УДК 004.27 DOI: 10.31114/2078-7707-2020-2-87-94

Методы и подходы к повышению надежности параллельной потоковой вычислительной системы

Д.Н. Змеев, Н.Н. Левченко, А.С. Окунев

Институт проблем проектирования в микроэлектронике РАН, г. Москва, nick@ippm.ru

***Аннотация* — В статье описаны основные подходы к обеспечению надежности аппаратно-программных средств параллельной потоковой вычислительной системы. Подход, связанный с динамическим перераспределением вычислений между исполнительными устройствами, делает возможным организацию дополнительных каналов связи между исполнительными устройствами и коммутаторами вычислительного ядра. Описаны варианты предотвращения переполнения ассоциативной памяти ключей: откачка и подкачка токенов в динамическом режиме работы и разбиение задачи на этапы. Также представлен общий алгоритм работы системы при ее восстановлении после сбоя/отказа и предложены варианты алгоритмов работы создания локальных контрольных точек. Описаны преимущества глобальной распределенной ассоциативно-вычислительная среды с точки зрения повышения надежности системы. Приведены результаты экспериментов на программной блочно-регистровой модели, связанных с оценкой средств восстановления и обеспечения надежности параллельной потоковой вычислительной системы.**

***Ключевые слова* — параллельная потоковая вычислительная система, надежность вычислений, потоковая модель вычислений, локальная контрольная точка.**

I. Введение

На протяжении последних лет разработчики суперкомпьютерных вычислительных систем во всем мире пытаются создать системы с реальной производительностью в эксафлопс (10^{18} flops – операций с плавающей точкой в секунду) и более. При этом стало очевидным, что без реализации новых подходов в архитектуре, параллельном программировании, а также без создания средств отказоустойчивости этих систем решить названную задачу затруднительно [1, 2].

В ИППМ РАН в настоящее время ведется работа над потоковой моделью вычислений [3] и реализующей ее архитектурой, разработана новая модель программирования, предлагаются новые подходы к реализации параллельной потоковой вычислительной системы (ППВС).

Одной из главных проблем, которую требуется решить в процессе проектирования ППВС, является вопрос надежности, так как предполагается, что эта система будет иметь в своем составе сотни тысяч, миллионы ядер. Под надежностью подразумевается обеспечение отказоустойчивости системы путем создания средств (как программных, так и аппаратных) восстановления работоспособности после сбоя или отказа. В потоковых вычислительных системах из-за сложной структуры параллельных вычислительных процессов возникают трудности с локализацией источника ошибки, ее обнаружения, изоляции отказавшего вычислительного ядра и восстановления нормального функционирования вычислительной системы после сбоя или отказа.

В статье описываются подходы к обеспечению надежности ППВС, связанные с динамическим перераспределением вычислений между исполнительными устройствами, преодолением переполнения ассоциативной памяти ключей процессора сопоставления, аппаратной поддержкой создания локальных контрольных точек, а также с реализацией глобальной распределенной ассоциативно-вычислительной среды. Однако, для повышения надежности высокопроизводительных вычислительных систем могут применяться и методы не связанные напрямую с архитектурой системы, например, описанные в работе [4].

II. Потоковая модель вычислений и реализующая ее архитектура

Потоковая модель вычислений с динамически формируемым контекстом (рис. 1) базируется на активации по готовности данных неделимых вычислительных квантов. Под вычислительным квантом понимается программный узел, который после своей активации обрабатывается до конца без приостановки вычислительного процесса и ожидания каких-либо дополнительных внешних данных.

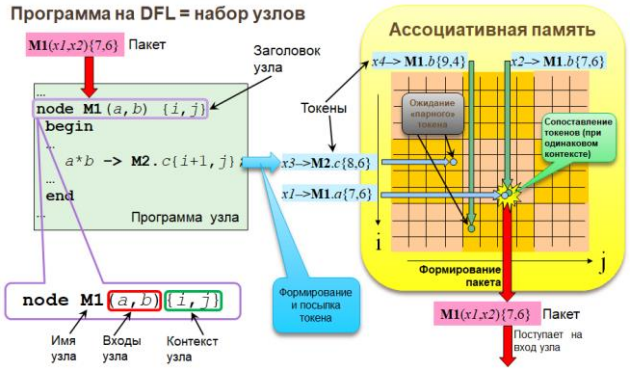

Рис. 1. Потоковая модель вычислений с динамически формируемым контекстом

В свою очередь, программа для ППВС представляет собой набор описаний программных узлов. Активация любого программного узла происходит только после

того, как на все его входы (одного конкретного программного узла с определенным именем и контекстом) поступят все необходимые токены, в которых содержится данное (операнд), набор служебных полей и ключ (индекс), однозначно определяющий местоположение этого данного в виртуальном адресном пространстве задачи.

