82. Ino F., Fujimoto N., Hagihara K. LogGPS: A Parallel Computational Model for Synchronization Analysis // ACM SIGPLAN Notices. 2001. Vol. 36, No. 7. P. 133–142. DOI: 10.1145/568014.379592.
83. Gropp W. et al. A High-performance, Portable Implementation of the MPI Message Passing Interface Standard // Parallel Computing. 1996. Vol. 22, No. 6. P. 789–828. DOI: 10.1016/0167-8191(96)00024-5.
84. Moritz C.A. et al. LoGPC: Modeling Network Contention in Message-Passing Programs // ACM SIGMETRICS Performance Evaluation Review. New York, New York, USA: ACM Press, 1998. Vol. 26, No. 1. P. 254–263. DOI: 10.1145/277851.277933.
85. Moritz C.A., Frank M.I. LoGPC: Modeling Network Contention in Message-Passing Programs // IEEE Transactions on Parallel and Distributed Systems. 2001. Vol. 12, No. 4. P. 404–415. DOI: 10.1109/71.920589.
86. Agarwal A. et al. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor // Scalable Shared Memory Multiprocessors. Proceedings of a workshop held May 26-27, 1990, in Seattle, Wash. / ed. Dubois M., Thakkar S. Boston, MA: Springer, 1992. P. 239–261. DOI: 10.1007/978-1-4615-3604-8_13.
87. Kubiatowicz J., Agarwal A. Anatomy of a Message in the Alewife multiprocessor // ACM International Conference on Supercomputing 25th Anniversary Volume. New York, NY, USA: ACM Press, 2014. P. 193–204. DOI: 10.1145/2591635.2667168.

88. Cameron K.W., Ge R., Sun X.-H. lognP and log3P: Accurate Analytical Models of Point-to-point Communication in Distributed Systems // IEEE Transactions on Computers. 2007. Vol. 56, No. 3. P. 314–327. DOI: 10.1109/TC.2007.38.
89. Cameron K.W., Ge R. Predicting and Evaluating Distributed Communication Performance // Proceedings of the 2004 ACM/IEEE Conference on Supercomputing. IEEE, 2004. P. 15. DOI: 10.1109/SC.2004.40.
90. Cameron K.W., Sun X.-H. Quantifying Locality Effect in Data Access Delay: Memory logP // Proceedings of the 2003 IEEE International Parallel and Distributed Processing Symposium (IPDPS'03). IEEE Comput. Soc, 2003. P. 8. DOI: 10.1109/IPDPS.2003.1213137.
91. Cappello F. et al. HiHCoHP-Toward a Realistic Communication Model for Hierarchical Hyperclusters of Heterogeneous Processors // Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001. IEEE Comput. Soc., 2001. P. 6. DOI: 10.1109/IPDPS.2001.924978.
92. Cappello F. et al. An Algorithmic Model for Heterogeneous Hyper-Clusters: Rationale and Experience // International Journal of Foundations of Computer Science. World Scientific Publishing Company, 2005. Vol. 16, No. 02. P. 195–215. DOI: 10.1142/S0129054105002942.
93. Bosque J.L., Pastor L. A Parallel Computational Model for Heterogeneous Clusters // IEEE Transactions on Parallel and Distributed Systems. 2006. Vol. 17, No. 12. P. 1390-1400. DOI: 10.1109/TPDS.2006.165.
94. Hoefler T. et al. LogfP — a Model for Small Messages in InfiniBand // Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. Washington, DC, USA: IEEE Computer Society, 2006. P. 319–319. DOI: 10.1109/IPDPS.2006.1639624.
95. Jepsen T.C. InfiniBand // Distributed Storage Networks: Architecture, Protocols and Management. Chichester, West Sussex, England: John Wiley & Sons, 2013. P. 159–174. DOI: 10.1002/9780470871461.ch6.
96. Nasri W., Tarhouni O., Slimi N. PLP: Towards a Realistic and Accurate Model for Communication Performances on Hierarchical Cluster-based Systems // 2008 IEEE International Symposium on Parallel and Distributed Processing. IEEE, 2008. P. 1–8. DOI: 10.1109/IPDPS.2008.4536486.
97. Hoefler T., Schneider T., Lumsdaine A. LogGOPSim – Simulating Large-Scale Applications in the LogGOPS Model // Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing — HPDC'10. New York, New York, USA: ACM Press, 2010. P. 597–604. DOI: 10.1145/1851476.1851564.
98. Valiant L.G. A Bridging Model for Multi-core Computing // Journal of Computer and System Sciences. Elsevier Inc., 2011. Vol. 77, No. 1. P. 154–166. DOI: 10.1016/j.jcss.2010.06.012.
99. Tu B. et al. Performance Analysis and Optimization of MPI Collective Operations on Multicore Clusters // The Journal of Supercomputing. Springer US, 2012. Vol. 60, No. 1. P. 141–162. DOI: 10.1007/s11227-009-0296-3.
100. Tu B. et al. Accurate Analytical Models for Message Passing on Multi-core Clusters // 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing. IEEE, 2009. P. 133–139. DOI: 10.1109/PDP.2009.18.
101. Sterling T. et al. SLOWER: A Performance Model for Exascale Computing // Supercomputing Frontiers and Innovations. 2014. Vol. 1, No. 2. P. 42–57. DOI: 10.14529/jsfi140203.

