УДК 004.4’23:004.453

ЛУЦКИЙ Г.М.,
СТИРЕНКО С.Г.,
ЗИНЕНКО А.И.,
ГРИБЕНКО Д.В.

ПРЕДСТАВЛЕНИЕ ЗАДАЧ В СИСТЕМАХ РАСПАРАЛЛЕЛИВАНИЯ С ИЗМЕНЯЕМОЙ ЗЕРНИСТОСТЬЮ

В этой статье рассматриваются подходы к представлению вычислительных задач в различных современных системах параллельного программирования, и предлагается новый подход, который позволят описывать задачу в терминах абстрактных операций без явного использования примитивов параллельного программирования. Предложенный метод позволяет построить систему времени выполнения, обеспечивающую параллельное выполнение программы пользователя с возможностью изменения зернистости в зависимости от типа используемого аппаратного обеспечения. Система может найти широкое применение в распределённых и грид вычислениях.

This article studies the approaches to description of computational tasks in different modern parallel programming systems and proposes a new approach that allows to define a task in terms of abstract operations without explicit usage of parallel programming primitives. The proposed technique allows to build a runtime system providing parallel execution of the user code that could change granularity on-the-fly with regards to the hardware being used. Such a system can be used in distributed and grid computing.

Введение

Информатика и средства вычислительной техники являются одной из наиболее быстро развивающихся областей современной науки. С момента начала активного её развития сохраняется положительная динамика, что явно следует из соблюдения таких экспоненциальных зависимостей, как например закон Мура [1] и его следствия. Это также подчёркивается большим количеством научных и государственных организаций, декларирующих развитие вычислительной техники одним из приоритетных направлений. [2, 3] Постоянный рост требований к вычислительным ресурсам для решения наиболее важных научных задач современности привёл к созданию ряда технологий, позволяющих предоставлять ресурс для их решения, в частности кластерных вычислений [4] и международной грид-инфраструктуры. [5]

Традиционным подходом к повышению производительности вычислительных комплексов являлось усовершенствование аппаратной компоненты путём увеличения тактовых частот, степени интеграции элементов, и другими подобными методами. Однако в последнее десятилетие рост производительности за счёт их применения практически прекратился [6], в связи с чем широкое использование приобрели параллельные и распределённые вычисления [7]. Несмотря на потенциально большие приросты производительности в теории, на практике применение параллельных или распределённых вычислительных систем не всегда достаточно эффективно. Показано [8], что теоретически линейный рост скорости вычисления при горизонтальном масштабировании системы не только подвержен деградации из-за возникновения накладных расходов на выполнение служебных функций системы, но и также ограничен сверху численной характеристикой применяемого в программной компоненте алгоритма в следствие закона Амдала. Из вышесказанного следует, что при решении сложных научных задач на современных компьютерных системах на скорость получения результатов влияет не только аппаратное, но и программное обеспечение, в том числе прикладное. [9]

Развитие параллельных и распределённых вычислений в аппаратной части привело также к появлению большого количества технологий программирования таких систем, библиотек для параллельных вычислений; созданию новых языков программирования [10, 11], ориентированных исключительно на параллельные вычисления или добавления их поддержки в уже существующие популярные языки. [12, 13] С другой стороны усложнение аппаратной компоненты требует от пользователя вычислительной системы более глубоких знаний принципов работы системы для достижения приемлемой производительности. Одним из факторов, наи-

более значительно влияющих на скорость вычисления, является правильное соотношение зернистости задачи с зернистостью системы. В то время как пользователь может более точно оценить возможность разбиения задачи, то учёт особенностей вычислительной системы лучше возложить на других специалистов. Таким образом, возникает необходимость в программной системе, которая предполагает описание алгоритмов вычисления, разделения на подзадачи и объединения в терминах предметной области. Это позволило бы параллельно выполнять данные алгоритмы на произвольной системе, но при этом пользователю не требуется углубляться в тонкости аппаратной организации системы. Данный подход может упростить использование гетерогенных ресурсов, в частности грид-инфраструктуры.