На вход программного узла данные поступают в виде пакета, который образуется в результате сопоставления готовых к исполнению токенов. Пакет содержит данные, над которыми будет выполнен код программы в этом узле. В результате работы программного узла вычисляются новые данные (исключительно на основе значений входов и атрибутов ключа) и отсылаются в виде токенов на другие программные узлы, причем атрибуты ключа адресата вычисляются непосредственно в этой же программе в динамике.

Архитектура ППВС (рис. 2) [5], которая реализует потоковую модель вычислений с динамически формируемым контекстом, представляет собой многоядерную масштабируемую вычислительную систему. Между ядрами в системе передаются единицы информации в виде токенов. Коммутация между ядрами осуществляется на основе значения номера ядра, вырабатываемого блоком хеширования на основе настраиваемой функции распределения вычислений. В состав вычислительного ядра входит процессор сопоставлений (ПС) с ассоциативной памятью ключей и памятью токенов, исполнительные устройства (ИУ), блоки хеширования и внутренний коммутатор токенов.

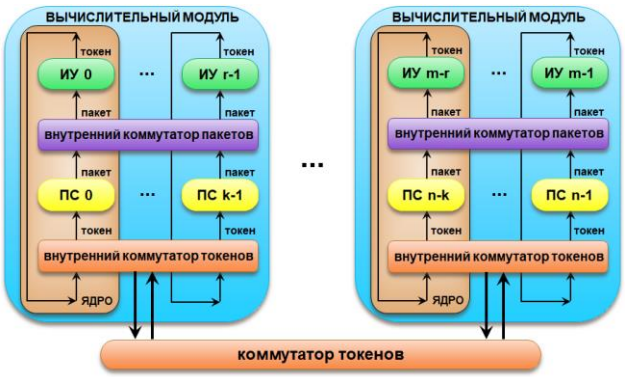

Рис. 2. Архитектура параллельной потоковой вычислительной системы

Вычислительные ядра (ВЯ) в пределах одного кристалла организуются в вычислительные модули (ВМ). Архитектура ППВС масштабируема и при увеличении числа ядер в системе падение реальной производительности на задачах со сложно организованными данными происходит существенно медленнее, чем при решении подобных задач на вычислительных системах с классической архитектурой.

III. Основные подходы к обеспечению надежности ППВС

Параллельная потоковая вычислительная система имеет кардинальные отличия от традиционных вычислительных систем. Исходя из анализа этих особенностей потоковой модели вычислений и ее архитектуры, специфики работы отдельных узлов и блоков параллельной потоковой вычислительной системы прорабатываются следующие основные подходы к обеспечению надежности аппаратно-программных средств ППВС:

- преодоление переполнения ассоциативной памяти ключей (АПК);
- локальное динамическое перераспределение вычислений;
- восстановление после сбоя или отказа ИУ;
- аппаратная поддержка локальных контрольных точек;
- переход к глобальной распределенной ассоциативно-вычислительной среде.

A. Преодоление переполнения АПК

Одной из главных проблем всех потоковых систем, в частности, параллельной потоковой вычислительной системы, является проблема переполнения ассоциативной памяти ключей процессора сопоставления ключами токенов, ожидающих взаимодействия с другими токенами. Это критически сказывается на надежности вычислительной системы.

Ситуация переполнения АПК порождается случайным отбором токенов на обработку, в результате чего растет среднее время пребывания ключа токена, ожидающего сопоставления, в АПК. Если не предпринимать никаких мер, то будет нарастать среднее время ожидания токеном в АПК прихода «парного» токена. Вместе с этим будет увеличиваться общее количество токенов, одновременно находящихся в АПК.

На возможность переполнения АПК влияет также:

- кратность токенов (в случае взаимодействия однократных токенов, происходит их взаимное уничтожение, а при наличии кратности – один из токенов будет оставаться в АПК;
- недостаточное число ИУ на один ПС или обработка ИУ большого программного узла;
- недостаточное число ПС на имеющееся число ИУ в вычислительном модуле;
- неэффективная функция распределения.

Аппаратно-программные методы предотвращения переполнения ассоциативной части АПК связаны с откачкой и подкачкой токенов в динамическом режиме работы (работа с токеном «заглушка»), а также с разбиением задачи на этапы. Более сложный и аппаратно-затратный метод реализации, связанный с построением иерархии ассоциативной памяти, описан в работе [6].

При реализации метода откачки и подкачки токенов в динамическом режиме вводится новый тип токена – токен «заглушка». На основе информации о загрузке АПК устройством управления процессора сопоставления принимается решение о необходимости откачки определенного диапазона токенов. Если откачка токенов определенного диапазона достаточна для продолжения работы без переполнения АПК, то задача выполняется, если АПК продолжает переполняться, то выбирается другой диапазон токенов для откачки.

