Особенности реализации программ, использующих нерегулярные сетки, с помощью DVM-системы.

В.А. Бахтин, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, С.В. Поляков, М.Н. Притула

ИПМ им. М.В. Келдыша РАН

В данной статье рассматриваются возможности использования DVM-системы для разработки программ с нерегулярными сетками. Исследуются ограничения DVMH-модели, снижающие эффективность выполнения этих программ. Предлагаются расширения этой модели, которые должны существенно упростить разработку программ с нерегулярными сетками, способных эффективно выполняться на суперкомпьютерах различной архитектуры, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi.

Введение

Для удовлетворения желания увеличения точности вычислений, исследователям-вычислителям приходится значительно измельчать расчетную сетку. Это приводит к пропорциональному росту потребления памяти ЭВМ, равно как и увеличению времени расчетов. Частично с этой проблемой позволяет справиться переход с использования структурированных сеток на неструктурированные. В этом случае появляется возможность варьировать подробность сетки по расчетной области, тем самым сократив и время на излишне точный обсчет некоторых областей, и оперативную память, освобожденную от хранения невостребованно подробных полей величин. Также привлекательной чертой является абстрагирование численных методов от геометрии расчетной области и практическое снятие требований к ней.

Все больше программ пишется в более общем виде – для нерегулярных сеток, нежели для регулярных, с прицелом на широкое применение и переиспользование программных кодов. Однако, такие программы значительно сложнее по своей структуре. При работе с регулярными сетками отношение соседства, равно как и пространственные координаты, не приходилось хранить явно, так как эти свойства и величины напрямую связывались с многомерными индексными пространствами массивов величин. Такой подход наряду с очевидным преимуществом в виде экономии памяти, также задавал понятные правила для распараллеливания вычислений как на уровне векторизации, так и на уровне вычислительных кластеров и сетей. С одной стороны, оптимизирующие компиляторы наблюдают обращения к

элементам массивов с константными смещениями, что позволяет организовать одновременное выполнение сразу нескольких витков цикла. С другой стороны, развился подход параллелизма по данным, в котором массивы данных разрезаются на блоки, каждый блок обрабатывается отдельным процессором с помощью той же (исходной) программы, время от времени обмениваясь граничными элементами.

Модель DVMH построена на парадигме параллелизма по данным. В основе этой модели лежит понятие распределенного многомерного массива. При этом у каждого процессора имеется не только локальная часть распределенного массива, но и так называемые теневые грани – копии элементов из локальных частей соседних процессоров, через которые осуществляется основное взаимодействие процессоров. Распределение вычислений производится посредством их отображения на распределенные массивы, и, вследствие заранее известных смещений по индексам используемых массивов величин, обращения происходят либо в свою локальную часть, либо в теневые грани, определямые как продолжение локальной части по конкретному измерению распределенного массива на заранее известную ширину. Например, для шаблона типа “крест” с 4 соседями, элемент с индексами (i,j) расчитывается по элементам с индексами (i-1,j), (i,j-1),(i+1,j),(i,j+1), что выливается в необходимость иметь теневые грани ширины 1 по обоим измерениям.

1. Применение имеющихся средств для задач на нерегулярных сетках

Модель DVMH подходит в первую очередь для написания параллельных программ на регулярных прямоугольных сетках, но и некоторые виды программ на нерегулярных сетках возможно распараллелить имеющимися средствами. Рассмотрим двумерную задачу теплопроводности с постоянным, но разрывным коэффициентом в шестиграннике (Рис. 1). Область состоит из двух материалов с различными коэффициентами температуропроводности:

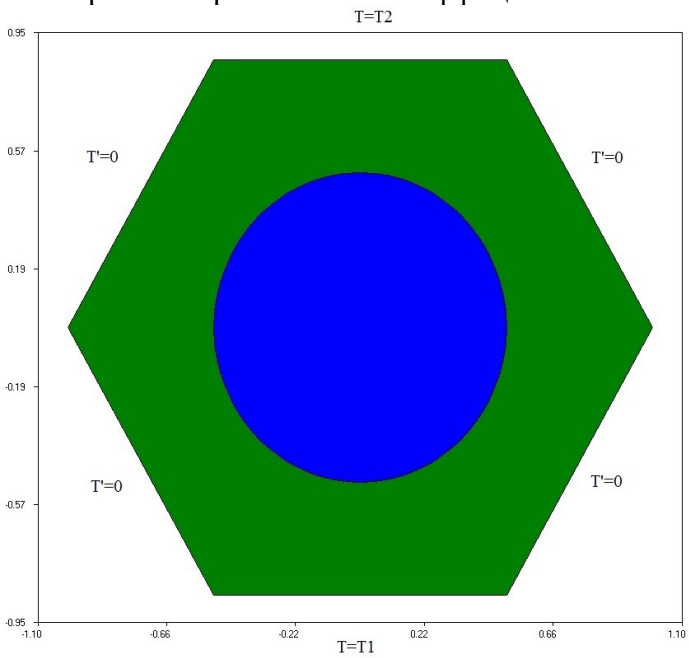

Рисунок 1. Расчетная область и граничные условия.

