ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № ?(??), 2014

УДК 004.272.3

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

Безбарьерное программирование в парадигме потоков данных на примере задачи молекулярной динамики[1]

Аннотация. Во многих HPC-задачах масштабирование сдерживается излишним применением барьеров, которые в условиях неравномерной и изменяющейся загрузки приводят к дополнительным простоям, не говоря о затратах на сам барьер. Мы предлагаем строить алгоритмы, опирающиеся исключительно на локальные попарные взаимодействия между соседями. Это трудно при традиционном программировании (SPMD, MPI, CSP), где проще прибегнуть к барьеру. Наш подход опирается на парадигму потоков данных, в которой синхронизация производится по готовности данных. Мы иллюстрируем его на примере алгоритма решения задачи молекулярной динамики. В ней обычно глобальные барьеры используются для поддержания списков соседей, обновляемых в процессе работы. Вместо барьеров нам приходится вести аккуратный подсчет находящихся поблизости частиц. В представленном алгоритме синхронизация опирается исключительно на локальные взаимодействия между вычислительными ядрами, обрабатывающими близкие области. Данное решение хорошо реализуется в модели вычислений и на архитектуре параллельной потоковой вычислительной системы “Буран”.

Ключевые слова и фразы: молекулярная динамика, масштабирование, асинхронные вычисления, барьерная синхронизация, параллельные вычисления, потоковая модель вычисления.

[1] Статья рекомендована к публикации в журнале «Программные системы: теория и приложения» Программным комитетом НСКФ-2014.

Введение

Масштабированию задач молекулярной динамики на суперкомпьютерах кластерного типа обычно мешают глобальные барьеры, используемые при пересчетах списков соседних частиц. Поэтому мы ставим задачу написания полностью асинхронного и эффективного алгоритма, опирающегося исключительно на локальные взаимодействия между вычислительными ядрами, ответственными за соседние области. Аналогичные цели ставились и другими авторами, при том, что они использовали традиционные подходы к программированию на кластерах: MPI, специальная библиотека RDMA [1], Java RMI [2].

При разработке и написании алгоритма мы используем потоковую модель вычислений, основанную на односторонних сообщениях – токенах. В ней программа задается как набор описаний (типов) узлов. Динамические экземпляры узлов активируются приходящими на них токенами, производят вычисления и посылают токены с результатами на другие экземпляры узлов.

Потоковая модель вычислений реализована в параллельной потоковой вычислительной системе «Буран» [3] и хорошо подходит для написания асинхронных алгоритмов. Операция посылки токена в ней играет такую же фундаментальную роль, как операции чтения и записи в память в обычных архитектурах.

В нашем решении одним экземпляром узла рассчитывается шаг эволюции одной частицы. Каждая частица взаимодействует с набором близких частиц, расположенных в пределах сферы с радиусом отсечения R_c. Взаимодействие между частицами реализуется через механизм посылки токенов и их взаимодействия в ассоциативной памяти по принципу: одна частица – один токен.

Распараллеливание основано на принципе пространственной декомпозиции, когда каждое вычислительное ядро обрабатывает частицы, находящиеся в его собственной области пространства – кубоиде. Каждый кубоид принимает (импортирует) частицы от соседних кубоидов, обрабатывает попарные взаимодействия и экс-

портирует вычисленную суммарную силу, действующую на каждую из принятых частиц, в ее «домашний» кубоид.

1. Алгоритм моделирования молекулярной динамики

Для моделирования молекулярной динамики используется метод «средней точки», предложенный в работе [1]. Это один из методов "нейтральной территории", который вычисляет взаимодействие между двумя частицами в кубоиде, возможно не содержащим ни одну из частиц данной пары. В методе «средней точки» таким является вычислительное ядро, кубоид которого содержит середину отрезка между этими двумя частицами. Как показано в работе [1], в этом варианте объем информации о частицах, передаваемой между кубоидами, минимален. В случае распределенной реализации это позволяет снизить нагрузку на коммуникационную среду.

