УДК 004.272.43+004.23+004.25

**Высокоуровневая система параллельного программирования многоядерного гибридного процессора**

Биконов Дмитрий Владиленович,

Бражкин Алексеевич Алексеевич,

Пузиков Артем Дмитриевич,

Сивцов Алексей Сергеевич,

Дементьев Владимир Викторович,

Спесивцев Дмитрий Александрович,

Эйсымонт Леонид Константинович

*ЗАО НТЦ "Модуль".125190, а/я 166. г. Москва, 4-ая улица 8 Марта, д.3. d.bikonov@module.ru*

**Аннотация**

Объясняется выбор набора функций Библиотеки управления процессами и их взаимодействиями (БУПВ/М), поддерживающего модели параллельных программ типа MPI и PGAS, предназначенных для реализации на процессоре-ускорителе NeuroMatrix NM6408MP (1879ВМ8Я). Описываются функции этой библиотеки их реализация.

**Ключевые слова:**

Массово-мультитредовые процессоры; модели параллельных вычислений; средства параллельного программирования; MPI; SHMEM; PGAS; графовые модели вычислений; управление потоком данных; внутрикристальные сети; сигналы; события.

# **Введение**

Разработка процессоров-ускорителей в последние годы претендует на главное направление прорыва в области суперкомпьютеров [1, 2, 3, 4, 5]. Построенные на их основе вычислительные узлы, обеспечивают не только приближение к преодолению экзафлопсного барьера (1018 операций над 64-х разрядными числами (FP64)) на научно-технических вычислениях, что еще совсем недавно представлялось сложнейшей задачей [6], но для области нейровычислений уже сейчас позволяют строить специализированные суперкомпьютеры с производительностью до 16 экзафлопс (числами FP16/FP32) [3,4]. Прогресс в разработке процессоров-ускорителей, обеспечен не только достижениями в технологиях, все заметнее стала использоваться специфика прикладных алгоритмов (в данном случае нейросетевых). Выдающиеся примеры последних месяцев - GPU Ampere [2] и Colossus Mk2 GC200 [4].

Заметна и другая тенденция, для комфортной работы с такими процессорами в их программном обеспечении стремятся скрыть присущую им специфику, но при этом не терять обеспечиваемую ею эффективность. В свете этой проблемы предлагаемая статья описывает выбор и реализацию средств параллельного программирования в виде библиотеки БУПВ/М для специализированного гибридного 21-ядерного процессора-ускорителя NeuroMatrix NM6408MP (1879ВМ8Я) [7] (далее-NM6408MP).

# **Концепция функциональности БУПВ/М и особенности NM6408MP**

Идея выбора функций БУПВ/М состояла в том, что это должен быть небольшой набор функций, соответствующих общепринятым моделям параллельных вычислений и предельно освобождающих пользователя от знания специфики процессора. Споры по этому поводу были такими, что до завершения реализации БУПВ/М по просьбе коллег были реализованы специальные библиотеки низкого уровня – БФСКН и БФСКН-Ф [8]

Рассмотрим особенности NM6408MP и связанную с ними логику выбора функций БУПВ/М. Будем использовать представление структуры процессора (см. Рисунок 1) и схему соединения его узлов (см. Рисунок 2) [8]

Структура NM6408MP иерархическая, имеются четыре кластера, в каждом из которых есть управляющий узел ARM-node и четыре узла NM-node, имеется также центральный узел ARM-node. В кластерах все NM-node связаны по принципу «каждый с каждым» и все с узлом ARM-node этого кластера. ARM-node кластеров также связаны друг с другом по принципу «каждый с каждым» и все с ARM-node центрального узла.

Изначально надо было решить, отражать ли в типах процессов БУПВ/М и их правах взаимодействия иерархическую структуру процессора NM6408MP?

# **Выбор моделей параллельных вычислений.**

Можно было выбрать, что ведущими являются ARM-node, а процессы на NM-node кластера считать подчиненными. Взаимодействие NM-node кластера между собой и ARM-node разрешить только внутри кластера. С центральным узлом взаимодействуют только ARM-node кластеров. При этом считать, что ARM-node центрального узла раздает задания ARM-node кластерных узлов, а те раздают задания на обработку NM-node своего кластера. Такая модель называется «master-slave». На такую модель ориентировались при разработке NM6408MP. Естественно, что такой точки зрения на функционал БУПВ/М придерживались разработчики этого процессора.

****

Рисунок 1. Структура микропроцессора NM6408MP

Разработчики БУПВ/М иначе взглянули на NM6408MP. Во-первых, было жаль терять возможности взаимодействия NM-node через высокоскоростную сеть, например, путь из NMPU03 в NMPU33 (Рисунок 2). Во-вторых, не хотелось ограничивать пользователей только моделью параллельной программы типа «master-slave».



Рисунок 2. Схема соединения узлов NM6408 через коммуникационные ресурсы внутрикристальной высокоскоростной сети коммутации каналов.

Имея опыт работы с суперкомпьютерами, построенными на многосокетных вычислительных узлах [10, 11, 12], разработчики БУПВ/М восприняли структуру NM6408MP иначе, а именно так. NM6408MP представляется как фрагмент суперкомпьютера, содержащий четыре 5-ти сокетные платы (это кластеры) с высокоскоростными внутриплатными сетями и межузловой сетью (это высокоскоростная сеть), дополнительно снабженный специальным средством, обеспечивающим работу с глобально адресуемой памятью (это среднескоростная AXI-сеть). Центральный узел NM6408MP представляется при этом как host-процессор такого фрагмента суперкомпьютера.

В соответствии с таким видением NM6408MP, в БУПВ/М была принята «плоская» MPI-модель процессов (не физических узлов, но с моделью размещения одного процесса на узле), в ней все процессы одинаковы и имеют логические номера в виде положительных целых. Размещение процессов на физических узлах происходит при начальной загрузке параллельной программы с host-процессора в соответствии с информацией из файла привязки логических номеров процессов к физическим узлам и не меняется при выполнении параллельной программы. В дополнение к этой модели процессов была принята обеспечиваемая аппаратными средствами возможность их работы над общей памятью, это уже PGAS-модель разделенной глобальной памяти.

Далее рассмотрим, к это отразилось в функциях БУПВ/М и какие появились проблемы.

# **Функционирование процессов и их взаимодействие**

Для MPI-модели типичным средством взаимодействия процессов являются функции передачи сообщений типа Send/Receive. Имеются три особенности их реализации в БУПВ/М.

Во-первых, в процессе-поставщике сообщение передается компортом прямого доступа к памяти, процессорное ядро в этом не участвует. В процессе-получателе сообщение принимается через его компорт прямого доступа к памяти, процессорное ядро не участвует в приеме. Каждый компорт дуплексный, их четыре в каждом узле, поэтому один процесс одновременно с вычислениями может производить 4 выдачи сообщений и 4 приема.