Рассмотрим фрагмент программы на языке Фортран, реализующий одну итерацию по времени для расчета по явной схеме (Рис. 2):

```
do i = 1, np2
  nn = ii(i)
  nb = npa(i)
  if (nb.ge.0) then
    s1 = FS(xp2(i),yp2(i),tv)
    s2 = 0d0
    do j = 1, nn
      j1 = jj(j,i)
      s2 = s2 + aa(j,i) * tt1(j1)
    enddo
    s0 = s1 + s2
    tt2(i) = tt1(i) + tau * s0
  else if (nb.eq.-1) then
    tt2(i) = vtemp1
  else if (nb.eq.-2) then
    tt2(i) = vtemp2
  endif
  s0 = (tt2(i) - tt1(i)) / tau
  gt = DMAX1(gt,DABS(s0))
enddo
do i = 1, np2
  tt1(i) = tt2(i)
enddo
```

Рисунок 2. Фрагмент Фортран-программы на нерегулярной сетке.

Как видно из этого фрагмента, массивы величин `tt1` и `tt2` являются одномерными несмотря на то, что расчетная область двумерная. Также видно, что количество соседей у каждой ячеки не является константой, а читается из массива `ii`, а сами номера соседних ячеек читаются из массива `jj`, с которым связан также массив `aa`, описывающий каждую такую связь с соседом (в данном примере – просто коэффициент при суммировании). Применить к такой программе статический подход к распределению данных и распараллеливанию не представляется возможным, т.к. связи задаются явно с помощью косвенной индексации и дополнительных массивов, значения которых известны только во время выполнения программы.

Тем не менее, такая программа может быть распараллелена с помощью DVM-системы при следующем ограничении: существует такое небольшое *M*, при котором все соседи ячейки с номером *n* имеют номера, отличающиеся от *n* не более, чем на *M*. Ключевое слово здесь *небольшое*, т.к. очевидно, что иначе в качестве такого *M* подойдет общее количество элементов сетки. Допустимая величина *M* – вещь субъективная, однако, учитывая описанный ниже способ распараллеливания, предлагается считать допустимым такое *M*, при котором *M/N* стремится к нулю при измельчении сетки (*N* – общее количество ячеек в сетке).

При распараллеливании в модели DVMH данной программы, были распределены:

- массивы искомых величин `tt1` и `tt2`;
- массивы описания элементов сетки `npa`, `xp2`, `yp2`;
- вспомогательные топологические массивы `ii` и `jj`;
- массив описания связей `aa`;

В результате, все вычисления производились только над распределенными данными с использованием только распределенных данных. Наиболее существенным при таком распараллеливании было вычисление необходимой ширины теневых граней для распределенного массива `tt1`.

Размеры теневых граней у `tt1` должны обеспечить присутствие (в локальной части или в теневой грани) на процессоре, владеющим элементом с индексом `i` также и всех элементов с индексами `jj(1:ii(i),i)`. Очевидно, что любое число *M* из приведенного выше ограничения является подходящей шириной теневой грани для массива `tt1`. Так как эта характеристика становится известной только во время выполнения, то был применен следующий подход:

1. сначала делалось блочное распределение одномерных массивов без теневых граней;
2. затем, анализируя результаты этого распределения и данные о связях, каждый процессор вычислял минимальную ширину теневой грани, покрывающую все его нужды (ссылки);
3. затем производилось объединение этих требований со взятием максимального из них.