Метод распределения вычислений во времени с использованием этапов является еще одним методом, разработанным для предотвращения переполнения АПК в ППВС. Задача разбивается на этапы с помощью задаваемых пользователем хеш-функций, что является универсальным методом планирования вычислений в ППВС. Все токены вычислительного процесса разделяются с помощью хэш-функции на некоторое количество групп, которые называются «временными этапами». Все этапы делятся на «активные» и «пассивные». В АПК процессора сопоставлений во время вычислений присутствуют ключи токенов только «активных» этапов. Токены «пассивных» этапов в это время находятся в прямоадресуемой памяти, в которой сравнения по ключам не происходит. Для включения в вычислительный процесс токенов «пассивных» этапов их необходимо активировать. Задача, которая будет разбита с помощью функции распределения по времени на этапы при аппаратной поддержке этого режима работы, потребует существенно меньшего объема АПК для ее выполнения на ППВС без каких-либо блокировок по переполнению, что позволит на меньшем объеме физической АПК пропускать задачи большего масштаба.

B. Локальное динамическое перераспределение вычислений

Данный подход заключается в организации дополнительных каналов связи между исполнительными устройствами для организации «переливов» как пакетов, так и токенов. Такое перераспределение вычислений в автоматическом режиме между ИУ позволяет даже при выходе из строя отдельных каналов связи между исполнительными устройствами и внутримодульным коммутатором пакетов, а также внутримодульным коммутатором токенов продолжать функционирование вычислительного модуля (рис. 3). Это же решение позволяет фактически отказаться от использования внутримодульного коммутатора пакетов, поскольку пакеты при занятости «своего» ИУ смогут по таким каналам передаваться на соседние – свободные ИУ. В свою очередь, токены образованные в результате выполнения программы узла при выходе из строя «своего» канала связи с внутримодульным коммутатором токенов смогут поступить в него через соседние ИУ.

C. Восстановление после сбоя или отказа ИУ

Для обеспечения нормального выполнения программы при сбое/отказе ИУ нужно зафиксировать следующую информацию о программе узла, которая выполнялась на момент отказа или сбоя в ИУ:

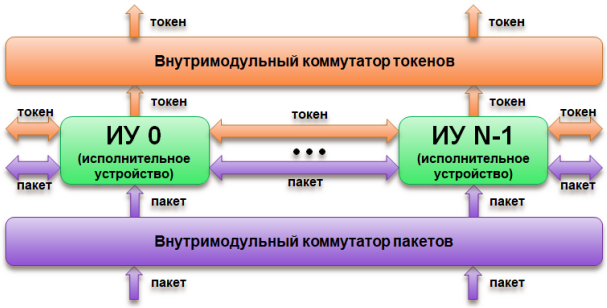

Рис. 3. Динамическое перераспределение вычислений между ИУ

- должен быть сохранен пакет, принятый для исполнения, обработка которого не завершилась в результате сбоя или отказа;
- необходимо запомнить все токены, которые выдавались из этого ИУ до момента прерывания и относились к пакету, обрабатываемому в данном ИУ.

Первый вариант реализации механизма восстановления системы после сбоя или отказа ИУ заключается в следующем. В состав аппаратуры включаются два буфера: буфер пакетов на входе ИУ и буфер токенов ИУ, который располагается на выходе ИУ.

Пока не закончилась обработка пакета, токены из буфера ИУ не передаются через коммутатор токенов в ПС. В обычном, бессбойном режиме работы ИУ буфер токенов начинает освобождаться только после завершения работы программного узла (то есть при завершении обработки пакета), и только в этот момент токены, относящиеся к данному программному узлу передаются для дальнейшей обработки. При сбое в ИУ происходит перезапуск пакета, а токены из буфера уничтожаются простым стиранием. В режиме восстановления работоспособности ИУ по сбою происходит трехкратный перезапуск на выполнение пакета, вызвавшего сбой. Если перезапуск неуспешный, фиксируется отказ ИУ, после чего это ИУ исключается из работы системы. При отказе ИУ в момент запуска механизма восстановления оно закрывается для приема пакетов от ПС, а пакеты, обработка которых не может быть продолжена на этом ИУ перенаправляются через коммутатор пакетов на другое свободное ИУ (например, с помощью дополнительных каналов связи).

Второй вариант реализации механизма восстановления системы при сбое или отказе ИУ будет выглядеть следующим образом. Каждому ИУ присваивается уникальный номер. Такое присвоение номера ИУ не будет отражаться на принципе выбора свободного ИУ для обработки пакета. Кроме того, каждый пакет, который обрабатывается на ИУ (он хранится во входном буфере пакетов до момента окончания работы узла) и следующий за ним, пронумеровываются. Это необходимо для исключения конфликтных ситуаций при работе механизма восстановления. Таким образом в состав ключа каждого токена вводятся два дополнительных поля - номер ИУ и номер пакета.

