УДК 519.725

С.Д. Винничук, А.Н. Давиденко, С.Я. Гильгурт, А.С. Потенко

Институт проблем моделирования в энергетике им. Г.Е. Пухова НАН Украины, Киев

ПРИМЕНЕНИЕ ГРИД-СИСТЕМЫ ПРИ ИССЛЕДОВАНИИ ЛИНЕЙНЫХ БЛОКОВЫХ КОДОВ

Для создания помехоустойчивых цифровых систем с исправлением ошибок на основе линейных блоковых кодов предлагается подход, основанный, с одной стороны, на использовании ряда способов уменьшения ресурсоемкости алгоритмов поиска порождающих матриц, с другой – на применении грид-систем в качестве высокопроизводительного вычислительного средства. Для выполнения расчетов были задействованы ресурсы Украинского национального грида. Анализ результатов позволил выявить закономерность, которая дает возможность еще больше сократить количество вычислений при дальнейшем поиске практически полезных блоковых кодов.

Ключевые слова: *линейный блоковый код, расстояние по Хеммингу, ускоренный алгоритм, грид.*

Введение

При создании помехоустойчивых цифровых систем актуальной является задача поиска наилучших кодов, контролирующих ошибки. Наиболее эффективные теоретические методы разработаны для класса линейных блоковых кодов. Однако нахождение порождающих матриц для таких кодов путем полного перебора относится к числу NP-полных задач, для решения которых требуется неприемлемо большое количество вычислительных ресурсов.

В настоящей работе предлагается подход, основанный, с одной стороны, на использовании ряда способов уменьшения ресурсоемкости алгоритмов поиска линейных блоковых кодов, с другой – на применении грид-систем в качестве высокопроизводительного вычислительного средства [1].

Алгоритм поиска линейного блокового кода

Для линейных блоковых кодов (n, k) закодированные слова получают как произведение вектора, содержащего полезные информационные биты, на порождающую матрицу G. Произвольный линейный блоковый код эквивалентен систематическому коду [2], в котором каждое кодовое слово начинается с информационных символов, а порождающая матрица систематического кода имеет вид

$$G = \left[\left[I_k \right] \left[A_{n-k} \right] \right], \qquad (1)$$

где I_k – единичная матрица порядка k, A_{n-k} – матрица, содержащая k строк и n-k столбцов.

Для кодов, исправляющих ошибки, наиболее важной характеристикой является показатель числа ошибок t, которые могут быть исправлены, который определяется через минимальное кодовое расстояние по Хеммингу

$$d^* \geq 2 \cdot t + 1, \qquad (2)$$

где d* определяется как минимальный вес w(c) (число ненулевых элементов) среди всех правильных (получаемых как произведение вектора информационных символов на порождающую матрицу) кодовых слов c, отличных от нуля

$$d^* = \min_{c \neq 0} w(c). \qquad (3)$$

Учитывая, что в случае систематических линейных блоковых кодов (n, k) все правильные кодовые слова определяются с помощью порождающей матрицы, в которой первая часть I_k является стандартной, минимальное кодовое расстояние d* оказывается функцией матрицы A_{n-k}:

$$d^* = d^*\left(A_{n-k}\right). \qquad (4)$$

Поскольку число вариантов матриц A_{n-k} конечно, может быть сформулирована задача поиска такой порождающей матрицы G (точнее, ее составной части A_{n-k}), для которой кодовое расстояние d* окажется максимальным, равным:

$$d^*_{max} = \max_{A_{n-k}} d^*(A_{n-k}). \qquad (5)$$

Поиск d^*_{max} путем полного перебора вариантов матриц A_{n-k} и всех правильных кодовых слов требует выполнения $k \cdot (n-k) \cdot n \cdot 2^k \cdot 2^{k \cdot (n-k)}$ операций, т.е. является предельно трудоемкой задачей, нереализуемой на практике. В работе [3] предложен ряд способов уменьшения вычислительной сложности задачи (5). Разработан алгоритм поиска порождающей матрицы систематического линейного блокового кода (n, k) с максимально возможным кодовым расстоянием d^*_{max} трудоемкостью $O(2^{k \cdot (n-k)}/k!)$. Предложен также его упрощенный вариант, позволяющий определять для d^*_{max} нижнюю оценку d^{**}_{max}, трудоемкость которого существенно ниже – $O(2^n)$.

Тем не менее, степенная зависимость вычислительной сложности от размерности задачи приводит к быстрому росту ресурсоемкости расчетов для значений n и k, представляющих практический интерес, что приводит к необходимости использования высокопроизводительных вычислительных средств.