На рисунке 3 приведен фрагмент DVMH-программы, вычисляющий ширину теневых граней для распределения массива `tt1`:

```
            allocate(npa_d(np2))
            nb1 = 1
            nb2 = np2
      ! Trying to figure out the size of the shadow edges
            shadL = 0
            shadR = 0
      !DVM$ PARALLEL(i) ON npa_d(i), NEW(nb1, nb2)
            do i = nb1, nb2
            enddo
      !DVM$    PARALLEL(i)    ON    npa_d(i),    reduction(max(shadL),
max(shadR))
            do i = 1, np2
              nn = ii(i)
              do j = 1, nn
                j1 = jj(j,i)
                if (j1.lt.nb1) then
                  shadL = max(shadL, nb1 - j1)
                elseif (j1.gt.nb2) then
                  shadR = max(shadR, j1 - nb2)
                endif
              enddo
            enddo
```

Рисунок 3. Фрагмент Fortran DVMH программы на нерегулярной сетке.

Учитывая, что теневые грани – это то, что будет пересылаться каждую итерацию по времени между процессорами, видно, что в этом подходе имеется сразу 2 места, ведущие к завышению пересылаемых объемов за счет ненужных данных.

Во-первых, ширина теневых граней в модели DVMH является универсальной величиной, не зависящей от номера процессора, т.е. нельзя одному процессору иметь ширину теневых граней 5, а другому – 10. Поэтому, чтобы удовлетворить требования всех процессоров, приходится задавать максимальную среди минимально необходимых отдельным процессорам

ширину теневых граней.

Во-вторых, в теневые грани возможно включать только непрерывные участки распределенного массива, непосредственно примыкающие к локальной части процессора и невозможно включать отдельные элементы распределенных массивов. Это ограничение заставляет иметь теневую грань достаточно широкую для того, чтобы объять все элементы массива, на которые имеются ссылки при вычислении собственных элементов.

Однако в рассматриваемой задаче размеры ячеек сетки были примерно одинаковыми, а также отсутствовали “вытянутые” элементы (с диаметром, значительно отличающимся от среднего), что позволило удачным для распараллеливания в модели DVMH способом упорядочить сеточные элементы с целью минимизации включения лишних элементов в теневые грани распределенных массивов. А именно, было применено геометрическое упорядочивание снизу вверх слева направо. Такой порядок нумерации сеточных элементов в сочетании с блочным распределением одномерных массивов величин (индексируемых номерами сеточных элементов) приводит к распределению расчетной области по процессорам слоями (горизонтальными полосами). Одну из расчетных сеток можно видеть на рис. 4, на котором изображен результат расчета (стационарное распределение температуры).

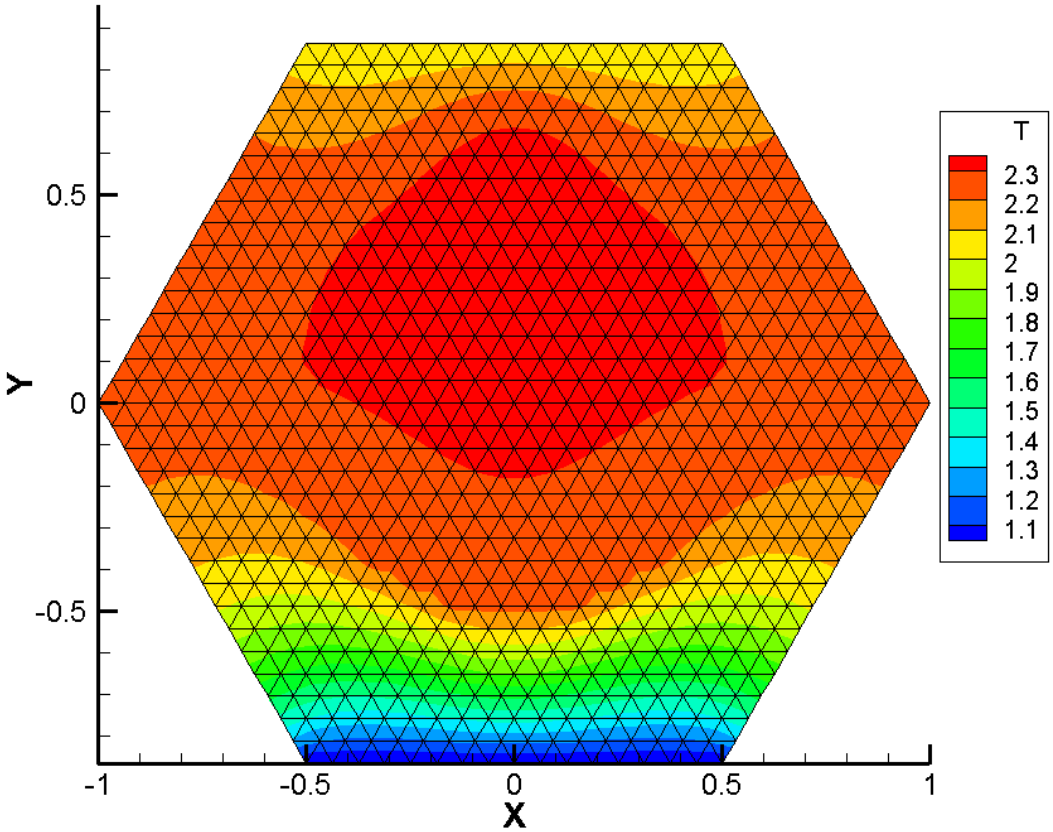

Рисунок 4. Стационарное распределение температуры.