102. Gerbessiotis A. V. Extending the BSP Model for Multi-core and Out-of-core Computing: MBSP // Parallel Computing. Elsevier B.V., 2015. Vol. 41. P. 90–102. DOI: 10.1016/j.parco.2014.12.002.
103. Amaris M. et al. A Simple BSP-based Model to Predict Execution Time in GPU Applications // 2015 IEEE 22nd International Conference on High Performance Computing (HiPC). IEEE, 2015. P. 285–294. DOI: 10.1109/HiPC.2015.34.
104. Maggs B.M., Matheson L.R., Tarjan R.E. Models of Parallel Computation: a Survey and Synthesis // Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences. IEEE Comput. Soc. Press, 1995. P. 61–70. DOI: 10.1109/HICSS.1995.375476.
105. Rico-Gallego J.A. et al. A Survey of Communication Performance Models for High-Performance Computing // ACM Computing Surveys. ACM, 2019. Vol. 51, No. 6. P. 1–36. DOI: 10.1145/3284358.

Ежова Надежда Александровна, аспирант, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

Соколинский Леонид Борисович, д.ф.-м.н., профессор, проректор по информатизации, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

---

**DOI: 10.14529/cmse190304**

# SURVEY OF PARALLEL COMPUTATION MODELS

**© 2019 N.A. Ezhova, L.B. Sokolinsky**

*South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)*

*E-mail: EzhovaNA@susu.ru, Leonid.Sokolinsky@susu.ru*

Received: 01.06.2019

This survey aims to present the state of the art in analytic parallel computation models, providing sufficiently detailed descriptions of particularly noteworthy efforts. Such models allow predicting the computation time, speedup, efficiency and scalability of parallel algorithms for various target multiprocessor platforms. Modeling the cost of computations and communications in multiprocessor systems is an important and challenging problem. It provides insights into the design of the parallel algorithms for optimization of their deployment in the increasingly complex high-performance computing. The survey shows the evolution of parallel computing models inspired by the evolution of multiprocessor systems, from single-level models with shared memory to multi-level hierarchical models with distributed memory, which correspond to multicore clusters. The review concludes with prospective directions for further research in the area of developing mathematical models for parallel computing.

*Keywords: parallel computing model, survey, parallel programming, multiprocessor systems, performance evaluation, execution time prediction.*

**FOR CITATION**

Ezhova N.A., Sokolinsky L.B. Survey of Parallel Computation Models. *Bulletin of the South Ural State* University. Series: Computational Mathematics and Software Engineering. 2019. vol. 8, no. 3. pp. 58–91. (in Russian) DOI: 10.14529/cmse190304.