Во-вторых, компорты передают и принимают сообщения порциями по двойному слову (по 64 разряда). При этом память адресуется либо непрерывно, с увеличением адреса очередного двойного слова на единицу (одномерно), либо с разрывами, когда сообщение пересылается фрагментами заданной длины с одномерной адресацией, но с заданными промежутками между ними (двумерная адресация, Рисунок 3).

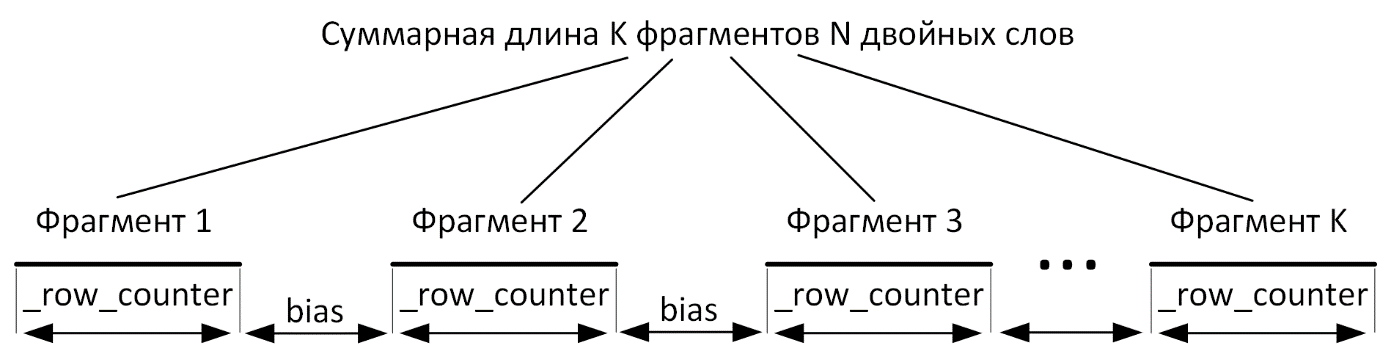


Рисунок 3. Двумерная адресация сообщений в памяти

В-третьих, компорты надо программно настраивать для передачи сообщений. Именно от этого БУПВ/М освобождает пользователя в первую очередь.

Синхронизация процессов в MPI-модели производится функциями барьерной синхронизации. Такие функции в БУПВ/М отражают особенности NM6408MP. Имеется возможность барьерной синхронизации только внутри одного кластера, а также во всем процессоре. Реализация таких функций полностью основана на аппаратных средствах. Дополнительно введены реализуемые программно специфические функции синхронизации, обеспеечивающие установку контрольных точек программы.

Для PGAS моделей возможность работы с разделяемой глобально адресуемой памятью поддерживается функциями типа get/put, а также функциями lock/unlock (mutex-переменные).

Кроме MPI и PGAS-моделей поддерживаются элементы новых графовых моделей с управлением потоком данных, для этого введены функции работы с сигналами и событиями.

# **Особенности работы с внутрикристальными памятями**

Реализация модели PGAS требует знания особенностей организации внутрикристальной памяти.

Внутрикристальная память NM6408MP распределена по вычислительным узлам. В центральном узле имеется локальная память объемом 512 KB c 4-х кратным расслоением. Каждый ARM-node кластеров включает локальную память объемом 256 KB с 4-х кратным расслоением, а каждый NM-node имеет локальную память объемом 512 KB, но с 8-кратным расслоением. Кроме процессорного ядра к локальной памяти в узлах подключены компорты прямого доступа и порт для внутрикристальной среднескоростной AXI-сети.

Выполняемые на узлах процессы могут работать со своими локальными памятями, а через порты AXI-шины им доступны локальные памяти других узлов и даже любые подключенные к процессору DDR-памяти. Задержки выполнения обращений к собственной локальной памяти и другим памятям значительно отличаются. Полная картина характеристик доступа к памятям дана в работе [8]. По сравнению с временем обращения к собственной локальной памяти, времена выполнения обращений к памятям удаленных узлов могут возрастать в десятки раз даже если нет конфликтов в сети. Таким образом, необходимы какие-то средства оптимизации времени доступа к удаленным памятям, рассмотрим их.

Процессорные ядра узлов NM6408MP имеют разные возможности использования кэш-памятей. В центральном узле имеется L2-кэш объемом 256 KB. Все узлы ARM-node имеют L1 кэш-памяти команд и данных объемом по 32 KB. Узлы NM-node имеют только кэш-памяти команд объемом 8 KB. Отказ в большей части узлов процессора от использования кэш-памятей типичен для современных многоядерных процессоров и считается правильным. Вместо кэш-памятей применяется возможность асинхронной подкачки команд и данных узлами в свои локальные памяти. Для этого в каждом кластере и центральном узле имеется блок MDMAC.

В функциях БУПВ/М эти особенности асинхронной подкачки учтены - введены функции обращения к блокам MDMAC, но их настройка со стороны пользователя не требуется. Более того, функции позволяют одному процессу использовать до 5 блоков MDMAC, если они свободны, это решается в динамике, без пользователя.

Модель PGAS предполагает наличие на уровне аппаратных средств по крайней мере атомарных операций с любым словом памяти, но в NM6408MP это реализовано не полностью, суть в следующем. Атомарные операции (в процессоре они называются операциями «эксклюзивного доступа») разрешены лишь для 0-го блока локальной памяти узлов ARM-node. Имеется еще одно жесткое ограничение — эксклюзивный доступ возможен только через имеющиеся в каждом таком 0-м блоке шесть специальных устройств, называемых мониторами. Таким образом, одновременный эксклюзивный доступ к словам какого-либо 0-го блока возможен не более, чем от шести процессов. Такое ограничение логично, если бы в параллельных программах использовалась лишь модель «master-slave».

Для выбранных в БУПВ/М более общих моделей вычислений такое ограничение надо cнять. Заметим, что такой проблемы не было бы если в каждом 0-м блоке имелся 21 монитор. В БУПВ/М эта проблема решена [8], пользователь об этом беспокоиться не должен.

# **Работа с высокоскоростной внутрикристальной сетью**

В обычной практике, в межкристальных и внутрикристальных сетях с любой топологией информация о маршруте пересылки сообщения помещается в его заголовок процессом-поставщиком. Этот заголовок используется аппаратными средствами сети в каждой очередной точке маршрута, так происходит продвижение сообщения по сети, пока оно не достигнет процесса-получателя.