В таблицах 1 и 2 приведены времена выполнения и достигнутые ускорения полученных DVMH-программ на кластере K-100, имеющим в каждом узле по 2 6-ти ядерных процессора Intel Xeon X5670 и 3 ГПУ Nvidia Tesla C2050. Расчеты производились на треугольной сетке из 8 миллионов узлов по явной и неявной схемам.

Таблица 1. Параллельная эффективность DVMH-программ на ЦПУ

Схема	Посл.	Параллельное выполнение, на разном количестве ядер ЦПУ													
		2		4		8		12		24		48		96	
	Время	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.
Явная	174	87,2	2	50	3,49	32	5,45	21,7	8,02	11,1	15,7	5,53	31,5	2,75	63,3
Неявная	928	728	1,27	611	1,52	449	2,07	309	3	155	6	79,6	11,7	42,8	21,7

Таблица 2. Параллельная эффективность DVMH-программ на ГПУ

Схема	Посл.	Параллельное выполнение, на разном количестве ГПУ											
		1		2		3		6		12		24	
	Время	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.
Явная	174	7	24,8	3,81	46	2,76	63	1,5	116	0,87	200	0,55	316
Неявная	928	77	12	42,3	22	29,8	31	16,3	57	9,64	96	6,55	142

2. Предложение по расширению модели DVMH для полноценной работы с нерегулярными сетками

В предыдущем пункте основной акцент был поставлен на неточный избыточный состав теневых граней распределенных массивов. Но также были упомянуты и две другие проблемы: необходимость переупорядочивания сеточных элементов (можно и в самой программе, но удобнее заранее) и ограниченность видов распределения вследствие одномерного представления массива величин (в описанном примере – только полосами).

DVMH-компиляторы преобразуют обращения к распределенным многомерным массивам к форме, независимой от размеров и положения локальной части на каждом процессоре, при этом индексные выражения остаются нетронутыми. В результате каждое обращение к распределенным данным ведется в глобальных (исходных) индексах, а при доступе к памяти применяются вычисляемые во время выполнения коэффициенты и смещения для каждого измерения. Такой подход (в отличие от изменения индексных выражений) позволяет абстрагироваться от содержания распараллеливаемых циклов, но и вводит серьезное ограничение на форму адресуемой каждым процессором части распределенного массива, называемой расширенной локальной частью, что является объединением локальной части и теневых граней. Так, в модели DVMH имеются только блочные распределения с теневыми гранями, т.е. расширенная локальная часть представляет собой подмассив исходного массива вида (A1:B1, A2:B2, A3:B3, ..., A*n*:B*n*). В качестве замечания можно отметить, что модель DVMH не дает средств описания даже циклических распределений (которые наряду с блочными популярны при распараллеливании программ на регулярных сетках), так как их поддержка потребовала бы выполнение операций деления при каждом обращении к массивам и была отвергнута в целях оптимизации.

В этом фундаментальном для внутреннего устройства и реализации компиляторов DVMH-языков и системы поддержки выполнения DVMH-программ решении и кроется источник перечисленных выше проблем и ограничений при распараллеливании программ на нерегулярных сетках. Рассмотрим один из вариантов расширения модели DVMH, который бы, с одной стороны, органично вписывался в существующую модель DVMH, дополняя ее функции, а с другой стороны позволял бы снять обсуждаемые проблемы и ограничения, причем не потеряв

значительно в эффективности параллельного выполнения.

Также ставится целью обеспечение возможности реорганизовывать данные (располагать элементы массивов в памяти в произвольном порядке) для эффективной работы на ГПУ.

2.1. Новые правила распределения

Предлагается добавить два новых правила поэлементного распределения: косвенное (indirect) и производное (derived).

Косвенное распределение задается массивом целых чисел, размер которого равен размеру косвенно распределяемого измерения, а значения задают номер домена. При этом доменов может быть как больше числа процессоров, так и меньше. DVM-система гарантирует принадлежность всех элементов одного домена одному и тому же процессору.