Применяемые подходы к описанию вычислительных задач

Одной из наиболее простых в использовании технологий, реализующих близкий к предложенному подход, является OpenMP [14, 15]. Данная технология определяет ряд директив компилятора, обрабатываемых препроцессором, для ручного указания участков исходного кода, допускающих параллельное выполнение. Поддержка спецификации OpenMP 3.0 реализована в большинстве современных компиляторов. [16, 17, 18, 19] Идеология работы с данной технологией предполагает аннотирование исходного кода программы на языках C, C++ или Fortran, в том числе и написанного ранее, так чтобы обеспечить параллельное выполнение некоторых частей программы, именуемых секциями, или итераций циклов. При проектировании данной технологии были учтены статистические данные [20] о том, что выполнение циклической обработки занимает в среднем 80% времени выполнения программы. Исходя из чего основные усилия были применены к созданию простого механизма описания возможности параллельного выполнения итераций цикла. На данный момент OpenMP позволяет аннотировать циклы произвольного уровня вложенности, указывать на данные, разделяемые всеми итерациями, и эффективным алгоритмом выполнять свёртку результатов с использованием предопределённых ассоциативных операторов. Кроме поддержки со стороны компиляторов, технология OpenMP предполагает предоставление конечному пользователю разделяемой библиотеки времени выполнения, которая обеспечивает непосредственно создание параллельных потоков выполнения, планирование и балансировку нагрузки подзадачами (секциями, итерациями) этих потоков. Она также имеет программный интерфейс (API), позволяющий пользователю частично контролировать процесс выполнения, однако это не является обязательным. Технология OpenMP рассчитана для использования в системах с общей памятью и её планировщик, использующий принцип разделения работ и единую очередь готовых к выполнению подзадач, имеет ряд режимов, связанных с этой особенностью [14, 21]. OpenMP позволяет пользователю обеспечить параллельное выполнение произвольных задач при помощи аннотирования исходного кода, преимущественно циклических частей. Наиболее важным преимуществом данной технологии является отсутствие необходимости в существенной модификации исходных кодов, что позволяет достаточно быстро изменить коды ранее написанных вычислительных библиотек. Однако, данная технология применима только в системах с общей памятью, в то время как наиболее производительными на данный момент являются системы с локальной памятью, в частности кластерные.

Существует ряд разработанных библиотек исходного кода, позволяющих решать с применением параллельных вычислений, некоторые научные задачи. К ним можно отнести используемые в Центре суперкомпьютерных вычислений НТУУ «КПИ» [22] программные пакеты Gromacs, fftw, Gamess и другие. Будучи специализированными средствами для решения конкретного класса задач, они предоставляют достаточно простой и в то же время эффективный подход к их параллельному решению, однако все они спроектированы с учётом особенностей конкретных задач, что приводит к невозможности обобщения технических решений, принятых при их разработке, на произвольные описываемые пользователем задачи.

К наиболее обобщённым из используемых пакетов программного обеспечения можно отнести комплекс Lapack [23] и библиотеку Ouverture [24]. Данные программные средства предоставляют программный интерфейс для решения задач линейной алгебры и численного решения дифференциальных уравнений в частных производных, используя особенности кластерных вычислительных систем и технологии MPI [25]. Библиотека Ouverture, в частности

содержит в себе компоненты A++ и P++ для описания работы с матрицами и тензорами в объектно-ориентированном стиле на языке C++. Компоненты имеют идентичный интерфейс, однако второй из них использует MPI для обеспечения параллельного выполнения операций.

Приведенные программные пакеты широко используются в вычислениях и обеспечивают существенное ускорение решения поставленных задач, но вместе с тем требуют от пользователя явного сведения задачи к заложенным в них абстракциям, что хотя и возможно для большого количества задач, может потенциально приводить к накладным расходам на преобразование в процессе работы программы. Кроме того данные преобразования могут потребовать значительного изменения исходного кода программ, если таковой был ранее написан. В последнее время ведущий производитель процессоров компания Intel анонсировала создание специализированных систем с общей памятью, использующих процессоры с большим количеством ядер (до 80) [26, 27], которые могут быть использованы для увеличения производительности существующих вычислительных комплексов. Аналогичный подход был применён при развитии вычислений общего назначения на видеоускорителях, например технология CUDA [28], однако в отличие от него технология Intel Manycore [29] использует традиционную x86 архитектуру процессора и набор инструкций. Последнее позволило спроектировать и разработать ряд высокоуровневых средств для решения пользовательских задач на параллельных вычислительных системах. Эти средства получили название Array Building Blocks и Thread Building Blocks [21].