В обычном режиме работы вычислительной системы токены направляются из ИУ в ПС (буфер токенов на выходе ИУ в этом варианте работы отсутствует) и пока не закончилась работа узла, они записываются в памяти ПС своего и других ядер, однако обработка этих токенов откладывается. Это легко реализуется с помощью специального технологического разряда в ключе токена. Закончив выполнение узла, ИУ направляет в ПС всех ядер специальный токен – «снять блокировку запуска», подтверждающий завершение работы узла. Этот токен инициирует выдачу всех отложенных токенов в ПС на поиск (так как все разряды ключа этого токена маскируются за исключением дополнительных полей) и работа продолжается. Этот специальный токен имеет бесконечную кратность, рассылается во все ПС и никогда не записывается в память ПС.

При сбое или отказе ИУ посылает в ПС специальный токен – «стереть при сбое», стирающий все отложенные до завершения работы узла токены в памяти ПС. Токен этого типа имеет те же признаки, что и токен «снять блокировку запуска». Одновременно с этим отправляется сообщение в устройство управления ПС о начале работы механизма восстановления после сбоя ИУ. В этом режиме также, как и в вышеописанном первом варианте работы аппаратуры восстановления, происходит трехкратный перезапуск пакета, вызвавшего сбой, на выполнение. Если перезапуск неуспешный, фиксируется отказ ИУ, после чего это ИУ исключается из работы системы. Отказавшее ИУ в момент запуска механизма восстановления также закрывается для приема пакетов от ПС, а пакеты, обработка которых не может быть продолжена на этом ИУ, перенаправляются через коммутатор пакетов на другое свободное ИУ.

Работа всей вычислительной системы не прерывается, соответствующая информация об отказавшем ИУ фиксируется в устройстве управления ПС и на хост-машине, которая производит нужные действия по реконфигурации. Этот вариант экспериментальный, так как доля накладных расходов для него будет очень большой.

D. Аппаратная поддержка создания локальных контрольных точек

Стандартная работа вычислительной системы выглядит как непрерывный процесс приема и передачи токенов, как между вычислительными ядрами, так и между ИУ и ПС внутри ядра. При фиксации сбоя или отказа в системе средства восстановления работают по следующему алгоритму (рис. 4).

Для работы локальных средств восстановления в вычислительном ядре и вычислительном модуле ключевой вопрос, который необходимо решить – это создание локальной контрольной точки (ЛКТ). Таким образом, разработка механизма формирования ЛКТ решает проблему восстановления работоспособности всей системы после сбоя в динамическом режиме (в рамках одного вычислительного ядра) и перераспределения вычислений в пределах многоядерного вычислительного модуля.

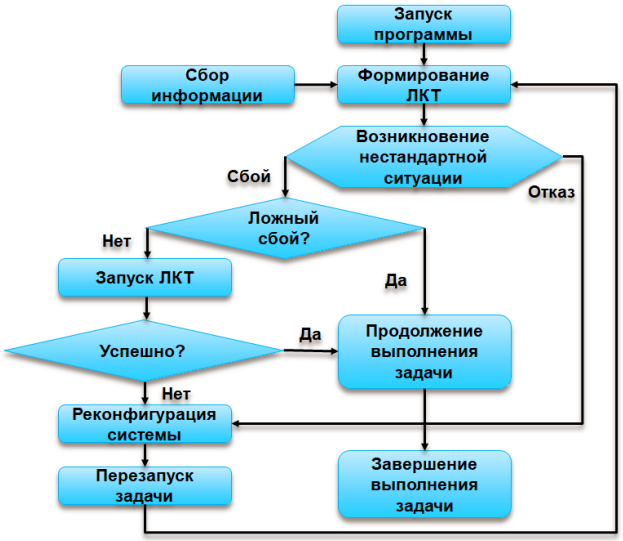

Рис. 4. Общий алгоритм работы системы при ее восстановлении после сбоя/отказа

В процессе выполнения вычислений формируются: входной поток токенов (поступают в процессор сопоставлений); поток пакетов (поступают из процессора сопоставлений на выполнение в ИУ); выходной поток токенов (из исполнительных устройств в коммутатор токенов).

Задание ЛКТ. Первоначально сам программист определяет правило формирования локальных контрольных точек для своей задачи, например, разбивает её на этапы и определяет интервал этапов, через который создается ЛКТ. Можно формировать ЛКТ для программы и аппаратным способом, например, создавать ЛКТ через определенное количество тактов выполнения программы, через каждую итерацию программы или каждый этап. Данный вопрос потребует более глубокого исследования и анализа.

Создание ЛКТ в автоматическом режиме. Опишем создание локальной контрольной точки в автоматическом режиме.

Прежде всего перед созданием ЛКТ производится остановка вычислительного ядра, ИУ заканчивает обработку пакетов, а процессор сопоставлений – токенов. Содержимое ассоциативной памяти ключей, памяти дескрипторов и памяти токенов копируется в память ЛКТ вычислительного ядра. Фиксируется информация об активности этапов. Кроме того, все токены, приходящие на вход процессора сопоставлений с момента останова вычислительного ядра, сохраняются в памяти входных токенов ЛКТ ВЯ. Причем этот процесс продолжается и после запуска вычислительного ядра. Кроме того, токены, которые образуются в результате обработки пакетов, фиксируются в отдельной области памяти ЛКТ ВЯ.