Особенность NM6408MP в том, что такой маршрут в высокоскоростной сети надо перед передачей сообщения настраивать напрямую. Это затрудняет работу пользователя, но позволяет упростить реализацию, обеспечивает более высокие характеристик по пропускной способности и задержкам. В высокоскоростной сети (Рисунок 2) возможны следующие маршруты: *тип* 1 — «компорт-компорт» внутри кластера; *тип* 2 — «компорт-коммутатор-компорт» также внутри кластера; *тип* 3 — «компорт-коммутатор-компорт» между узлом кластера и центральным узлом; *тип* 4 — «компорт-коммутатор-коммутатор-компорт» между узлами разных кластеров; *тип* 5 — «X-маршрут» между узлами, для которых нет прямого соединения через высокоскоростную сеть.

Информация о всех таких маршрутах в БУПВ/М хранится в постоянной таблице TRS размером 21х21 элемент. В элементах TRS для каждой пары внутренних номеров процесса-поставщика и процесса-получателя маршрут зашифрован. Внутренние номера процессов определяются по их логическим номерам через таблицу TFX, которая строится при загрузке параллельной программы. X-маршрутов в TRS много (65.7%), но при разработке БУПВ/М это не стало поводом для отказа от принятой простой и однородной MPI-модели процессов.

Если маршрут в TRS не является Х-маршрутом, то его компорты и коммутаторы (сетевые ресурсы) захватываются и настраиваются перед передачей сообщения. Для этого используется возможность доступа процессора к их управляющим регистрам через общее адресное пространство процессора. Однако есть трудность – из процесса-поставщика нельзя настроить компорт процесса-получателя и наоборот. Такая проблема была в БУПВ/М решена за счет реализации контролируемого встречного автоматического построения маршрута.

Если маршрут в TRS является X-маршрутом, то в текущей версии БУПВ/М применяется простое решение - используется свободный в данный момент блок MDMAC. Были и другие варианты. Наиболее приоритетный – с применением векторных операций узлов NM-node и специальных скрытых транзитных процессов в узлах ARM-node. Реализация этого варианта пока отложена, но в будущем не исключается.

Построение маршрутов скрыто от пользователя, выполняется с использованием полной информации о состоянии сетевых ресурсов. Для этого сетевым ресурсам поставлены в соответствие специальные информационные структуры, называемые дряйверами ресурсов. В каждом таком драйвере отражается занятость ресурса и его текущее состояние. Контроль и управление ресурсами через их драйвера происходит по достаточно сложным алгоритмам с применением множества специальных системных mutex-переменных.

Далее в разделе 3.1 тема передачи сообщений по высокоскоростной сети будет продолжена в связи с рассмотрением подгруппы функций типа Send/Recеive. Реализация функций этой подгруппы оказалась самой сложной.

# **Обеспечение управляемости вычислений и технические особенности реализации БУПВ/М**

Как сообщалось, для обеспечения контроля и управления сетевыми ресурсами были введены их драйвера. Однако есть еще одни объекты, которые могут находиться в разных сложных состояниях. Это сами процессы параллельной программы. Для отображения их состояния каждому процессу также поставлен в соответствие драйвер, но его принято называть фреймом. Это типичное решение для операционных и run-time систем поддержи выполнения программ с множеством выполняемых параллельных или квазипараллельных процессов.

Фрейм процесса имеет объем 105 слов, содержит: заголовок процесса (2 слова); поле принимаемых сигналов и их маску (1 cлово); адреса первого и последнего фрейма обращений к MP\_send (1 слово); адреса первого и последнего фрейма обращений к MP\_recv (1 слово); ссылка на следующий процесс в списке ожидающих переменной типа событие или mutex-переменной (1 слово); кванты времени на выполнение процесса пользователя и системные процессы (2 слова, только для узлов ARM-node); четыре области для фреймов обращений к MP\_send (4х12 слов); четыре области для фреймов обращений к MP\_recv (4х12 слов); счетчик контрольных точек (1 слово, только для центрального узла).

Всего во фрейме процесса, включая фреймы MP\_send и MP\_recv, имеется 190 информационных подполей размером от нескольких бит до слова. Фрейм процесса хранится в локальной памяти узла, на котором процесс выполняется, доступ к фрейму защищен системными mutex-переменными, это критический ресурс.

Приведем еще общие сведения по сложности драйверов ресурсов: драйвер компорта — 6 слов и 24 поля в нем; драйвер 4-х портового коммутатора — 4 слова и 36 полей; драйвер 3-х портового коммутатора — 4 слова и 24 поля; драйвер MDMAC — 13 слов и 13 полей; драйвер событийной переменной — 2 слова и 4 поля; драйвер muteх-переменной пользователя — 3 слова и 5 подполей. Драйвера ресурсов размещаются в локальных памятях узлов.

Системные mutex-переменные размещаются большей частью в 0-х блоках локальной памяти ARM-node кластеров и в 0-м блоке локальной памяти центрального узла.

При работе с драйверами и фреймами важно применение барьерных синхронизаций для операций с памятью, в ARM-node это команда DMB, а в NM-node — это специальное сочетание команд, после выполнения которых гарантируется, что ранее выданные обращения к памяти завершены.

При выполнении параллельных программ на таком сложном по структуре процессоре могут складываться конфликтные ситуации по использованию ресурсов, которые разрешимы со стороны пользователя. Для этого применяется аппарат выдачи кодов возврата расширенные средства отладки и трассировки.

# **Функции библиотеки БУПВ/М**

Библиотека БУПВ/М содержит более 50 функций, 8 функций из них для выполнения на host-процессоре, остальные – для NM6408MP. Одноименные функции для NM6408MP реализуются в трех вариантах — для узлов NM-node, ARM-node кластеров (CPU) и центрального узла (ССPU). Рассмотрим далее эти функции.

*Функции host-процессора*:MP\_load — загрузка параллельной программы в NM6408MP; MP\_run — запуск параллельной программы; MP\_stop — останов параллельной программы; MP\_wait\_checkout — ожидание контрольной точки в ССPU-процессе параллельной программы; MP\_barrier\_and\_swap\_scalar **—** барьерная синхронизация с CCPU-процессом параллельной программы и обмен скалярами; MP\_barrier\_and\_swap\_array **—** барьерная c ССPU-процессом параллельной программы и обмен информацией о массивах для обмена; MP\_run\_trace\_in — установка режима приема фрагментов трасс с буфера соответствующего процесса параллельной программы; MP\_stop\_trace\_in — снятие режима приема фрагментов трасс с буфера соответствующего процесса параллельной программы.