Компьютерная реализация алгоритма

Первоначальная версия программы, реализующей упрощенный вариант алгоритма, предназначалась для исполнения на универсальном персональном компьютере. Данная версия позволяла вычислять порождающие матрицы для кодов размерностью не выше (42, 21).

Предварительная оптимизация использования массивов и применение класса bitset языка C++ позволили на порядок повысить быстродействие и сократить требования к оперативной памяти.

Для нахождения матриц больших размерностей дальнейшие расчеты проводились на кластере Института проблем моделирования в энергетике им. Г.Е. Пухова НАН Украины. Кластер построен по классической схеме Beowulf и включает в себя помимо управляющего три вычислительных узла, каждый из которых содержит два четырехъядерных процессора Intel Xeon и 8 Гб ОЗУ. Такая конфигурация позволяет запускать одновременно от 3 до 24 задач (экземпляров программы) в зависимости от требуемого для них объема оперативной памяти.

Для ускорения расчетов были использованы табличные вычисления, предварительно рассчитанных значений сумм по модулю 2. Уменьшить разрядность индексов без дополнительных затрат на преобразование удалось путем применения конструкции языка C++ union. Данная конструкция позволяет хранить по одному и тому же физическому адресу одновременно несколько переменных, что дает возможность обращаться к отдельным фрагментам длинного целого числа как к коротким переменным. В результате удалось увеличить быстродействие программы на несколько порядков по сравнению с первоначальной версией.

Тем не менее, при максимальных значениях n и k потребовались значительные вычислительные ресурсы. Так, поиск порождающей матрицы для кода (62, 32) занял 9944616 с (более 115 суток) процессорного времени на вышеупомянутом кластере. Для проведения дальнейших расчетов вычислительных ресурсов одного кластера оказалось недостаточно. С целью поиска порождающих матриц в области значений n = 4 ... 64, k = 2 ... 32 вычисления были перенесены в грид-среду.

Применение грид-сети

Для выполнения расчетов были задействованы ресурсы Украинского национального грида (УНГ) [4]. В качестве программного обеспечения промежуточного уровня (middleware) использовалась среда ARC (Advanced Resource Connector), разработанная европейской коллаборацией Nordugrid [5]. Данная среда в настоящее время является наиболее распространенной в украинской грид-инфраструктуре. Вычисления выполнялись на грид-ресурсах, входящих в две виртуальные организации (ВО): matmoden – "Математическое моделирование задач энергетики" на базе ИПМЭ им. Г.Е. Пухова НАН Украины и academia – ВО для обучения и тестирования на базе Национального университета «Киево-Могилянская академия».

В работе использовались следующие грид-узлы:

- arc.matmoden.kiev.ua (PIMEE ARC) – кластер Института проблем моделирования в энергетике им. Г.Е. Пухова НАНУ, г. Киев;
- arc.univ.kiev.ua – кластер Киевского национального университета имени Тараса Шевченко МОНУ, г. Киев;
- cluster.immsp.kiev.ua (IMMSP Cluster) – кластер Института проблем математических машин и систем НАНУ, г. Киев;
- ng.grid.fcss.ukma.kiev.ua (KMA Grid Cluster) – кластер Национального университета "Киево-Могилянская академия" МОНУ, г. Киев;
- ng.tntu.edu.ua (TNTU Site) – кластер Тернопольского государственного технического университета им. Ивана Пулюя МОНУ, г. Тернополь;
- nordu.hpcc.ntu-kpi.kiev.ua (KPI training cluster) – кластер Национального технического университета Украины "КПИ" МОНУ, г. Киев;
- pamela.imp.kiev.ua (IMP ARC CE) – кластер Института металлофизики им. Г.В. Курдюмова НАНУ, г. Киев;
- simulator.imath.kiev.ua (IMATH Cluster) – кластер Института математики НАНУ, г. Киев;
- uagrid.org.ua (ICYB SCIT-3) – кластер SCIT-3 Института кибернетики им. В.М. Глушкова НАНУ, г. Киев.