В дополнение к собственным частицам, каждое вычислительное ядро/кубоид должно импортировать все те частицы из других вычислительных ядер/кубоидов, которые могут оказаться членами пары взаимодействующих в этом ядре частиц. Для этого достаточно импортировать все частицы, находящиеся на расстоянии не более чем $R_c/2$ от поверхности данного кубоида. Предполагается, что $1< R_c/\mathrm{b}<2–2\varepsilon$, где b – длина стороны кубоида и $\varepsilon>0$. Таким образом, область импорта ограничена 26-ю соседними кубоидами и изолирована от других кубоидов слоем, толщина которого не менее ε.

В каждом цикле решения задачи (одна итерация) происходит следующее:

A. В каждый кубоид копируются все частицы из его области импорта;
B. Силы взаимодействия вычисляются для каждой пары импортированных или собственных частиц и суммируются по каждой частице (внутри каждого кубоида);
C. Возвращение накопленных (парциальных) сил в «домашний» кубоид для каждой (импортированной) частицы;
D. Суммирование парциальных сил и интегрирование уравнения движения частиц.

При выходе частицы за границу кубоида должна быть организована миграция частиц – смена домашнего кубоида частицы. Мы объединяем эту фазу с фазой A: если в фазе D частица пересекает границу, тогда в следующей фазе A в ее экспортируемые копии ставится метка о принадлежности к другому кубоиду (отличному от того, из которого она фактически пришла), а ее оригинал переходит, соответственно, в этот новый кубоид. При этом должно быть гарантировано, что шаг миграции никогда не превысит ε.

Расчет сил производится с использованием потенциала Леннард-Джонса между любыми двумя частицами, удаленными друг от друга на расстояние не более чем R_c. Уравнения движения интегрируются методом Верле.

Описанный алгоритм хорошо реализуется в потоковой модели вычислений на архитектуре ППВС «Буран».

2. Модель вычисления ППВС “Буран”

В модели вычислений (рис. 1), реализуемой в ППВС «Буран», вычисления разделены на относительно небольшие фрагменты, названные программными узлами (или просто узлами), каждый из которых активируется по готовности данных. Программа, написанная на разработанном параллельном языке ППВС «Буран» - DFL, состоит из описаний таких программных узлов.

Программный узел состоит из названия, входных аргументов, контекста (представляет собой набор обозначенных целочисленных параметров, значения которых определяют состояние и положение в виртуальном адресном пространстве экземпляра конкретного узла) и списка операций. Такие экземпляры узлов называются виртуальными, поскольку о фактическом «появлении» экземпляра узла можно говорить только после прихода операнда на один из входов узла. Активируется экземпляр узла, то есть его программа запускается, когда на все входы поступают операнды. Программа узла может посылать токены с данными на другие экземпляры узлов. Стандартный токен имеет структуру, в состав которой вхо-

дит данное, имя узла, имя входа, значение контекста и ряд дополнительных атрибутов. Имя узла и значение контекста образуют ключ, который и идентифицирует конкретный экземпляр узла. Некоторые поля контекста в токене могут быть замаскированы символом «*», что указывает на то, что токен посылается на многие экземпляры узлов. Обычно токены, которые участвовали в активации программных узлов, удаляются после активации экземпляра узла, если значение кратности токенов не превышало единицы. Под кратностью понимается количество взаимодействий, в которых принимает участие токен до своего удаления.

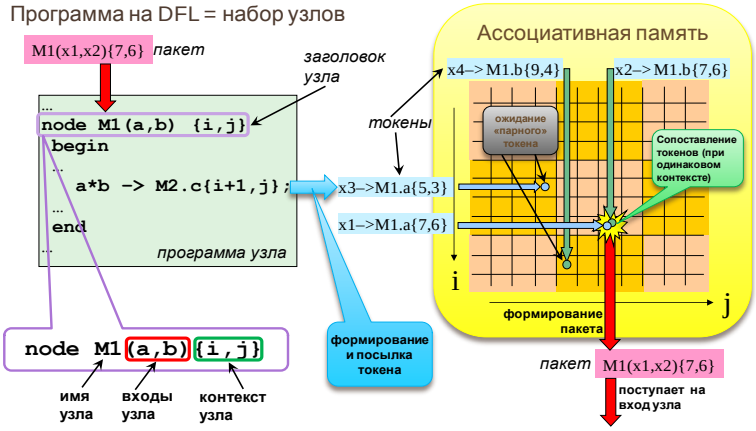


Рис. 1. Потоковая модель вычислений с динамически формируемым контекстом

Каждый виртуальный узел привязан к вычислительному ядру в соответствии со своим контекстом. Функция распределения, задаваемая отдельно от основной DFL-программы, преобразует контекст в номер вычислительного ядра.