*Функции процессора NM6408MP* разбиты на следующие подгруппы*:* инициализации и завершения выполнения параллельной программы (MP\_init, MP\_finalize); пересылки сообщений типа Send/Recv (14 функций); пересылки областей памяти (4 функций); работы с сигналами (5 функций); работы с событиями (4 функции); работа с критическими ресурсами (MP\_lock, MP\_unlock); пересылки в общем адресном пространстве (MP\_put, MP\_get, MP\_global\_address); барьерной синхронизации (4 функции); самоидентификации и времени (MP\_my\_id, MP\_node\_gettime, MP\_gettime); трассировки и отладки (10 функций).

# **Подгруппа функций пересылки сообщений типа Send/Recv.**

Это наиболее сложная по реализации подгруппа, ее функции обеспечивают пересылку по динамическим и статическим маршрутам [8], далее приведем более полные сведения и новые результаты оценочного тестирования.

*Динамические маршруты*. Имеются четыре функции для пересылки по динамическому маршруту (MP\_send, MP\_recv, MP\_2D\_send, MP\_2D\_recv). Здесь и далее в именах функций «2D» означает двумерную адресацию, а «1D» – одномерную. Указание об одномерной адресации иногда опускается, как в MP\_send и MP\_recv.

Название «динамический» маршруту выбрано из-за того, что в данном случае процесс-поставщик и процесс-получатель, обращаясь соответственно к MP\_sendиMP\_recv сначала строят маршрут передачи сообщения. Если маршрут построен, то далее передается сообщение, а после этого его ресурсы освобождаются. Достоинство динамического маршрута - пользователь освобожден от его прокладки и освобождения. Недостаток такого маршрута - к затратам на пересылку добавляются накладные расходы на захват и потом освобождение маршрута.

*Статический маршрут*. Имеются 10 функций для работы со статическими маршрутами. Эти маршруты сначала строятся (MP\_send\_path\_on, MP\_recv\_path\_on, MP\_2D\_send\_path\_on,MP\_2D\_recv\_path\_on, MP\_path\_check), потом по ним много раз могут пересылаться сообщения, причем разной длины (MP\_send\_path, MP\_recv\_path), когда маршрут оказывается ненужным он удаляется (MP\_drop\_path).

Достоинство статических маршрутов очевидно. Если требуется много пересылок для пары процессов, то для каждого сообщения нет накладных расходов на захват, настройку и освобождение маршрута. Недостаток таких маршрутов – занятость его ресурсов, что может помешать пересылкам между другими процессами.

Рассмотрим оценки пропускных способностей по динамическим и статическим маршрутам для двух вариантов: короткий маршрут «компорт-компорт» между узлами NMPU00 и NMPU01(Рисунок 4); длинный маршрут «компорт-коммутатор-коммутатор-компорт» между узлами NMPU03 и NMPU33 (Рисунок 5). Видно, что пересылка по статическому маршруту позволяет развить большую пропускную способность. Для маршрута из NMPU03 в NMPU33 выигрыш наиболее заметен. Дополнительно отметим два факта.

Во-первых, текущая версия БУПВ/М странслирована компилятором Си с уровнем оптимизации OPT=3, но ручная доводка полученного кода не проводилась. Если транслировать с уровнем OPT=0, то результаты для БУПВ/М оказываются хуже не менее, чем на 1.0 GB/s.

Во-вторых, результаты для функций БУПВ/М сравниваются с наилучшими практически достижимыми на низкоуровневой библиотеке БФСКН-Ф, ее функции странслированы компилятором Си в режиме высшего уровня оптимизации OPT=3. Для этой библиотеки приведены два результата — когда ожидание работы компортов производится опросом его регистра с битом признака окончания работы (полинг) и когда задействован более сложный механизм обработки прерываний.

Обратим еще внимание, что даже для БФСКН-Ф максимальная пропускная способность для маршрута из NMPU03 в NMPU33 оказывается ниже, чем для маршрута из NMPU00 в NMPU01. Это объясняется тем, что в длинном маршруте имеется больше работающих ресурсов, образующих ступени конвейера пересылки. Работа такого конвейера оказывается не всегда идеальна, происходит иногда рассинхронизация ступеней.

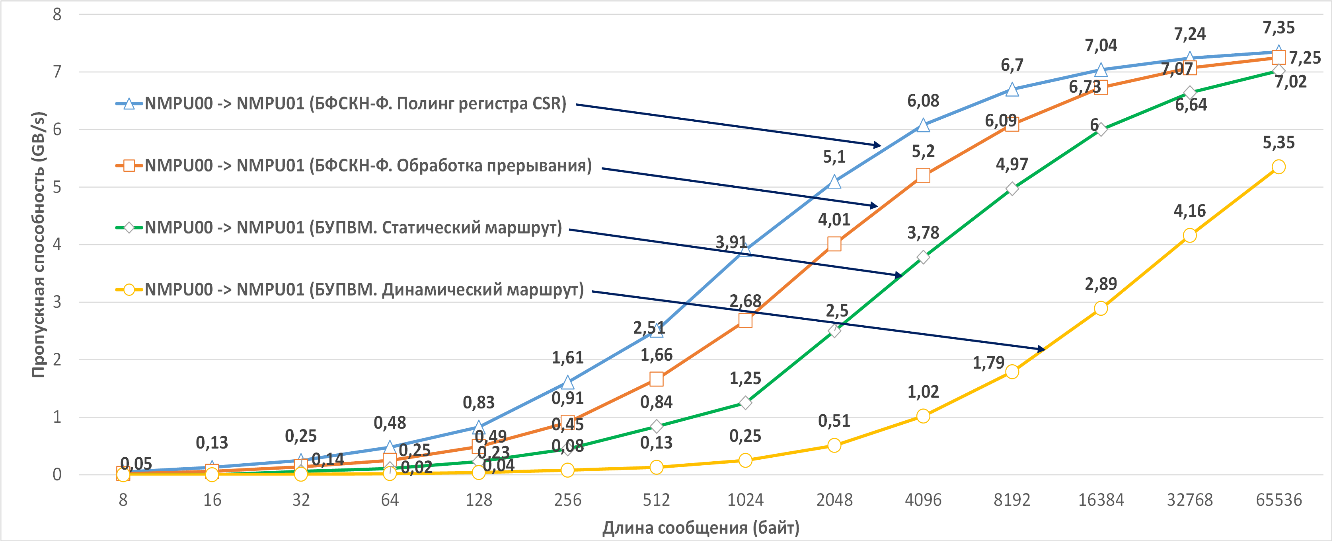


Рисунок 4. Пропускные способности для разных пересылок сообщений между узлами NMPU00 и NMPU01