Производное распределение задается правилом, по форме похожим на правило выравнивания (ALIGN) модели DVMH. Однако, у него появляется значительно большая гибкость. Синтаксис можно примерно описать так:

derived-rule ::= *derived-elem-spec-list* **with** *derived-templ-spec* [**+ overlay**] [**overlay=***name*]
derived-elem-spec ::= **[** *int-range-expr* **]**
int-range-expr ::= *прозвольное целочисленное выражение + в индексных выражениях допустимы диапазоны (но не вложенные), использование align-dummy переменных (не ограниченное линейной формой) из правой части или пустые скобки. Пустые скобки - все измерение.*
derived-templ-spec ::= *var-name derived-templ-axis-spec...*
derived-templ-axis-spec ::= **[ ]** | **[@** *align-dummy* **]** | **[** *int-expr* **]**

Перед *align-dummy* добавлена собачка для возможности отличить этот случай от случая указания константного индекса. Все ссылки на распределенные массивы в *int-range-expr* обязаны быть доступны (элемент входит в расширенную локальную часть) для соответствующего элемента шаблона (перебор элементов шаблона осуществляется по его локальной части и, если указано, его **overlay**-теневой грани). В **overlay** указывается имя теневой грани, к которой будут автоматически отнесены элементы, подлежащие к распределению указанным правилом на более, чем один процессор. При этом принадлежать каждый такой элемент будет к локальной части только одного процессора (какого – не регламентируется), а на остальных он будет включен в теневую грань с указанным в **overlay** названием. Если правило распределения приводит к наложению, то указание **overlay** с именем обязательно (иначе – ошибка времени выполнения). Элементов, не распределенных ни на один процессор, быть не должно, такие случаи являются ошибкой времени выполнения и приводят к останову. Вычисленные несуществующие индексы распределяемого массива игнорируются, не приводя к ошибке.

Наложение вводится для возможности распределять связанные объекты, такие как треугольные ячейки с вершинами сетки. В таком случае появляется возможность построить одно распределение на основе другого, причем в обе стороны (в зависимости от того, что первично распределялось).

На форму результирующих локальных множеств распределенных массивов ограничений не ставится, т. е. это полноценное поэлементное распределение.

2.2. Новые теневые грани

Добавлен иной вид указания теневых граней. Место этого указания там же, где и было – в директиве описания распределенного массива. Теневая грань — это набор элементов, не принадлежащих текущему процессу (требование принадлежности соседнему процессу

снимается), для которых, во-первых, возможен доступ без специальных указаний из любой точки программы, и, во-вторых, введены специальные средства работы с ними: обновление указанием **shadow_renew**, расширение параллельного цикла указанием **shadow_compute**, огранизация выполнения циклов с зависимостями указанием **across** и т. п.

Синтаксис можно примерно описать так:

shadow-spec ::= **shadow** *subscript-range...* | **shadow (** *irreg-shadow-spec-list* **)**
subscript-range ::= **[** *int-expr* [**:** *int-expr*] **]**
irreg-shadow-spec ::= *irreg-shadow-elem-spec-list* **with** *derived-templ-spec* [*add-shadow...*] [**name =** *name*]
irreg-shadow-elem-spec ::= *irreg-shadow-axis-spec...*
irreg-shadow-axis-spec ::= **[ ]** | **[** *int-range-expr* **]**
add-shadow ::= **+** *shadow-name*

Только один из *irreg-shadow-axis-spec* может быть непустыми скобочками (одна теневая грань находится в пределах одного измерения). В целом, новый способ задания поэлементной теневой грани такой же, как и при поэлементном **derived** распределении, однако добавляется возможность построения теневой грани с использованием других теневых граней (хотя эта возможность была бы более понятна и гибка при введении исполняемой директивы добавления теневой грани вместо описательной). Из полученного множества элементов массива вычитается локальная часть (**overlay** грань не вычитается), а тому, что осталось, опционально (в отличие от **overlay** грани, по причине важности понимания ее наличия) можно дать имя, чтобы при обновлении (**shadow_renew**) можно было бы обновлять только ее (по-умолчанию обновлению подлежат все теневые грани массива). В правой и левой части допустимо ссылаться на элементы распределенного массива, теневая грань которого задается этим правилом, считая, что теневые грани к массиву добавляются вторым шагом непосредственно после его распределения. Таких теневых граней можно, в отличие от блочных, указывать несколько.