В нашем решении задачи молекулярной динамики мы задаем трехмерную решетку кубоидов и определяем их координаты в полях контекста каждого токена или узла. Для этой задачи положительный эффект может быть получен при блочном или блочно-

циклическом распределении кубоидов. Прикладная программа вычисляет координаты кубоида для каждой частицы исходя из ее координат, а функция распределения завершает работу по распределению частиц по вычислительным ядрам.

3. Реализация задачи молекулярной динамики на ППВС «Буран»

3.1. Общее описание

Основная проблема реализации алгоритма асинхронной обработки взаимодействий состоит в определении момента, когда все нужные взаимодействия пар частиц состоялись (то есть конец фазы B), и когда все силы просуммированы (то есть конец фазы D). Ниже мы описываем некоторые особенности нашего решения.

Все частицы, которые могут взаимодействовать в данном кубоиде, направляются в него из соседних кубоидов (включая сам данный кубоид). И каждая новая частица, собственная или импортированная, прибывшая в кубоид, взаимодействует со всеми прибывшими ранее. Для каждой пары взаимодействующих частиц создается своя активация программы узла. Реализация взаимодействия частиц опирается на так называемые групповые узлы, в которых множество приходящих токенов взаимодействуют по принципу «каждый с каждым».

Чтобы определить момент, когда все необходимые частицы были импортированы и все их взаимодействия состоялись, каждый экспортирующий кубоид считает частицы, направленные каждому соседу (включая себя), и посылает ему подсчитанное количество. Также для каждой частицы подсчитывается количество импортировавших ее соседних кубоидов. Импортирующий кубоид получает 27 (по числу соседей включая себя) подсчитанных значений и использует их сумму как точное число ожидаемых частиц. Это число фактически используется в фазе B для определения готовности частичных сумм «сил взаимодействий» отдельно для каждой частицы. Таким образом, фаза C может быть инициирована для

каждой частицы независимо. Конец фазы D также определяется для каждой частицы независимо, используя подсчитанное ранее число кубоидов, в которые частица была экспортирована, и фаза A следующего цикла стартует асинхронно для каждой частицы.

Подсчет частиц опирается на механизм суммирующих входов, когда на некоторый вход узла приходит множество токенов-слагаемых и отдельным токеном на другой вход приходит количество слагаемых.

Наш подход к решению задачи молекулярной динамики использует только локальные коммуникации между соседними ядрами/кубоидами. Более того, разные частицы обрабатываются, в основном, независимо и асинхронно. Единственная синхронизация связана с подсчетом экспортированных и импортированных частиц. Поэтому можно ожидать, что данный алгоритм не имеет препятствий для неограниченного масштабирования при решении задачи МД на нашей системе.

Ниже следует подробное описание алгоритма.

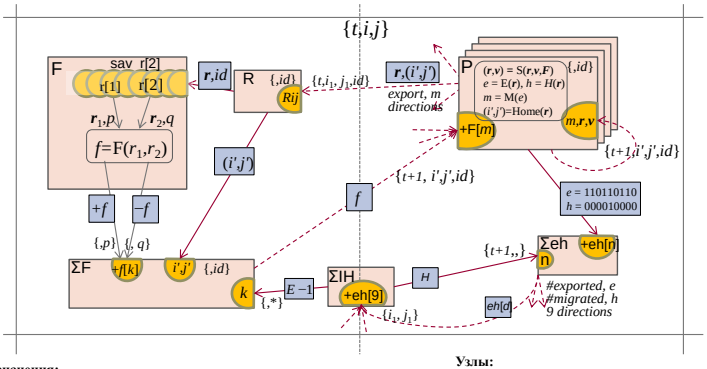
3.2. Схема алгоритма

На Рис.2 потоковый алгоритм представлен в графической форме. Показан двумерный вариант. Трехмерный вариант отличается только тем, что вместо двух координат кубоида (i,j) надо использовать три, и что число соседей (включая себя) – 27, а не 9.