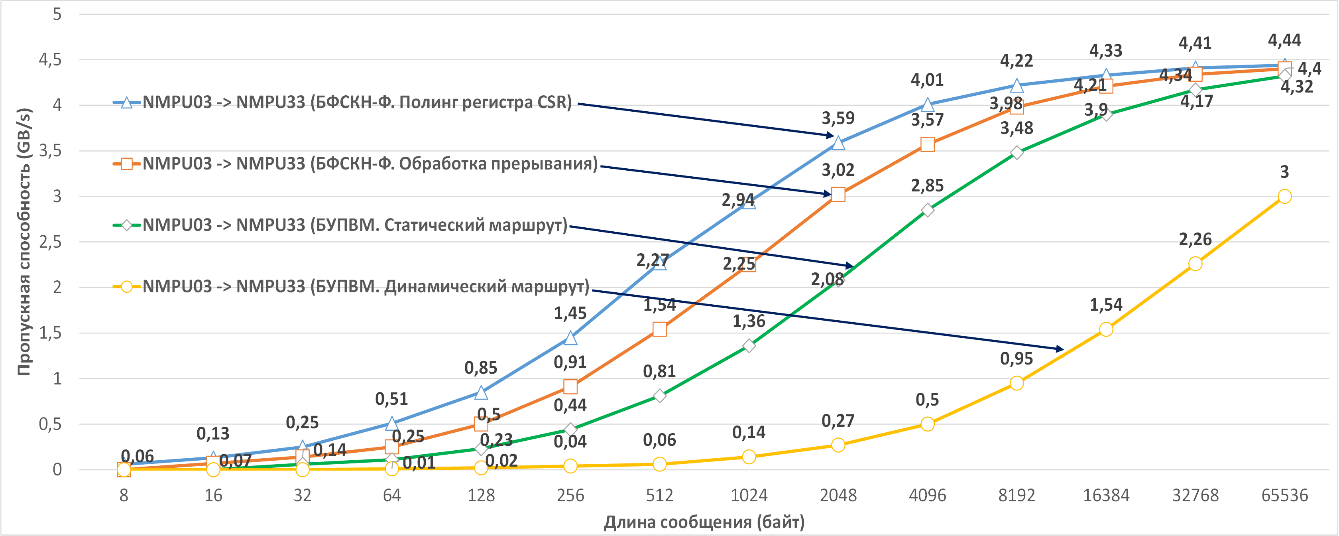


Рисунок 5. Пропускные способности для разных пересылок сообщений между узлами NMPU03 и NMPU33

Завершая рассмотрение этой подгруппы, рассмотрим контроль особых случаев при выполнении функций БУПВ/М. Для этого используются переменные для кодов возврата, и событийные переменные. Например, обращение к MP\_send имеет вид:

**MP\_send (MP\_process\_id** Dest,

**void** \*local\_memory\_source,

**int** num\_word,

**MP\_code** V1,

**MP\_code** V2,

**MP\_event** V3E)

где: Dest — это логический номер процесса-получателя; local\_memory\_source — указатель на начало выдаваемого сообщения из текущего процесса; num\_word — количество двойных слов в сообщении; V3E — событийная переменная, перед входом в MP\_sendона должна иметь значение NO, а при завершении выдачи сообщения из процесса-поставщика она устанавливается в YES.

Имеются переменные V1 и V2 для кодов возврата выполнения MP\_send. Для их объяснения опишем в общих чертах процесс выполнения MP\_send.

Выполнение MP\_send трехфазное: *фаза 1* — проверка параметров обращения, захват и настройка ресурсов маршрута, запуск асинхронной передачи сообщений по этому маршруту, выход в процесс параллельной программы; *фаза 2* — выдача сообщения из процесса-поставщика, фиксация окончания этой выдачи через его компорт, освобождение уже ненужной части ресурсов маршрута; *фаза 3* — условно выделенная фаза, означает, что пересылка завершена окончанием приема через компорт процесса-получателя, освобождается оставшаяся часть ресурсов маршрута.

Действия каждой из фаз могут быть выполнены или нет. Некоторые из причин фатальны, другие — временные и обходятся повторным обращением к MP\_send. Особенности завершения фазы 1 указываются кодами возврата в V1 и V2, переменная V2 используется и для записи кода возврата фазы 2.

Переменная V1 отображает результат проверки параметров обращения и допустимость обращения к MP\_send: V1=0 – обращение допустимо и параметры заданы правильно; V1=1,2…6 – разные аварийные ситуации.

Переменная V2 для фазы 1 (если V3E=NO) отображает результат захвата и настройки ресурсов маршрута, запуска передачи сообщения: V2=0 — ресурсы маршрута захвачены и настроены, началась пересылка сообщения через компорт процесса-поставщика; V2=1 — маршрут в процесс-получатель не X-маршрут, но его ресурсы заняты другими пересылками, поэтому передача сообщения не началась; V2=2 — маршрут в процесс-получатель является X-маршрутом, но в данный момент нет свободного блока MDMAC.

Переменная V2 для фазы 2 (если V3E=YES) отображает результат выдачи сообщения из процесса-поставщика: V2=3 — обнаружены ошибки в работе передающего компорта процесса-поставщика, неопределенная ситуация по передаче сообщения, часть маршрута процесса-поставщика освобождена; V2=4 — сообщение выдано из процесса-поставщика (но еще может не быть принятым в процессе-получателе), часть маршрута процесса-поставщика освобождена.

В процессе-поставщике можно узнать о получении процессом-поставщиком сообщения. Для этого можно использовать функцию MP\_wait\_event для событийной переменной, заданной как V3E в обращении к MP\_recv процесса-получателя.

# **Подгруппа функций пересылки областей памяти.**

Функции данной подгруппы производят пересылку содержимого памяти с одного глобального адреса (считываемый блок) в другой (записываемый блок) через среднескоростную сеть и блок MDMAC. Блок MDMAC выбирается из пяти имеющихся в процессоре, какой оказывается свободным. Адресация памяти для считываемого и записываемого блока одномерная или двумерная, в произвольном сочетании, это отражено в именах функций. Пересылка происходит асинхронно, одновременно с выполнением других операторов процесса, где было обращение к такой функции. По окончании пересылки устанавливается значение YES указанной в обращении событийной переменной. Особые ситуации, возникающие при выполнении пересылки отмечаются кодами возврата в переменных, указанных в обращении функциям.

Имеются следующие функции в данной подгруппе: MP\_move, MP\_move\_1D\_2D, MP\_move\_2D\_1D, MP\_move\_2D\_2D.