Для поэлементно-распределенных массивов теневые грани по-умолчанию (кроме **overlay**, которая появляется в результате производного распределения) отсутствуют. Добавлять теневые грани по ходу выполнения программы, возможно, будет удобным, однако на данном этапе такой функционал не закладывается.

2.3. Выравнивание, классификация распределенных массивов

Выравнивание массивов на поэлементно распределенные допускается, однако в ограниченном виде: недопустимо линейное правило выравнивания с коэффициентом, отличным от единицы на поэлементно-распределенное измерение. Таким образом, по поэлементно-распределенным измерениям возможно или размножение, или выравнивание на константу, или смещение, но невозможна «раздвижка» (коэффициент в линейном правиле по модулю больше 1) или «реверс» (коэффициент в линейном правиле отрицателен).

Стоит отметить, что массив считается блочно-распределенным, если одновременно:

1) или он сам распределен одним из старых методов, или выравнен на блочно-распределенный массив;

2) у него указанные (или не указанные вовсе) в старом виде теневые грани.

Т. е., иначе говоря, выравнивание массива на поэлементно-распределенный массив делает выравниваемый массив поэлементно-распределенным, равно как и указание теневых граней в новом виде делает его поэлементно-распределенным вне зависимости от вида примененного распределения. При этом если указанное поэлементное распределение фактически является блочным (и теневые грани тоже могут быть интерпретированы как блочные), то система поддержки оставляет за собой право считать такой массив блочно-распределенным.

2.4. Связывание поэлементно-распределенных массивов

Требования на группу связанных измерений распределенных массивов: совпадение (или смещение на константу) для распределенного измерения набора локальных для процесса глобальных индексов (а так как это требование накладывается на все процессы, то это означает одинаковое или со смещением выравнивание распределенного измерения), монотонность по включению индексов по распределенному измерению теневых граней (имена теневых граней значения не имеют, важен лишь набор элементов массива, в них входящих). Синтаксис можно примерно описать так:

#pragma dvm tie (*irreg-ref-list* **)**
irreg-ref ::= *var-name irreg-axis-ref...*
irreg-axis-ref ::= **[ ]** | **[i** [*add-op primary-expr*] **]**
add-op ::= **+** | **-**

Ровно одно из измерений каждого массива из списка должно быть с непустыми скобочками. Это исполняемая директива: система поддержки выполнения DVMH-программ пытается связать заданную группу. При этом если какие-то из указанных измерений массивов уже связаны, то производится попытка включения в их группу (а измерение массива может принадлежать только одной группе связанных измерений массивов) остальных массивов. При неудаче — ошибка времени выполнения.

Таким образом, разрешается постепенно наращивать группу связанных измерений.

Вводится и симметричная ей исполняемая директива отвязывания:

#pragma dvm untie (*irreg-ref2-list* **)**
irreg-ref2 ::= *var-name irreg-axis-ref2...*
irreg-axis-ref2 ::= **[ ]** | **[:]**

В результате ее выполнения все указанные измерения перестанут быть связанными с какими бы то ни было другими измерениями распределенных массивов.

Свойство связанности — очень важное для возможности перехода к локальной индексации массивов в местах, где важна производительность. Это свойство по сути означает согласованность локальных нумераций элементов поэлементно-распределенных массивов, включая элементы в теневых гранях. Наличие таких директив позволяет, во-первых, привести массивы в нужное состояние, т.к. даже одинаково распределенные массивы с одинаковыми по составу теневыми гранями могут быть упорядочены различным образом в памяти, а такая директива приводит их к единообразному виду. Во-вторых, процедура приведения к такому виду потенциально ресурсоемкая, а потому желательно выделить ее явно, чтобы пользователю было видно на что конкретно уходит время, и быть уверенным, что эта процедура выполняется как можно реже (в идеале — однократно).

2.5. Замена индексации на локальную в циклах

Очевидно, что при описанной свободе поэлементных распределений, процедура для перевода глобального (исходного) индекса в локальный (использующийся непосредственно для доступа к памяти) является неприемлемо вычислительно-сложной для применения при каждом обращении к массивам в высокопроизводительном коде. Поэтому в данном предложении уделено внимание способу организации обращений к поэлементно-распределенным массивам сразу по локальным индексам.

Будем считать везде далее, что локальные и глобальные индексы у всех элементов блочно-распределенного массива равны и, соответственно, эти термины взаимозаменяемы по отношению к таким массивам. Правило для доступа к памяти локальной части блочно-распределенного массива по глобальным индексам описано в начале раздела 2.