Каждый прямоугольник на схеме изображает тип узла. Атрибуты (поля) контекста показаны в фигурных скобках. Чтобы не загромождать схему, общие поля контекста разных узлов $\{t,i,j\}$ вынесены наверх, а в каждом узле указываются только возможные дополнительные поля, если они есть, например $\{,id\}$. Здесь t – номер цикла, i,j - координаты кубоида, id – уникальный код частицы.

Квадратики на стрелках изображают передаваемые данные. При конце стрелок указывается контекст целевого узла. В начале – условие отправки. Усеченный диск на границе, в который идут стрелки, обозначает вход (порт). В нем стоит метка: обычно это имя порта, но также могут быть дополнительные атрибуты. Квадратные скобки указывают, что это собирающий вход, то есть на

него должно придти несколько токенов, и в скобках задано их количество. Если это не константа, а символическое имя, совпадающее с именем другого входа, то количество задается значением, поданным на этот другой вход. Если перед именем стоит знак (+), то значения, приходящие на этот вход, должны быть просуммированы и сумма станет значением входа.



Обозначения:
+n[*m*] – вход n, суммирующий *m* слагаемых (имя может быть опущено)
(*i'*,*j'*) – координаты кубоида, в котором будет главная копия частицы (home)
i_1 (j_1)– одно из: *i*−1, *i*, *i*+1 (возможные направления экспорта частицы)
m – количество направлений экспорта частицы (число единиц в строке *e*)
#exported, e – количество экспортируемых частиц (по направлениям)
#migrated, h – количество мигрирующих частиц (по направлениям)
d – одно из 9 направлений на соседа (включая направление на себя)

Узлы:
F – вычисляет силы попарных взаимодействий
ΣF – суммирует силы в рамках кубоида
P – суммирует полную силу на частицу, вычисляет ее новые координаты и скорости (***r***,***v***), посылая ее в *m* направлениях (экспорт) и на новое место (*i'*,*j'*).
Σeh - подсчет числа экспортируемых и эмигрирующих частиц в 9 соседних кубоидов
ΣIH - подсчет числа импортируемых и иммигрировавших частиц

Рис.2. Организация взаимодействий в алгоритме молекулярной динамики на ППВС.

Особый собирающий вход узла F изображен множеством дисков с пометкой вида **sav** r[2]. Это значит, что все приходящие токены удерживаются на входе r, и при этом каждые два создают активацию (пакет) с соответствующими аргументами. Каждый токен, поступающий на вход F.r, соответствует одной импортированной частице (включая собственные частицы кубоида) и содержит ее координаты и код: (***r***,*id*). Каждая активация рассчитывает силу

взаимодействия $f = F(\boldsymbol{r_1},\boldsymbol{r_2})$ для одной пары $((\boldsymbol{r_1},p), (\boldsymbol{r_2},q))$ и посылает ее с разными знаками на узлы, аккумулирующие силы для частиц p и q соответственно. Внутри функции F также проверяется условие средней точки – середина отрезка лежит в данном кубоиде – и если оно не выполнено, а также если расстояние между частицами больше R_c, то сила взаимодействия считается нулевой.

Ключевая проблема состоит в определении количества k суммируемых сил на узле ΣF. Оно всегда на 1 меньше общего количества взаимодействующих частиц на узле F (при отсутствии взаимодействия посылаются нули). Для его определения на стороне «экспортера» подсчитываем (в узле Σeh) количества частиц, экспортируемых по каждому из 9 направлений. Сам этот подсчет завершается по общему числу частиц h, находящихся сейчас в кубоиде (то есть имеющих этот кубоид в качестве «домашнего»). Эти количества посылаются вслед экспортируемым частицам, а на стороне «импортера» 9 полученных слагаемых суммируются (узлом ΣIH).