В другом варианте при формировании ЛКТ после записи в память входных токенов и содержимого АПК и памяти токенов запись выходных токенов из ИУ будет осуществляться в специальный буфер ИУ.

Первоначально средства восстановления при сбое могут отработать ситуацию на уровне повтора пакета, при активации которого произошел сбой, на том же или другом ИУ вычислительного ядра. Этот перезапуск пакета (до трех раз) встроен в общий алгоритм восстановления работы ВЯ и не связан с восстановлением из ЛКТ.

Восстановление из ЛКТ. Этот процесс происходит следующим образом.

Фиксируется локализация ситуации в вычислительном ядре, устанавливаются управляющие сигналы и запускается процесс восстановления работы из ЛКТ (необходимо повторить всё, что происходило до сбоя).

Для этого производится останов работы ВЯ с локальным сбросом и начинается процесс восстановления работы из ЛКТ. Опишем варианты реализации этого процесса.

В первом варианте подразумевается, что при выполнении участка вычислительного процесса от одной ЛКТ до другой, кроме «собирания» всех входных токенов, производится подсчет числа всех выходных токенов из ВЯ. При сбое производится локальный останов, фиксируется число выданных на момент сбоя выходных токенов (Nсб.вых.т.), а затем производится перезапуск всех токенов, записанных в памяти ранее при формировании ЛКТ. При восстановлении токены из ИУ накапливаются, подсчитываются, и те токены (в количестве Nсб.вых.т.), которые были уже выданы до сбоя, из ВЯ не выходят. Продолжается выполнение задачи.

Второй вариант восстановления из ЛКТ похож на первый, но в нем при выполнении участка вычислительного процесса от одной ЛКТ до другой будут подсчитываться не выходные токены из ВЯ, а число пакетов, приходящих в ИУ. В момент сбоя фиксируется число выданных на момент сбоя пакетов в ИУ (Nсб.вх.п.), производится запуск сохраненных токенов данной ЛКТ. С начала процесса восстановления фиксируются все пакеты, направляемые на ИУ, и те пакеты, которые уже были активированы до сбоя (в количестве Nсб.вх.п.), еще раз подаваться для выполнения в ИУ не будут, вычисления продолжаются. Этот вариант позволяет сократить время восстановления работы ВЯ из ЛКТ.

Предлагается и третий вариант восстановления из ЛКТ. Токены из ИУ выдаются только после того как пакет, их порождающий, уже «отработал», то есть выполнилась программа узла. Поэтому нам достаточно запомнить какие пакеты выполнились успешно, тогда не надо их повторять при восстановлении. Для этого при выполнении участка вычислительного процесса от одной ЛКТ до другой контексты (номера) всех пакетов запоминаются в памяти ЛКТ (список пакетов). При восстановлении сначала происходит перезапуск всех токенов ЛКТ, а затем контексты всех вновь образуемых пакетов сравниваются с сохраненными в списке пакетами. Если восстановление после сбоя прошло нормально, работа продолжается. Преимуществом реализации этого подхода является меньшие затраты оборудования по сравнению, например, со вторым вариантом, а, главное, полностью исключается некорректная работа средств восстановления.

Для всех вариантов, как уже описывалось, может быть трижды произведен повтор попытки восстановления при возникновении сбойной ситуации, и, если локальное восстановление невозможно, фиксируется отказ ВЯ, приводящий к перезапуску задачи. В многоядерном вычислительном модуле есть свои средства восстановления и в некоторых случаях возможно продолжение работы без отказавшего ВЯ. Решается проблема выхода из строя на уровне ВМ следующим образом. Создается контрольная точка на уровне ВМ и осуществляется переход на работу с другим количеством ВЯ в модуле. Причем, если используется архитектура ГРАВС (описываемая в следующем разделе), перенастройка хэш-функции для распределения вычислений происходит автоматически.

В дальнейшем будут проведены исследования всей иерархии средств восстановления ППВС, начиная с ЛКТ вычислительного ядра и заканчивая глобальными контрольными точками системного уровня.

E. ГРАВС

Глобальная распределенная ассоциативно-вычислительная среда [7] дает заметные преимущества по сравнению с базовой архитектурой ППВС, прежде всего с точки зрения повышения надежности системы.

Архитектура глобальной распределенной ассоциативно-вычислительной среды (ГРАВС) представляет собой набор ассоциативных вычислительных ядер (АВЯ), соединенных между собой коммутатором токенов. Отдельное АВЯ состоит из набора локальных ассоциативных элементов (ЛАЭ), каждый из которых фактически выполняет функции процессора сопоставлений в базовой архитектуре ППВС и процессорных элементов (в базовой архитектуре – исполнительных устройств).