# **Подгруппа функций работы с сигналами**

Каждый процесс БУПВ/М может получить до 16-ти разных сигналов. Имеются следующие функции: MP\_wait\_signal, MP\_wait\_signal\_with\_timeout — неограниченное или ограниченное во времени ожидание получения заданного набора сигналов в текущем процессе; MP\_wait\_any\_signal, MP\_wait\_any\_signal\_with\_timeout — неограниченное или ограниченное во времени ожидание хотя бы одного сигнала из заданного набора в текущем процессе с выдачей номера этого сигнала; MP\_send\_signal — выдача установленного в обращении набора сигналов в заданный процесс; MP\_broadcast\_signal — выдача установленного в обращении набора сигналов в процессы заданного списка; MP\_drop\_signal — сброс заданных сигналов в текущем процессе.

Ожидание сигналов в процессе не совмещается с выполнением других операторов этого процесса, т.е. процесс на операторе ожидания «зависает».

# **Подгруппа функций работы с событиями**

Переменная типа событие (MP\_event) может иметь два значения: NO — событие не произошло; YES — событие произошло. Если событие не произошло, то с переменной может быть связано множество процессов, ожидающих его. Эти процессы находятся в пассивном состоянии. Если связанное с переменной событие наступает, то все ожидающие его процессы из ее списка становятся активными, переменная получает после этого значение NO.

Переменные типа событие могут находиться по любому адресу, они доступны любому процессу параллельной программы. Имеются следующие функции: MP\_wait\_event, MP\_wait\_event\_with\_timeout — неограниченное и ограниченное во времени ожидание указанного события текущим процессом; MP\_set\_event — установка значения YES указанной событийной переменной с активацией ожидающих это событие процессов; MP\_drop\_event — установка значения NO указанной переменной.

# **Подгруппа функций работы с критическими ресурсами**

Переменные с исключительным доступом или mutex-переменные (тип MP\_mutex) служат для управления доступом к ресурсам, с которыми не могут одновременно работать несколько процессов. Mutex-переменная может находиться в состоянии «блокирована» (lock) или «не блокирована» (unlock). Если mutex-переменная блокирована, некоторым процессом, то с ней может быть связан список ожидающих ее разблокировки процессов, они находятся в пассивном состоянии. Mutex-переменная может быть разблокирована только заблокировавшим ее процессом. Если происходит разблокировка mutex-переменной, то при наличии списка ожидающих ее разблокировки процессов выбирается первый из этого списка, и он становится новым процессом, заблокировавшим эту переменную. Остальные процессы списка остаются в ожидании очередной разблокировки этой mutex-переменной. Если же список ожидающих разблокировки mutex-переменных процессов пуст, то она просто переходит в разблокированное состояние. Переменные типа MP\_mutex могут находиться по любому адресу, они доступны любому процессу параллельной программы.

Имеются следующие функции работы с mutex-переменными: MP\_lock — захват текущим процессом (блокировка) указанной в обращении mutex-переменной, если она незаблокирована, то захват ее, если заблокирована, то текущий процесс подключается в список ожидающих ее разблокировки пассивируется; MP\_unlock — текущий процесс, который должен быть обязательно заблокировавшим указанную mutex-переменную, снимает ее блокировку, иначе — аварийная ситуация.

# **Подгруппы функций доступа к памяти по глобальному адресу, барьерной синхронизации, самоидентификации и времени.**

Кроме функций пересылки через блок MDMAC имеются функции MP\_put и MP\_get пересылки также через среднескоростную сеть, но с использованием процессорных ядер узлов. Особенность этих функций еще в том, что один из участков памяти находится в локальной памяти узла.

Для работы с памятью через глобальное адресное пространство оказывается полезной функция MP\_global\_address, которая по физическому адресу выдает соответствующий глобальный адрес.

Подгруппа функций барьерной синхронизации включает: MP\_local\_barrier — барьерная синхронизация внутри кластера; MP\_global\_barrier — барьерная синхронизация всех процессов программы;MP\_pair\_barrier — парная барьерная синхронизация текущего процесса с указанным в обращении к этой функции процессом.

Следует иметь ввиду, что выполняемый процесс может иметь запущенные из него асинхронные процессы пересылки сообщений. Таких асинхронных процессов для одного процесса может быть до восьми (см. раздел 2.3). Эти асинхронные процессы при барьерной синхронизации не останавливаются, останавливается только процесс выборки и выполнения команд в процессорном ядре узла. Если требуется полная остановка процесса, включая его асинхронные процессы, то пользователь может до обращения к функции барьерной синхронизации обратиться к функциям MP\_wait\_event для ожидания событий окончания этих асинхронных процессов.

Еще одна функция данной подгруппы — MP\_global\_exception. Обращение к этой функции позволяет из любого процесса параллельной программы произвести останов ее выполнения с некоторым указанным в обращении кодом особой ситуации. В зависимости от значения этого кода могут произойти действия по оценке состояния параллельной программы и оборудования процессора, после чего выполнение программы может либо возобновиться, либо произойти ее аварийное завершение.

Подгруппа самоидентификации и времени содержит следующие функции: MP\_my\_id — текущий процесс может определить свой логический и физический номер; MP\_node\_gettime — текущий процесс выдает в специальном формате время от начала выполнения программы в узле, включая время его простоев, используется локальный таймер узла; MP\_gettime — текущий процесс выдает в специальном формате общее для всего процессора время, оно берется из центрального узла и ведется по таймеру этого узла.

Функции времени могут используются и неявно при автоматической трассировке событий, происходящих при выполнении параллельной программы.

# **Режимы запуска программы, функции трассировки и отладки.**

Для параллельной программы, выполняемой на подключенном к хост-процессору процессоре-ускорителе, обычно предусматриваются расширенные ее взаимодействия с хост-процессором. В БУПВ/М со стороны параллельной программы для связи с хост-процессором используется центральный узел (далее — ССPU), обеспечивается следующее:

* управление параллельной программой от хост-процессора через процесс на узле CCPU с использованием механизма работы в «контрольных точках»;
* ввод/вывод данных, через процесс на узле CCPU;
* управляемый со стороны процесса на узле CCPU сбор трассировочной информации о процессах на любых узлах.

*Контрольная точка* — особое состояние параллельной программы после выполнения глобальной барьерной синхронизации посредством обращения во всех процессах к функциям MP\_global\_barrier\_and\_wait\_event, при выполнении которой останавливаются все процессы кроме CCPU-процесса. Каждый из остановленных процессов ожидает события в указанной событийной переменной в обращении к MP\_global\_barrier\_and\_wait\_event. Эта переменная указывается во всех процессах произвольно.

Выполняемая процессором параллельная программа может многократно переходить в состояние «контрольной точки» и выходить из нее. Выход из контрольной точки управляется установкой значения YES событийным переменным, по которым процессы ожидают выхода из барьера.

Общая конфигурация процессов на NM6408MP и хост-процессоре и схема их взаимодействия представлены на рис.6. Двумя большими овалами с подписями «Процессор NM6408MP» и «Хост-процессор» на рис.6 выделены соответственно процессы, на NM6408MP и хост-процессоре.