По завершении суммирования сил взаимодействия, полученные частичные силы надо послать обратно в домашний кубоид каждой из частиц и там сложить. Число слагаемых m – это число направлений, в которые эта частица была экспортирована. Оно определяется в узле P, который и силы возвращенные суммирует (входом f), и движение частицы интегрирует, и экспорт нового положения частицы производит. При экспорте подсчитывается и число направлений экспорта m для данной частицы, которое будет использовано на следующей итерации как число слагаемых сил.

Узел P также осуществляет миграцию. Если новое положение частицы выходит за границу кубоида, то она должна оказаться в другом кубоиде. Для этого по новым координатам частицы вычисляются новые координаты кубоида i',j'. Они передаются при экспорте частицы вместе с ее новым положением, чтобы импортер знал, куда передавать частичные силы обратно. Одновременно в кубоид (i',j') передаются все обновленные атрибуты данной частицы (положение $\boldsymbol{r}$, скорость $\boldsymbol{v}$ и число направлений экспорта m). Таким образом, миграция происходит как бы автоматически, без отдельной фазы миграции.

С учетом миграции в каждой итерации определяется новое количество частиц в кубоиде. Для этого при подсчете числа экспортируемых мы также подсчитываем число мигрирующих по каждому направлению частиц. По каждой частице определяется пара битовых векторов: e и h, каждый длиной 9 бит. Каждый бит отмечает наличие (1) или отсутствие (0) экспорта (e) или миграции (h) в соответствующем направлении. Число единиц вектора e образует величину m, а в векторе h всегда ровно одна единица. В узле Σeh вектора e и h суммируются так, что в каждой позиции получается количество отправленных частиц. Эти количества ($e[d]$,$h[d]$) передаются в каждом направлении d (от 1 до 9) на узел ΣIH, где они суммируются на входе +eh[9]. Далее полученные количества E и H используются в качестве числа импортированных частиц и числа собственных частиц кубоида на следующей итерации.

4. Результаты экспериментов

Были проведены экспериментальные расчеты на потактовой модели ППВС. В силу ее крайней медлительности, использовались только задачи объемом 64 частицы, 256 частиц и 1024 частицы. Две итерации последней крутились на ноутбуке с Inter® Core™ i7-3630QM 2.4 GHz более 2 часов. Но были определены времена исполнения на моделируемой ППВС в тактах: условно 1 такт = 1 ns.

Мы оценивали масштабируемость задачи в режиме умножения: с увеличением размера задачи в 4 раза во столько же раз увеличивалось число процессорных ядер. При этом всегда на одном ядре моделировался один кубоид. Между ядрами темп передачи токенов ограничен уровнем 1 токен за 20 тактов (внутри ядра – 2 такта). Моделирование производилось соответственно на 16, 64 и 256 ядрах. При этом каждое ядро считалось 8-тредовым, с 2-я наборами функциональных элементов, что позволило задействовать параллелизм работы внутри одного кубоида, при этом в кубоиде в среднем всего 4 своих частицы. В Таблице 1 приведены времена выполнения одной итерации в условных микросекундах для данных задач.

Таблица 1. Результаты моделирования.

Число ядер (кубоидов), 2D-тор	64 атома	256 атомов	1024 атома
1 (16)	57.3=16*3.6		
4x4 (16)	4.8		
8x8 (64)		6.0	
16x16 (256)			6.7

Можно видеть, что при переходе от одного ядра к многим мы теряем около 30%, что связано с появлением задержек на передачу токенов и с некоторой неравномерностью распределения частиц по кубоидам (частицы разбрасывались случайным образом). При дальнейшем увеличении размера потери еще немного растут, что связано в первую очередь с увеличением максимальной плотности частиц в кубоиде, то есть с увеличением неравномерности загрузки. Других потерь, связанных с масштабированием, не выявлено.

Заключение

Наша работа на данном этапе направлена не на получение значимого прикладного результата, а на определение возможностей и эффективности применения ППВС для задач молекулярной динамики. Уже получены обнадеживающие результаты на потактовой модели ППВС, исполняемой в одном процессоре. В дальнейшем будут проведены испытания также на эмуляторе системы с использованием кластерного суперкомпьютера.