На рис. 5 представлено ассоциативное вычислительное ядро ГРАВС. К новым свойствам архитектуры ГРАВС можно отнести следующее:

- в архитектуре ГРАВС распределение вычислений в отличие от базовой архитектуры ППВС осуществляется до отдельного процессора (кристалла, модуля) или АВЯ, то есть главная особенность новой архитектуры заключается в том, что выход из строя отдельного элемента (например, ЛАЭ) не повлечет за собой необходимости глобального пересчета хэш-функции, используемой для конкретной задачи;
- архитектура ГРАВС предоставляет более удобные механизмы организации гетерогенной архитектуры (при разработке аппаратуры гетерогенность процессоров может заключаться как в различном наборе узлов и блоков самого процессора, так и в различном числе самих ЛАЭ вычислительной системы);
- архитектура ГРАВС повышает надежность вычислительной системы за счет снижения

накладных расходов на восстановление её функционирования при выходе из строя отдельных ЛАЭ;

- архитектура ГРАВС позволяет работать с АВЯ, содержащими ЛАЭ, число которых может быть не кратным степени двойки.

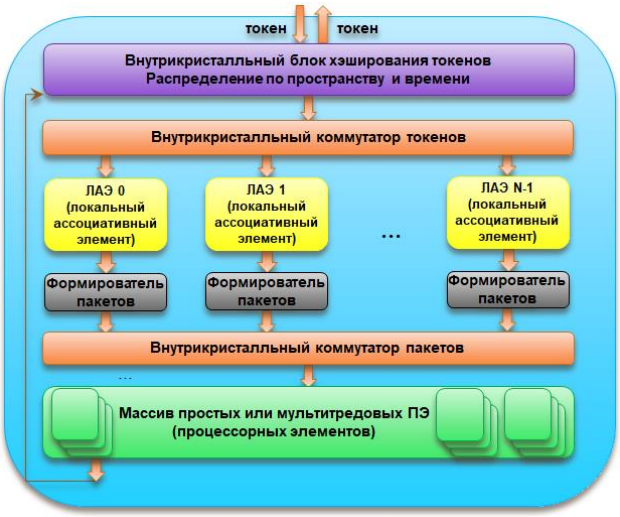

Рис. 5. Ассоциативное вычислительное ядро ГРАВС

Всё вышеперечисленное повышает отказоустойчивость, архитектурную гибкость, производительность, улучшает технологичность изготовления кристаллов, а также обеспечивает более простую настройку параллельной программы для распределения по вычислительным ядрам.

IV. Эксперименты

Проведена серия экспериментов на программной блочно-регистровой модели, связанных с оценкой средств восстановления и обеспечения надежности ППВС.

В частности, оценивалось то, как будут отличаться режимы работы ИУ без накопления и с накоплением выходных токенов (во втором случае, токены выдаются из ИУ только в момент завершения выполнения программного узла). Результаты тестирования таковы, что можно сказать - время прохождения программы с использованием сохранения выходных токенов конкретного программного узла до завершения его выполнения отличается только на доли процентов от времени прохождения программы без сохранения выходных токенов. Это свидетельствует о том, что временные расходы на обеспечение надежности выполнения программного узла в ИУ незначительны.

Часть экспериментов, оценивающих надежность работы архитектуры ГРАВС, проводилась на задачах «МД» (молекулярная динамика) [8] и «ПМ» (перемножение матриц) [9] для конфигурации вычислительной системы, состоящей из 16 вычислительных ядер, сгруппированных по 4 вычислительных ядра в один вычислительный модуль («процессор» ГРАВС). Исследовалось поведение вычислительной системы при отказе одного вычислительного ядра в процессоре (рис. 6). Исходя из представленных данных, глобальная распределенная ассоциативно-вычислительная среда в среднем на 12% быстрее выполняет вычисления при выходе из строя одного вычислительного ядра по сравнению со стандартной архитектурой ППВС – многомодульной вычислительной средой (ММВС). При выходе из строя большего числа вычислительных элементов эффект будет более выражен.

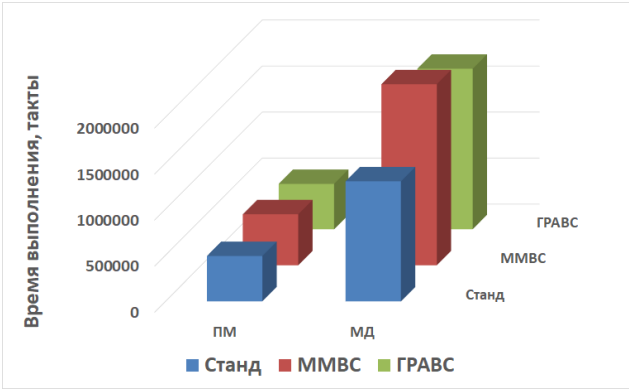

Рис. 6. Время прохождения задач при выходе из строя одного вычислительного ядра, где ПМ – задача «Перемножение матриц», размерность матрицы 64*64, «МД» - задача «Молекулярная динамика», размерность (8 кубоидов, 256 частиц, 10 итераций), «Станд.» – стандартная архитектура ППВС при работающих ВЯ, «ММВС» – многомодульная вычислительная среда при одном неработающем ВЯ, «ГРАВС» – глобальная распределенная ассоциативно-вычислительная среда при одном неработающем ВЯ

V. Заключение

Для систем с нетрадиционной архитектурой, таких как ППВС, требуются новые, отличающиеся от традиционных, подходы к обеспечению надежности.