Технология Array Building Blocks (ArBB) ориентирована на параллельное выполнение операций над массивами в языке C++ с использованием специализированной библиотеки и JIT-компиляции [30]. Такой подход позволяет при первоначальном компилировании преобразовывать операции над объектами ArBB в некоторую метаинформацию для библиотеки времени выполнения и JIT-компилятора ArBB, который в свою очередь непосредственно перед выполнением может сгенерировать последовательность исполняемых инструкций для конкретного типа используемого процессора, например с применением векторизации по технологиям AVX и SSE [31]. Согласно произведённым измерениям уменьшение времени вычисления подавляющего большинства задач перекрывает накладные расходы работы JIT-подсистемы технологии. ArBB предполагает работу с одно-, двух- и трёхмерными массивами с использованием как предопределённых, так и определяемых пользователем операций. Сущности библиотеки могут быть семантически представлены как векторы, матрицы и тензоры третьего порядка, в терминах которых может быть описана задача. Следовательно, данная технология с точки зрения пользователя осложняется тем же, что и ранее рассмотренный пакет Ouverture. ArBB позволяет генерировать поток векторных инструкций для систем с общей памятью, однако не поддерживает работу в системах с локальной памятью, что ограничивает её распространение. Использование векторизации позволяет избавится от чересчур мелкого уровня зернистости векторно-матричных операций, на которые ориентирована данная технология.

Более обобщённой библиотекой для работы с параллельными вычислениями от Intel является Thread Building Blocks (TBB) [32]. Данная библиотека предоставляет программный интерфейс с активным использованием шаблонов языка C++ и задействует дополнительный планировщик в разделяемой библиотеке времени выполнения для обеспечения эффективного распараллеливания. Отличительной особенностью данной библиотеки является высокий уровень абстракции: пользователь описывает в задачу в терминах некоторого набора действий, а также принципы разделения на подзадачи и объединения результатов вычислений. При этом отсутствует необходимость явного описания параллельных вычислений. Во время работы программы библиотека времени выполнения автоматически выполняет разбиение задачи на необходимое число подзадач, с учётом различных стратегий разбиения и планирования, обеспечивает параллельное вычисление результатов подзадач и их объединение в окончательное решение.

В процессе работы используется планирование по запросам (work-stealing scheduling) [33, 34] и множество очередей подзадач для каждого процессора. Такой подход достаточно эффективен для задач общего вида и особенно эффективен для задач обладающих естественным параллелизмом, т. е. независимых по данным [21].

Библиотека TBB использует при планировании вычислений особенности архитектур вы-

числительных систем с общей памятью, и, следовательно, не может быть легко адаптирована для работы в распределённых системах, однако концептуальное описание произвольных задач может быть обобщено для их поддержки. Также стратегии разделения задач в ТВВ не предполагают контроля зернистости за исключением определения минимальной неделимой подзадачи пользователем. Данная библиотека предполагает изначальное описание задачи в своих абстракциях, но вместе с тем не препятствует использованию готовых библиотек для решения подзадач [35].

Описание задач в терминах абстрактных операций

Для достижения поставленной выше цели варьирования зернистости задач в зависимости от оптимальной для используемого аппаратного обеспечения, предлагается описывать вычислительные задачи в терминах некоторых абстрактных операций над ними, так как использование примитивов параллельного программирования напрямую приводит к явному выбору зернистости задачи в процессе программирования.

Задача, вычислимая ЭВМ, может быть представлена в виде некоторой последовательности операций $\langle t_i \rangle$, выполнение которых позволяет получить требуемый результат. Многие алгоритмы не являются строго последовательными в силу того, что они могут быть разбиты на ряд независимых последовательностей операций, или представлены в виде графа задачи $G = (T_i, E)$, где узлы T_i – последовательности операций, а ребра E – зависимости между ними. При распараллеливании граф представляется в ярусно-параллельной форме, где каждый ярус L_k содержит множество независимых между собой последовательностей операций. В дальнейшем будем рассматривать множества операций в рамках одного яруса или задачу, которая может быть представлена в таком виде. Данный подход может быть применен без потери общности к произвольным представленным в виде графа задачам путём применения известных алгоритмов сопоставления графа задачи графу вычислительной системы.