Рассмотрим сначала простой случай: цикл, в котором все поэлементно-распределенные измерения массивов индексируются только переменной цикла или выражением, эквивалентным (i + N), где i — переменная цикла, а N — целое число, одинаковое для всех витков цикла. Для этого в директиву **parallel** вводится указание **byref**.

byref означает, что параллельный цикл будет исполняться в локальных индексах. Назовем нерегулярным измерением цикла измерение параллельного цикла, отображенное (напрямую или косвенно через выравнивание) на поэлементно-распределенное измерение. Тогда параллельный цикл может быть исполнен в локальных индексах при одновременном выполнении условий:

1) Всякая переменная нерегулярного измерения цикла используется только для индексации измерений массивов, для которых выполнено свойство связанности (см. директивы **tie** и **untie**), включая то измерение массива, на которое данное измерение цикла отображено директивой **parallel**.

2) Все обращения к поэлементно-распределенному измерению массива производятся только по переменной нерегулярного измерения цикла выражением, эквивалентным (i + N), где i — переменная цикла, а N — целое число, одинаковое для всех витков цикла.

3) Все обращения к связанным поэлементно-распределенным измерениям в рамках одного витка происходят в соответствующие (связанные между собой) индексы этих измерений.

Как видно, описанный механизм не может справиться с косвенной индексацией, которая очень широко применяется в программах на нерегулярных сетках. Поэтому вводится дополнительная конструкция, призванная снять ограничение на прямую индексацию переменными цикла поэлементно-распределенных измерений массивов:

#pragma dvm reference_region *ref-spec-list*
compound-statement
ref-spec ::= **reference (** *irreg-ref-list* **<=** *origin-spec-list* **)**
irreg-ref ::= *var-name irreg-axis-ref...*
irreg-axis-ref ::= **[ ]** | **[i** [*add-op primary-expr*] **]**
origin-spec ::= *var-name* [*add-op primary-expr*] [**:** *var-name* [*add-op primary-expr*]] [**except (** *except-rel-op primary-expr* **)**]
add-op ::= **+** | **-**
except-rel-op ::= **<** | **<=** | **==** | **>=** | **>**

Конструкция говорит о том, что следует подготовить косвенную индексацию по локальным индексам для указанных в ней сочетаний. **reference** означает, что указанные в *origin-spec-list* переменные используются исключительно как ссылки на массивы из *irreg-ref-list* и свящанные с ними, а массивы из *irreg-ref-list* (и связанные с ними) индексируются либо напрямую переменными циклов либо косвенно исключительно значениями указанных в *origin-spec-list* массивов (или значениями из указанных диапазонов). Допустимо указание смещений в индексации, диапазонов и исключений. Значение в массиве индексов, подпадающее под условие исключения не подлежит переводу в локальные индексы индексируемого массива и просто игнорируется. Допускается указание различных массивов в качестве левой и правой границ диапазона, но они должны иметь одинаковый набор локально доступных элементов (локальная часть + все теневые грани), при этом их связанность не требуется. Указанные в одном списке *irreg-ref-list* измерения массивов должны быть предварительно связаны. Каждое измерение массива может быть упомянуто не более одного раза в левой части (либо быть связанным с измерениями только лишь из одного списка *irreg-ref-list*) и независимо от этого весь массив должен быть упомянут не более одного раза (диапазон с самим собой считается одним разом) в правой части. Следует отметить, что связанные измерения массивов не обязательно перечислять в левой части в полном объеме, т. е. достаточно указать только одно из связанных измерений массивов для оптимизации доступа к другим связанным с ним массивам.

Косвенная локальная индексация включается в циклах, имеющих указание **byref**, а

остальные параллельные циклы в таком регионе обрабатываются так же, как и вне его.

Конструкция статическая. Вложенность статическая и динамическая позволяется. При этом при статической вложенности указания достаточно только дополнять, а при динамической необходимо указывать все желаемые локальные ссылки повторно, т. к. действие данной директивы распространяется только на следующий за ним с синтаксической точки зрения составной оператор. При этом вступать в противоречие с динамически объемлющими регионами ссылок запрещено и будет диагностироваться, как ошибка времени выполнения.

2.6. Работа с поэлементными теневыми гранями

Во-первых, предлагается расширить указания **shadow_renew** и **shadow_compute**, добавив возможность указывать названия теневых граней (для регулярных сеток было достаточно ширины).