Процессы на NM6408MP обозначены окружностями, выделен процесс на узле CCPU (далее CCPU-процесс), остальные процессы обозначены pp1, pp2, … ppn.

Процессы на хост-процессоре также обозначены окружностями, при этом выделены:

* главный процесс (Host-процесс), он взаимодействует с CCPU-процессом, выполняет управляющие функции, функции ввода/вывода и функции управления трассировкой в процессах параллельной программы.
* вспомогательные (теневые) T-процессы T-CCPU, T-pp1, …T-ppn, каждый из которых привязан к процессу параллельной программы, они требуются только для получения трассировочной информации, управление этим производит Host-процесс (отображено на рисунке односторонними стрелками).

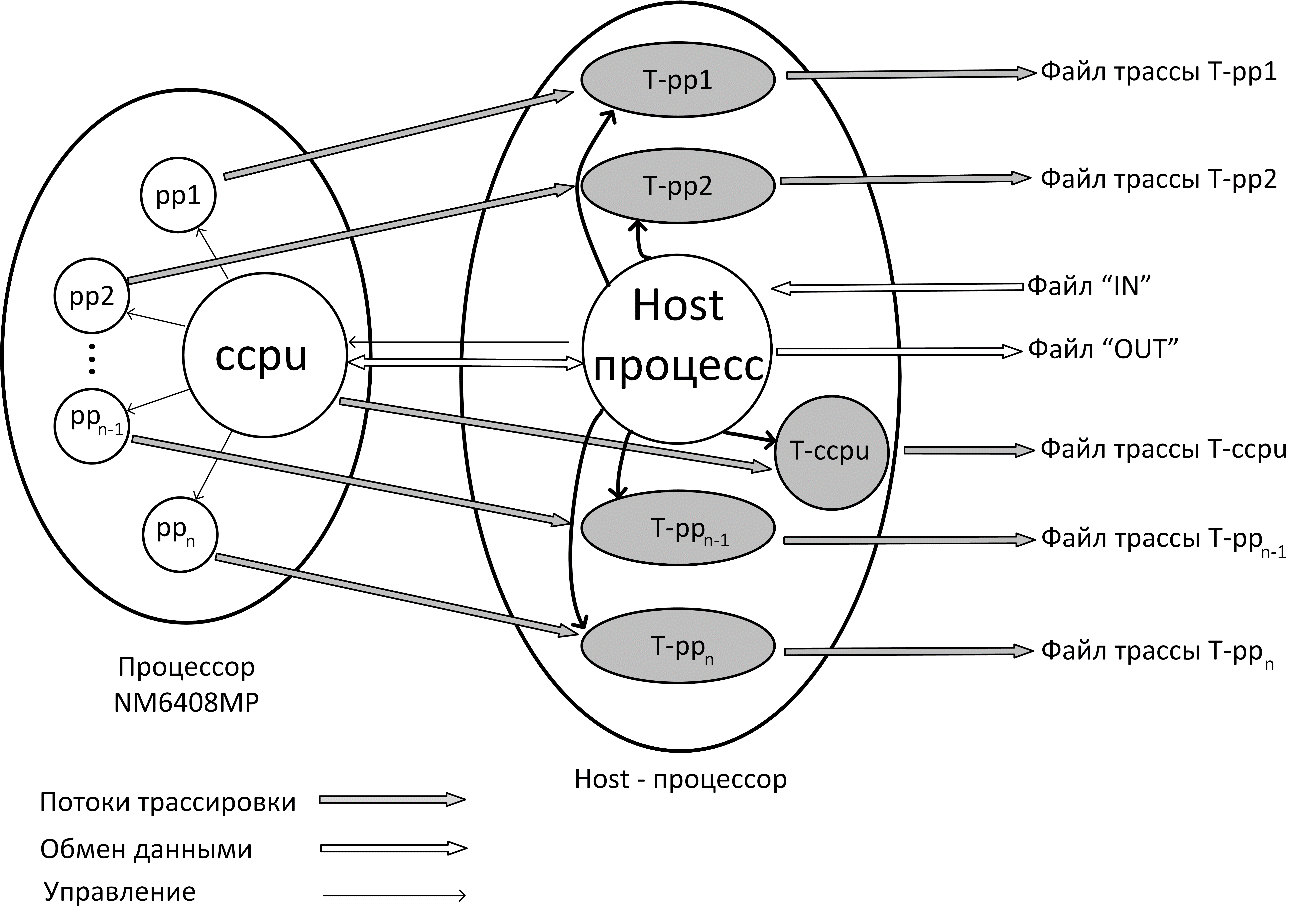


Рисунок 6. Конфигурация процессов и их взаимодействия для самого общего режима отладки и трассировки параллельной программы

Файлы группы «IN» — это файл для функции MP\_load кода параллельной программы, файл привязки логических номеров процессов к физическим адресам узлов, файлы с исходными данными параллельной программы.

Файлы группы «OUT» — это файлы с результатами выполнения параллельной программы.

Файлы T-CCPU, T-pp1, T-pp2, …T-ppn — это трассы событий в соответствующих им процессах параллельной программы. Трассы представлены в упакованном виде, для их чтения на хост-процессоре имеется специальная программа-распаковщик в удобный символьный вид. Планируется разработать программу их графической визуализации с интерактивным графическим интерфейсом.

Предполагаются три вида взаимодействия процессов параллельной программы и хост-процессора:

* управляющие воздействия от Host-процесса на CCPU-процесс, а также от CCPU-процесса на pp(i)-процессы, обозначены на рис.6 однонаправленными стрелками;
* обмен данными при вводе/выводе, только между Host-процессом и CCPU-процессом, обозначается на рис.6 двунаправленной двойной стрелкой без заливки;
* пересылка трассировочной информации (трасс), обозначена на рис.6 однонаправленными двойными стрелками с серой заливкой.

Представляются важными следующие частные случаи запуска и выполнения параллельной программы:

**Режим А** — Счетный режим без трассировки, только с выводом результатов в конце выполняемой программы (формально – это одна контрольная точка в конце программы)

**Режим B** — Отладочный режим с множеством контрольных точек и вводом/выводом на них, но без трассировки.

**Режим С** — Отладочный режим с множеством контрольных точек и вводом/выводом на них, с трассировкой только ССPU-процесса.

**Режим D** — Отладочный режим с множеством контрольных точек и вводом/выводом на них, с трассировкой любого процесса, причем с возможностью управления этими трассировками от хост-процессора через CCPU-процесс.