Пусть задача T_0 описывается некоторым набором операцией, подлежащих выполнению. Определим оператор разбиения $split\ T \to \{T_i\}$, выполняющий разбиение задачи на подзадачи с непересекающимися, т.е. независимыми между собой, операциями. Обозначим их подзадачами первого порядка $T_1^{\langle i \rangle}$. Используя структурную рекурсию определим данный оператор таким образом, что применение его к подзадаче i-го порядка приводит к получению множества подзадач $(i+1)$-го порядка. Для сохранения структурности будем считать исходную задачу подзадачей 0-го порядка. Обозначим критерий остановки рекурсии

$$split\ T_i^{\langle a,b,\ldots,k \rangle} = \{T_{(i+1)}^{\langle a,b,\ldots,k,0 \rangle}\},$$

$$card\ \{T_{(i+1)}^{\langle a,b,\ldots,k,0 \rangle}\} = 1.$$

Разбиение подзадачи, приводящее к получению вырожденного множества (кардинальное число, соответствующее количеству элементов, равно единице) приводит к остановке рекурсии, т.к. дальнейшее разбиение невозможно.

Введём понятие пути разбиения $\langle a,b,\ldots,k \rangle$, который демонстрирует, что данная подзадача была получена из a-той подзадачи 1-го порядка, b-той подзадачи 2-го порядка и т.д.

Для непосредственного выполнения последовательности вычислительных операций введем оператор $compute\ T_i^{\langle a,b,\ldots,k \rangle} \to R_i^{\langle a,b,\ldots,k \rangle}$, выполняющий фактически преобразование из подзадачи i-го порядка, в частичный результат того же порядка. Этот оператор сохраняет значения порядка и пути разбиения.

Окончательный результат может быть получен путём комбинирования частичных результатов, для которого вводится оператор $merge\ \{R_{(i+1)}^{\langle a,b,\ldots,k,p \rangle}\} \to R_i^{\langle a,b,\ldots,k \rangle}$, выполняющий корректное слияние всех частичных результатов $(i+1)$-го порядка в частичный результат i-го порядка. Введение ограничения на слияние в терминах равенства путей разбиения $\langle a,b,\ldots,k \rangle$ и необходимости использования всех частичных результатов, позволяет сделать данный оператор неассоциативным.

Принцип построения системы варьирования зернистости

Пользователь описывает задачу, подлежащую решению, как последовательность кортежей

$$\langle (split, compute, merge, T_0, :: T, :: R) \rangle,$$

где *split* – оператор декомпозиции на подзадачи, *merge* – оператор композиции частичных результатов, *compute* – оператор вычисления, описанный так, что может быть применён к подзадаче произвольного порядка, T_0 – начальная задача, $::T$ и $::R$ – типы задачи и результата соответственно, в случае использования типизированной абстракции, в частности – языков программирования; элементы последовательности представляют собой ярусы графового представления задачи.

Непосредственно вычисление сводится к выполнению для каждого элемента последовательности следующих действий.

1. Разбиение выполняется до тех пор, пока не будет получена подзадача, удовлетворяющая требованиям относительно зернистости. Разбиение может производиться до различной глубины в случае неравномерного деления задачи. В случае же чрезмерного мелкого разделения подзадач, некоторые из них могут быть сгруппированы для выполнения на одном ресурсе.
2. Непосредственное выполнение подзадач или групп подзадач.
3. Комбинирование частичных результатов до получения результата исходной задачи. Комбинирование целесообразно выполнять на одном из ресурсов, которые использовались при вычислении для уменьшения накладных расходов на пересылки данных.

Данный подход позволяет достаточно эффективно представлять, в частности, так называемые задачи с параллелизмом по данным. Например, векторно-матричные операции, обладающие естественным параллелизмом, предполагающим мелкое зерно распараллеливания, могут использовать в качестве оператора разделения простейшее разбиение на n равных частей путём изменения индексов, а в качестве оператора слияния – объединение последовательных блоков.