Для выполнения циклов с зависимостями, предлагается такое расширение указания **across**:

across (*irreg-across-spec-list* **)**
irreg-across-spec ::= *var-name* **(** [*flow-shadow-name-list*] **:** [*anti-shadow-name-list*] **)**
flow-shadow-name ::= *shadow-name*
anti-shadow-name ::= *shadow-name* [**(inout)**]

Наличие такого указания потенциально не позволяет выполненять цикл всеми процессами одновременно и независимо. Очередность должна быть непротиворечива для прямых зависимостей: сначала работают процессоры, не имеющие (т. е. имеющие их пустыми) **across**-теневых граней прямой зависимости по данным; затем те, которые зависят по этим граням только от них и так далее.

Если в графе зависимостей, наводимых записываемыми теневыми гранями есть циклы (т. е. невозможно установить порядок), то предполагается, что порядок записей в записываемые теневые грани не имеет значения (однако между чтением и записью записываемой теневой грани не должно быть ее записи другим процессором). В остальном — очередность не регламентирована. При этом на усмотрение системы поддержки выполнения программ процессоры могут выполнять части цикла параллельно, имея в виду один из допустимых порядков выполнения. Получение *flow* и *anti* с **inout** **across**-теневых граней происходит непосредственно перед выполнением части цикла процессором, таким образом получая изменения всех ранее выполнивших свои части процессоров. Те **across**-теневые грани, которые помечены как **inout**, рассылаются своим владельцам после после обработки части цикла конкретным процессором. Присутствующие при таком цикле теневые обмены (**shadow_renew**) выполняются как обычно — коллективно перед циклом, а с ними и *anti* **across**-теневые грани, не имеющие указания **inout**.

После окончания всего цикла записываемые теневые грани автоматически не обновляются. Для получения актуальных данных в них следует использовать **shadow_renew** в следующем (или где они понадобятся) цикле.

Замечание: если нужна нередукционная запись в теневые грани (пример — формирование матрицы коэффициентов связей узлов сетки (по форме — матрица смежности) путем обхода локальной части ячеек) без необходимости сохранения порядка, то указывается только *anti* зависимость с **inout**. Однако, более эффективным будет завести теневую грань ячеек и сделать цикл с **shadow_compute**, тем самым повторяя некоторые вычисления, но получая возможность независимого заполнения коэффициентов в матрице смежности.

Заключение

В процессе разработки расширения модели DVMH для решения задач, использующих нерегулярные сетки, помимо изучения мировой литературы и опыта работы с неструктурными сетками в ИПМ, были созданы экспериментальные параллельные DVM-версии программ для решения краевой задачи для двумерного квазилинейного параболического уравнения, записанного в дивергентной форме в различной постановке на неструктурированных треугольных сетках.

На ЭВМ К-100 эти параллельные версии продемонстрировали следующие ускорения по сравнению с исходными последовательными программами. Вариант с явной схемой (8 млн узлов сетки): на 12-ядерном узле - 8 раз, на 1 ГПУ 25 раз, на 24 ГПУ - 316 раз. Вариант с неявной схемой (8 млн узлов сетки): на 12-ядерном узле - 3 раза, на 1 ГПУ 22 раза, на 24 ГПУ — 142 раза.

Реализация предложенного расширения модели DVMH позволит существенно расширить класс программ, которые могут быть разработаны на языках Fortran-DVMH и C-DVMH и эффективно выполняться на суперкомпьютрах различной архитектуры, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi.

Литература

1. Андрианов А.Н., Ефимкин К.Н. Подход к параллельной реализации численных методов на неструктурированных сетках // Вычислительные методы и программирование. 2007. Т. 8. С 6-17.
2. Козубская Т.К. Разработка моделей и методов повышенной точности для численного исследования задач прикладной аэроакустики. Диссертация на соискание ученой степени доктора физико-математических наук, Москва, 2010
3. Mahesh Ravishankar, John Eisenlohr, Louis-Noel Pouchet, J. Ramanujam, Atanas Rountev, P. Sadayappan. Code Generation for Parallel Execution of a Class of Irregular Loops on Distributed Memory Systems. The Ohio State University, Louisiana State University. SC12, November 10-16, 2012.
4. Unstructured grid applications on GPU: performance analysis and improvement. Lizandro Solano-Quinde, Zhi Jian Wang, Brett Bode,Arun K. Soman. University of Illinois, Urbana, IL, Iowa State University, Ames, IA. Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, 2011.
5. В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. // Вестник Южно-Уральского университета, серия "Математическое моделирование и программирование", №18 (277), выпуск 12, 2012 г. , С. 82-92.

...раздел будет дописан позже...