Находящийся в контрольной точке CCPU-процесс может связаться с хост-процессором посредством парной барьерной синхронизации. Со стороны CCPU-процесса это выполняется обращением к функции MP\_checkpoint, а со стороны хост-процессора — функцией MP\_wait\_checkpoint для ожидания контрольной точки. Выход из контрольной точки в CCPU-процессе выполняется обращением к MP\_drop\_checkpoint.

Контрольная точка фиксируется на фрейме CCPU-процесса в полях «Состояние процесса» и «Счетчик контрольных точек». По этой причине даже если хост-процессор обратится к MP\_wait\_checkpoint раньше, чем в CCPU-процессе будет выполнено MP\_drop\_checkpoint для предыдущей контрольной точки, то со старой контрольной точкой хост-процессор не сможет синхронизоваться.

В заключение описания функций БУПВ/М осталось рассмотреть функции трассировки событий, используемые в процессах параллельной программы (для управления выдачей трасс в хост-процессор) и на хост-процессоре (для управления приемом трасс в хост-процессоре).

Для управления выдачей трасс имеются функции: MP\_trace\_on — включение трассировки системных событий; MP\_trace\_off — отключение трассировки системных событий; MP\_trace\_event — выдача в трассу сообщения о событии пользователя; MP\_run\_trace\_out, MP\_stop\_trace\_out — включение и выключение выдачи в хост-процессор фрагментов трассы из кольцевого буфера трассировки.

Для управления приемом трасс в хост-процессоре используются функции: MP\_run\_trace\_in, MP\_stop\_trace\_in — включение и выключение приема фрагментов трасс в указанном теневом процессе.

# **Выводы**

Реализация выбранных в БУПВ/М моделей параллельных программ на процессоре-ускорителе NM6408MP показывает, что это возможно и даже без потери преимуществ процессора-ускорителя. Были отмечены и недостатки как системы команд, так и архитектуры процессора-ускорителя, их пришлось компенсировать программно.

Было замечено, что недостающие средства в системе команд NM6408MP для реализации функций БУПВ/М появились в некоторых новых разработках процессоров, например, реализованы в блоках синхронизации процессов микропроцессора Xavier. Такой набор средств был введен в проекте перспективного микропроцессора mPX [9], который может оказаться развитием NM6408MP.

Еще один вывод состоит в том, что в современных условиях новый процессор-ускоритель должен разрабатываться сразу с учетом того, что он будет применяться в вычислительных узлах суперкомпьютеров как элемент сети таких процессоров. Для этого необходимо предусматривать наличие большого количества внекристальных высокоскоростных линков. В качестве примеров правильного решения можно указать израильский процессор Gaudi [5] и английский процессор Colossus [3, 4].

# **Заключение**

Авторы считают, что в данной работе новыми являются следующие положения и результаты:

1. Разработан набор функций, поддерживающих, с одной стороны, общепринятые модели параллельных программ типа MPI и PGAS, а с другой стороны, учитывающих специфику процессора-ускорителя NM6408MP.
2. Проведена реализация разработанного набора функций на специализированном процессоре ускорителе NM6408MP, в которой не потеряна возможность эффективного использования особенностей процессора, а недостающие возможности аппаратных средств компенсированы программно.
3. Принципиальное значение разработки рассмотренного набора функций и его эффективной реализации доказывают на примере отечественного процессора-ускорителя правильность актуальной в мире выбранной тенденции применения простых общепринятых моделей параллельных программ для программирования сложных специализированных процессоров-ускорителей.
4. Исходя из полученного опыта разработки, сформулированы направления развития архитектуры микропроцессоров семейства NeuroMatrix.

Главная задача статьи состояла в объяснении выбранного набора функций БУПВ/М, с учетом особенностей NM6408MP и краткого описания этих функций. В процессе опытной эксплуатации выбранный набор функций неизбежно будет дорабатываться. Ожидается, что появятся функции контроля целостности программы и сервисные функции графического представления трасс на хост-процессоре в стиле системы Vampir.

Новая задача состоит в разработке на базе БУПВ/М системы БУПВ/МХ для множества NM6408MP, подключенных к материнской плате с хост-процессорами в составе вычислительного узла суперкомпьютера.

**Литература**

[1] Durant L. [*et al.*] Inside Volta: The World’s Most Advanced Data Center GPU, 10 may 2017 //

[2] NVIDIA A100 Tensor Core GPU Architecture, May 2020, 83 pp.

[3] Jia Z. [*et al.*] Dissecting the Graphcore IPU Architecture via Microbenchmarking. // Technical report, High Performance Computing R&D Team Citadel, 7 Dec 2019 - 91 pp.

[4] Toon N. Introducing 2nd generation IPU systems for AI at scale //15 july 2020//

[5] Gwennap L. Habana Offers Gaudi for AI Training // Microprocessor report, June 17, 2019, 4 pp.

[6] Горбунов В.С., Эйсымонт Л. К. Экзафлопсный барьер: проблемы и решения // Открытые системы, №6, 2010 – С. 12-15.

[7] Эйсымонт А.Л., Черников В.М., Черников Ан.В., Черников Ал.В., Косоруков Д.Е., Насонов И.И., Комлев А.А. Гетерогенная многопроцессорная система на кристалле с производительностью 512 Gflops // Системы высокой доступности, 2018, т.14, №3, с.49-56.

[8] Биконов Д.В., Пузиков А.Д., Сивцов А.С., Эйсымонт Л.К. Трехуровневая система параллельного программирования гибридного 21-ядерного скалярно-векторного микропроцессора NM6408MP. // Вопросы кибербезопасности, номер 4, 2019, стр.22-34.

[9] Эйсымонт Л.К., Никитин А.И., Биконов Д.В., Бражкин А.А, Пеплов И.C., Федоренко П.П., Ермаков С.С., Эйсымонт А.Л., Комлев А.А. Исследовательский проект массово-параллельного процессора на базе мультитредовых ядер со специализированными ускорителями // Вопросы кибербезопасности, N3, 2020, c.22-39.

[10] Кудрявцев М., Эйсымонт Л., Мошкин Д., Полунин М. Суперкластеры — между прошлым и будущим // Открытые системы, №8, 2008.

[11] Речинский А., Горбунов В., Эйсымонт Л. Суперкластер с глобально адресуемой памятью // Открытые системы, №7, 2011, с.21-25.

[12] Левин В.К. и др. Коммуникационная сеть МВС-Экспресс // Информационные технологии и выислительные системы. 2014, №1, стр.10-24.

[13] Биконов Д.В., Бражкин А.X., Пузиков А.Д., Сивцов А.С., Черников А.В., Эйсымонт Л.К. Высокоуровневая система параллельного программирования для сетей многоядерных гибридных процессоров. 2020 (в печати)