Выводы

На сегодняшний день существует ряд технологий, призванных упростить программирование параллельных систем для решения научных задач путём избавления пользователя от необходимости явно учитывать особенности различных архитектур таких систем. Большинство существующих средств ориентированы на системы с общей памятью, в то время как наиболее распространёнными и наиболее производительными системами являются распределённые и системы с локальной памятью. Для последних существует ряд программных комплексов, обеспечивающих параллельное решение определённого класса задач с использованием существующих технологий.

Наиболее удобным подходом к описанию вычислительных задач является перечисление данных и действий, которые необходимо выполнить над ними, наряду с принципами разбиения их на независимые в вычислениях подзадачи и объединения частичных результатов. Такой подход может незначительно ограничить круг решаемых задач или потребовать их адаптации, однако позволяет абстрагировать архитектуру конкретной вычислительной системы и обеспечить параллельное выполнение на произвольной системе.

Эффективность таких вычислений, однако, будет существенно зависеть от оптимальности выбора зерна распараллеливания. Реализация вышеописанных принципов в виде библиотеки поддержки параллельного программирования для распределённых систем должна обязательно учитывать возможность варьирования зернистости задачи. Субоптимальные значения могут быть получены для различных классов задач на основе статистического, эволюционного или онтологического подхода.

Список литературы

1. Moore G. E. Cramming more components onto integrated circuits [Текст] // Electronics. – 1965. – Т. 38, No 8.
2. Decision no 1982/2006/ec of the European Parliament and of the Council of 18 December 2006 concerning the Seventh Framework Programme of the European Community for research, technological development and demonstration activities (2007-2013) [Text] // Official Journal of the Europian Union. – 2006. – Vol. 149. – Pp. 1–41.
3. Постанова Кабінету Міністрів України № 1020 «Про затвердження Державної цільової науково-технічної програми впровадження і застосування грід-технологій на 2009–2013 роки.» – 2009.
4. Geoffray P., Pham C., Tourancheau B. A software suite for high-performance communications on clusters of SMPs [Text] // Cluster Computing. – 2002. – Vol. 5, no. 4. – Pp. 353–363. – ISSN 1386-7857.