Повысить степень надежности параллельной потоковой вычислительной системы можно используя особые свойства как потоковой модели вычислений, так и реализующей её архитектуры. Однако это требует разработки оригинальных алгоритмов сбора информации для формирования локальных контрольных точек, способов фиксации сбоев и отказов, а также введения новой аппаратуры в состав вычислительного ядра для автоматического восстановления работы после сбоя.

Динамическое перераспределение вычислений между ИУ делает возможным организацию дополнительных каналов связи между исполнительными устройствами для организации «переливов» как пакетов, так и токенов. Такое перераспределение вычислений между ИУ позволяет даже при выходе из строя отдельных каналов связи между ИУ и коммутатором продолжать функционирование этих ИУ.

Описаны подходы к решению проблемы ППВС - переполнению АПК, представлены варианты предотвращения переполнения, связанных с откачкой и подкачкой токенов в динамическом режиме работы и с разбиением задачи на этапы.

Разработан общий алгоритм работы системы при ее восстановлении после сбоя/отказа и предложены варианты аппаратной поддержки создания локальных контрольных точек. Описаны подходы к созданию ЛКТ в автоматическом режиме и восстановлению из ЛКТ.

Кратко описана глобальная распределенная ассоциативно-вычислительная среда, реализация которой дает заметные преимущества по сравнению с базовой архитектурой ППВС, прежде всего с точки зрения повышения надежности системы.

Описанные в статье основные методы и подходы к созданию средств восстановления после сбоя/отказа, позволят обеспечить высокую степень надежности аппаратно-программных средств параллельной потоковой вычислительной системы, сохраняя при этом её преимущества.

Литература

[1] Донгарра Дж. Экзафлопсное будущее суперкомпьютеров // Суперкомпьютеры. 2010. №1(1). С. 21–23.

[2] Горбунов В.В., Елизаров Г.А., Эйсымонт Л.К. Экзафлопсные суперкомпьютеры: достижения и перспективы // Открытые системы. 2013. № 7. С. 10-15.

[3] Zmeev D.N., Klimov A.V., Levchenko N.N., Okunev A.S., Stempkovskii A.L. Features of the Architecture Implementing the Dataflow Computational Model and Its Application in the Creation of Microelectronic High-Performance Computing Systems // Russian Microelectronics. 2019. Vol. 48. No. 5. P. 292-298. doi: 10.1134/S1063739719050111.

[4] Гаврилов С.В., Иванова Г.А., Рыжова Д.И., Стемпковский А.Л. Методы повышения надежности комбинационных микроэлектронных схем на основе мультиинтервального анализа быстродействия // Системы высокой доступности. 2015. Т. 11. № 4. С. 69-76.

[5] Стемпковский А.Л., Левченко Н.Н., Окунев А.С., Цветков В.В. Параллельная потоковая вычислительная система – дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. 2008. №10. С. 2-7.

[6] Левченко Н.Н., Окунев А.С., Змеев Д.Н., Климов А.В. Архитектура ассоциативной памяти ключей и методы предотвращения ее переполнения в параллельной потоковой вычислительной системе «Буран» // Вестник Рязанского государственного радиотехнического университета. 2018. № 65. С. 63-73.

[7] Ivannikov A.D., Levchenko N.N., Okunev A.S., Stempkovsky A.L., Zmejev D.N. Global Distributed Associative Environment - Evolution of Parallel Dataflow Computing System "Buran" // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2018), Kazan, Russia, Sept 14 – 17. 2018. P. 655-659.

[8] Khodosh L.S., Klimov A.V., Levchenko N.N., Okunev A.S., Zmejev D.N. Research of Asynchronous Algorithm for Molecular Dynamics Task on the PDCS "Buran" Models // In bk.: Proceedings of IEEE EAST-WEST DESIGN & TEST SYMPOSIUM (EWDTS'2016), Yerevan, Armenia, October 14-17. 2016. P. 327-330.

[9] Levchenko N.N., Okunev A.S., Zmejev D.N. Investigation of Sparse Matrix Multiplication Task for the PDCS "Buran" // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2017), Novi Sad, Serbia, Sept 29 – Oct 2. 2017. P. 174-177.