Интересно сравнить возможности и эффективность проектируемой ППВС «Буран» как универсального программируемого супервычислителя, применяемого к задачам молекулярной динамики, с теми же свойствами машины Anton [4], разработанной специально для этих задач компанией D.E. Shaw Research. Будучи построенной из очень специализированных микросхем, эта машина показала 100-кратное превосходство против традиционных кластерных суперЭВМ. Учитывая, что Anton имеет некоторое сходство с ППВС «Буран» (например, потоковая организация вычислений и коммуникаций), мы надеемся, что ППВС «Буран» займет промежуточное положение между традиционными универсальными си-

стемами кластерного типа и узко ориентированными машинами специального назначения.

Список литературы

[1] Bowers, K. J., Chow, E., Xu, X., Dror, R.O., Eastwood, M.P., Gregersen, B.A., Klepeis, J.L., Kolossvary, I., Moraes, M.A., Sacerdoti, F.D., Salmon, J.K., Shan, Y., Shaw, D.E. *Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters.* Proceedings of the ACM/IEEE Conference on Supercomputing (SC06), Article No. 84, New York, NY: IEEE, 2006.

[2] Mederski, J., Mikulski, L., Bala, P. *Asynchronous Parallel Molecular Dynamics Simulations.* Lecture Notes in Computer Science Volume 4967, 2008, pp 439-446.

[3] 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» № 10, 2008, с. 2–7.

[4] Dror, R.O., Young, C., Shaw, D.E. *Anton, a special-purpose molecular simulation machine.* In: Encyclopedia of parallel computing, Springer, 2011, pp. 60-71.

[5] Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский. *Использование модели вычислений с управлением потоком данных и реализующей ее архитектуры для систем эксафлопсного уровня производительности* // Сборник трудов V Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем – 2012» (МЭС-2012), Москва, ИППМ РАН, 2012. С. 459-462.

[6] Д.Н. Змеев, Н.Н. Левченко, А.С. Окунев, А.В. Климов. *Способы регулирования вычислений в параллельной потоковой вычислительной системе* // Сборник трудов VI Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем – 2014» (МЭС-2014), Москва, ИППМ РАН, 2014. В печати.

Об авторах:

Дмитрий Николаевич Змеев

Научный сотрудник Института проблем проектирования в микроэлектронике РАН.

e-mail: zmeev@ippm.ru

Аркадий Валентинович Климов

Старший научный сотрудник Института проблем проектирования в микроэлектронике РАН.

e-mail: arkady.klimov@gmail.com

Николай Николаевич Левченко

Кандидат технических наук, заведующий отделом Высокопроизводительных микроэлектронных вычислительных систем Института проблем проектирования в микроэлектронике РАН.

e-mail: nick@ippm.ru

Анатолий Семенович Окунев

Кандидат технических наук, ведущий научный сотрудник Института проблем проектирования в микроэлектронике РАН.

e-mail: oku@ippm.ru

Образец ссылки на публикацию:

Д. Н. Змеев, А. В. Климов, Н. Н. Левченко, А. С. Окунев. *Безбарьерное программирование в парадигме потоков данных на примере задачи молекулярной динамики* // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. ??, № ??(??), с. ??–??.

URL: http://psta.psiras.ru/read/??

A. V. Klimov, N. N. Levchenko, A. S. Okunev, D. N. Zmejev. Programming without barriers in the dataflow paradigm by example of molecular dynamics.

ABSTRACT. The scaling of HPC is often constrained due to excessive use of global barriers, especially when the load is non-uniform and varying, to say nothing about the cost of the barrier itself. We propose to build algorithms that rely solely on local pairwise interactions between neighbors. It is rather difficult in traditional programming (SPMD, MPI, CSP), when it is easier to resort to a barrier. Our approach is based on the dataflow paradigm in which execution is controlled by data availability. We demonstrate it by the example of the algorithm for molecular dynamics simulation. It typically uses global barriers to maintain neighbor lists which must be updated during the processing. Instead of the barriers, we have to keep accurate count of nearby particles. In the presented algorithm, synchronization relies solely on local interactions between cores processing nearby areas. This solution fits well the dataflow computation model and the architecture of a dataflow computer system "Buran".

Key Words and Phrases: molecular dynamics, scaling, asynchronous computing, barrier synchronization, parallel computing, dataflow computation model.