5. The grid: blueprint for a new computing infrastructure [Text] / Ed. by Ian Foster, Carl Kesselman. – San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. – ISBN 1-55860-475-8.
6. Sutter H. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software [Text] // Dr. Dobb's Journal. – 2005. – Vol. 30, no. 3.
7. Sutter H., Larus J. Software and the concurrency revolution [Text] // Queue. – 2005. – Vol. 3, no. 7. – Pp. 54–62. – ISSN 1542-7730.
8. Amdahl Gene M. Validity of the single processor approach to achieving large scale computing capabilities [Text] // Proceedings of the April 18-20, 1967, spring joint computer conference. – AFIPS '67 (Spring). – New York, NY, USA: ACM, 1967. – Pp. 483–485.
9. Software as a service for data scientists [Text] / Bryce Allen, John Bresnahan, Lisa Childers et al. // Commun. ACM. – 2012. – Vol. 55, no. 2. – Pp. 81–88. – ISSN 0001-0782.
10. Armstrong Joe. Programming Erlang: Software for a Concurrent World [Text]. – Pragmatic Bookshelf, 2007. – ISBN 193435600X, 9781934356005.
11. Armstrong Joe. A history of Erlang [Text] // Proceedings of the third ACM SIGPLAN conference on History of programming languages. – HOPL III. – New York, NY, USA: ACM, 2007. – Pp. 6–1–6–26.
12. International Organization for Standartization (ISO). ISO/IEC 14883, Standard for Programming Language C++: Tech. rep.: 2011.
13. Code Gen Options – Using the GNU Compiler Collection (GCC) [Electronic resource]. – Access mode: http://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc/Code-Gen-Options.html. – Last access: 15.10.2011. – Title from the screen.
14. OpenMP 3.0 Complete Specification [Electronic resource]. – Access mode: http://www.openmp.org/mp-documents/spec30.pdf. – Last access: 04.10.2010. – Title from the screen
15. Board OpenMP Architecture Review. OpenMP Application Program Interface Version 3.1 [Text]. – 2011.
16. Lattner Chris. Llvm and Clang: Advancing compiler technology [Text] // Proceedings of FOSDEM 2011: Free and Open Source Developers European Meeting. – 2011. – Pp. 234–239.
17. Lattner Chris. Llvm and Clang: Next generation compiler technology [Text] // Proceedings of BSD conference. – 2008. – Pp. 63–67.
18. Novillo D. Openmp and automatic parallelization in GCC [Text] // Proceedings of the GCC developers' summit. – 2006. – Pp. 132–141.
19. Marowka Ami. Performance of openmp benchmarks on multicore processors [Text] // Proceedings of the 8th international conference on Algorithms and Architectures for Parallel Processing. – ICA3PP '08. – Berlin, Heidelberg: Springer-Verlag, 2008. – Pp. 208–219.
20. Aiken A., Nicolau A. Optimal loop parallelization [Text] // SIGPLAN Not. – 1988. – Vol. 23, no. 7. – Pp. 308–317. – ISSN 0362-1340.
21. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, Михайленко А.В. – К., 2011. – 154 с.
22. Центр суперкомп'ютерних обчислень НТУУ «КПІ» [Електронний ресурс]. – Режим доступу: http://hpcc.org.ua/. – Останнє звернення: 15.04.2011. – Назва з екрану.
23. Anderson E., Bai Z., Bischof C. LAPACK Users' Guide [Text]. – 1992.
24. Brown David, Henshaw William, Quinlan Daniel. Overture: An object-oriented framework for solving partial differential equations [Text] // Scientific Computing in Object-Oriented Parallel Environments / Ed. by Yutaka Ishikawa, Rodney Oldehoeft, John Reynders, Marydell Tholburn. – Springer Berlin / Heidelberg, 1997. – Vol. 1343 of Lecture Notes in Computer Science. – Pp. 177–184. – ISBN 978-3-540-63827-8.
25. MPI: A message-passing interface Standard, version 2.2 [Text]. – Stuttgart, Germany: High Performance Computing Center Stuttgart (HLRS), 2009. – 648 pp.
26. Larrabee: a many-core x86 architecture for visual computing [Text] / Larry Seiler, Doug Carmean, Eric Sprangle et al. // ACM Trans. Graph. – 2008. – Vol. 27, no. 3. – Pp. 18:1–18:15. – ISSN 0730-0301.
27. Evaluation and improvements of programming models for the intel scc many-core processor [Text] / C. Clauss, S. Lankes, P. Reble, T. Bemmerl // Proceedings of International conference on High Performance Computing and Simulation (HPCS), 2011. – 2011. – Pp. 525–532.
28. Kirk D. Nvidia CUDA software and gpu parallel computing architecture: Tech. rep.: 2007.
29. Intel Manycore Testing Lab [Electronic resource]. – Access mode: http://software.intel.com/en-us/articles/intel-many-core-testing-lab/. – Last access: 15.04.2011. – Title from the screen.
30. Aycock John. A brief history of just-in-time [Text] // ACM Comput. Surv. – 2003. – Vol. 35, no. 2. – Pp. 97–113. – ISSN 0360-0300.

31. Intel SSE4 Programming Reference D91561-001 2007 [Electronic resource]. – Access mode: http://softwarecommunity.intel.com/isn/Downloads/IntelSSE4ProgrammingReference.pdf. – Last access: 08.10.2010. – Title from the screen.
32. Intel thread building blocks documentation [Electronic resource]. – Access mode: http://threadingbuildingblocks.org/documentation.php. – Last access: 08.05.2012. – Title from the screen.
33. Slaw: a scalable locality-aware adaptive work-stealing scheduler for multi-core systems [Text] / Yi Guo, Jisheng Zhao, Vincent Cave, Vivek Sarkar // SIGPLAN Not. – 2010. – jan. – Vol. 45, no. 5. – ISSN 0362-1340.
34. Blumofe Robert D., Leiserson Charles E. Scheduling multithreaded computations by work stealing [Text] // J. ACM. – 1999. – sep. – Vol. 46, no. 5. – Pp. 720–748. – ISSN 0004-5411.
35. Bws: balanced work stealing for time-sharing multicores [Text] / Xiaoning Ding, Kaibo Wang, Phillip B. Gibbons, Xiaodong Zhang // Proceedings of the 7th ACM european conference on Computer Systems. – EuroSys '12. – New York, NY, USA: ACM, 2012. – Pp. 365–378. http://doi.acm.org/10.1145/2168836.2168873.