Methods and Approaches for Improving the Reliability of the Parallel Dataflow Computing System

D.N. Zmejev, N.N. Levchenko, A.S. Okunev

Institute for Design Problems in Microelectronics of RAS, Moscow, nick@ippm.ru

***Abstract* — The approaches to the creation of hardware and software tools for the restoration of dataflow computing systems operation after a fault or failure have their own specifics due to the complex structure of parallel computing processes and difficulties in localizing the source of the error.**

The article describes the methods and approaches using which the degree of reliability of the parallel dataflow computing system is increased. These methods are associated with overcoming the overflow of the content addressable memory of keys of the matching processor, local dynamic redistribution of computations between execution units, operation of recovery tools after an execution unit fault or failure, hardware support for creating local checkpoints, and also with the implementation of the global distributed associative computing environment.

The problem of overflow of the content addressable memory of keys of the matching processor critically affects the reliability of the computing system. Hardware and software methods for preventing overflow of such memory are associated with spooling/swapping tokens in dynamic mode and with dividing the task into stages.

Local dynamic redistribution of computations between execution units allows even in case of failure of individual communication channels to continue the functioning of the computational module. The options for recovering from a fault or failure of an execution unit are described.

For the operation of local recovery tools in the computational core and the computational module, hardware support for local checkpoints is proposed. The developed mechanism for the formation of local checkpoints solves the problem of system recovery after a fault in dynamic mode.

To increase the reliability of the system, it is also proposed to transit to the global distributed associative computing environment, the distribution of computations for which is carried out to a separate processor. In this case, there is a decrease in overhead costs for the restoration of its operation in the event of the failure of individual local associative elements.

The main methods and approaches proposed in the article to create tools of recovery after a fault/failure, increasing fault tolerance, architectural flexibility of the system, and improving the manufacturability of crystal manufacturing, will make it possible to create highly reliable specialized and universal supercomputers based on the parallel dataflow computing system.

***Keywords* — parallel dataflow computing system, computation reliability, dataflow computing model, local checkpoint.**

REFERENCES

[1] Dongarra Dzh. Jekzaflopsnoe budushhee superkomp'juterov (Exaflops future of supercomputers) // Superkomp'jutery. 2010. №1(1). S. 21–23.

[2] Gorbunov V.V., Elizarov G.A., Jejsymont L.K. Jekzaflopsnye superkomp'jutery: dostizhenija i perspektivy (Exaflops supercomputers: achievements and prospects) // Otkrytye sistemy. 2013. № 7. S. 10-15.

[3] Zmeev D.N., Klimov A.V., Levchenko N.N., Okunev A.S., Stempkovskii A.L. Features of the Architecture Implementing the Dataflow Computational Model and Its Application in the Creation of Microelectronic High-Performance Computing Systems // Russian Microelectronics. 2019. Vol. 48. No. 5. P. 292-298. doi: 10.1134/S1063739719050111.

[4] Gavrilov S.V., Ivanova G.A., Ryzhova D.I., Stempkovskij A.L. Metody povyshenija nadezhnosti kombinacionnyh mikrojelektronnyh shem na osnove mul'tiinterval'nogo analiza bystrodejstvija (Methods for improving the reliability of combinational microelectronic curcuits based on multiinterval timing analysis) // Sistemy vysokoj dostupnosti. 2015. T. 11. № 4. S. 69-76.

[5] Stempkovskij A.L., Levchenko N.N., Okunev A.S., Cvetkov V.V. Parallel'naja potokovaja vychislitel'naja sistema – dal'nejshee razvitie arhitektury i strukturnoj organizacii vychislitel'noj sistemy s avtomaticheskim raspredeleniem resursov (Parallel Dataflow Computing System - the Further Development of Architecture and the Structural Organization of the Computing System with Automatic Distribution of Resources) // INFORMACIONNYE TEHNOLOGII. 2008. №10. S. 2-7.

[6] Levchenko N.N., Okunev A.S., Zmeev D.N., Klimov A.V. Arhitektura associativnoj pamjati kljuchej i metody predotvrashhenija ee perepolnenija v parallel'noj potokovoj vychislitel'noj sisteme «Buran» (Building variants of content addressable memory of keys for parallel dataflow computing system «Buran») // Vestnik Rjazanskogo gosudarstvennogo radiotehnicheskogo universiteta. 2018. № 65. S. 63-73.

[7] Ivannikov A.D., Levchenko N.N., Okunev A.S., Stempkovsky A.L., Zmejev D.N. Global Distributed Associative Environment - Evolution of Parallel Dataflow Computing System "Buran" // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2018), Kazan, Russia, Sept 14 – 17. 2018. P. 655-659.

[8] Khodosh L.S., Klimov A.V., Levchenko N.N., Okunev A.S., Zmejev D.N. Research of Asynchronous Algorithm for Molecular Dynamics Task on the PDCS "Buran" Models // In bk.: Proceedings of IEEE EAST-WEST DESIGN & TEST SYMPOSIUM (EWDTS'2016), Yerevan, Armenia, October 14-17. 2016. P. 327-330.

[9] Levchenko N.N., Okunev A.S., Zmejev D.N. Investigation of Sparse Matrix Multiplication Task for the PDCS "Buran" // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2017), Novi Sad, Serbia, Sept 29 – Oct 2. 2017. P. 174-177.