В пользовательский интерфейс middleware ARC, начиная с выпущенной в конце 2012 года версии 2.0.0, был внесен ряд изменений. Префикс ng- большинства команд работы в грид-среде был заменен на arc-. Тем не менее, полного соответствия команд не наблюдается. Ниже приведены примеры наиболее распространенных вариантов основных команд в новой нотации, позволяющие пользователю эффективно работать в грид-системе:

1. Генерация прокси-сертификата: **arcproxy -S matmoden**. Для описания заданий в среде ARC используется язык Extended Resource Specification Language (xRSL). В общем случае задание содержит информацию о расположении выполняемого файла задачи, аргументах командной строки, размещении входных и выходных данных, информацию о требованиях, предъявляемых к вычислительным ресурсам (максимальное время вычислений, минимальный объем памяти, тип процессора и т.п.). Пример задания (содержимого .xrsl-файла) на языке RSL:

```
&(jobname="H4824")
(executable=matr.sh)
(inputfiles=("matr.sh" "./matr.sh"))
(arguments="48" "24")
(outputfiles=("end.txt" "end.txt"))
(outputfiles=("matr.txt" "matr.txt"))
(stderr=VDHP.err)
(stdout=VDHP.out)
(GRIDTime=25000)
(disk=1)
(gmlog=gmlog)
```

2. Запуск задания в грид-среду: **arcsub MATR_48x24.xrsl**, **arcsub MATR_48x24.xrsl -d3**. В случае успешного запуска задания система сообщает пользователю присвоенный задаче идентификатор (task ID) вида https://uagrid.org.ua:60000/arex/tK7MDm 7dmTinsrz MDm6jQjGmAFKDmABFKDm3aGKDmFBFKDmaruhF o. В предыдущих версиях ARC случайная последовательность, содержащаяся в этом идентификаторе, содержала только десятичные цифры

3. Проверка статуса грид-задания: **arcstat -a, arcstat -c arc.matmoden.kiev.ua.** Возможны следующие состояния задания:

- PREPARED – подготовка к выполнению;
- INLRMS:Q – ожидание освобождения ресурса в очереди грид-узла;
- INLRMS:R – выполнение задания;
- FINISHING – завершение задания;
- FINISHED – задание успешно выполнено;
- FAILED – выполнение задание с ошибкой.

4. Получение результатов вычислений: **arcget <task ID>**. Данная команда удобна тем, что удаляет из грид-среды всю информацию о выполненном задании, в результате чего облегчается дальнейшее использование команды arcstat.

5. Прерывание выполнения задания в случае необходимости: **arckill <task ID>**.

В силу определенной специфики организация вычислительного процесса в грид-среде отличается как от вычислений на отдельной ПЭВМ или рабочей станции, так и от использования кластера. К таким особенностям можно отнести:

- создание универсального программного кода, как можно более независимого от операционной среды конкретного грид-узла, на который будет направлено задание для исполнения;
- непредсказуемость времени старта и окончания счета заданий;
- высокая вероятность аварийного прерывания завершения задания.

При проведении расчетов в процессе выполнения настоящей работы были учтены данные особенности. Так, с целью обеспечения независимости от условий работы на произвольных грид-узлах не использовалось распараллеливание с применением библиотеки MPI. Из-за временной неопределенности получения результатов усложняется организация последовательных многоэтапных расчетов, в которых отдельные подзадачи зависят по данным друг от друга. Поэтому предпочтительным, а в ряде случаев единственно возможным выходом становится разбиение общей задачи на большое число независимых подзадач, которые могут быть запущены и посчитаны автономно. В настоящем исследовании порождающие матрицы для каждой пары значений n и k рассчитывались отдельными задачами независимо друг от друга. Для снижения риска потери данных из-за прерывания задания, особенно высокого для задач, считающихся в течение несколько суток, использовались инкрементные вычисления с сохранением результатов промежуточных расчетов.

Результаты расчетов

Как показали эксперименты, ресурсоемкость задачи поиска блоковых кодов быстро увеличивается с ростом параметров k и (n - k). Но зависимости эти носят случайный и непредсказуемый характер.

По предложенному упрощенному алгоритму в данной работе были найдены порождающие матрицы и значения d^{**}_{max} максимального кодового расстояния линейных блоковых кодов (n, k) для n = 4 … 64, k = 2 … 32, полученные значения которого приведены в табл. 1. Для большей обозримости результатов здесь опущены данные для наименее интересных кодов с малыми значениями n и k. Строки соответствуют значениям k в диапазоне от 10 до 32. Столбцы соответствуют параметру, численно равному разности (n - k), также в диапазоне от 10 до 32. Этот параметр, по сути, является шириной матрицы A_{n-k} из представления (1).

Поскольку число k в плане практического применения кода определяет полезные информационные биты, а параметр (n - k) имеет смысл избыточных битов, положение найденных кодовых расстояний в таблице можно интерпретировать следующим образом. Чем левее и ниже расположена ячейка, тем меньше накладные расходы на кодирование информации, и тем больший практический интерес представляет соответствующий блоковый код. С учетом соотношения (2) приходим к заключению, что для исправления ошибок выгоднее использовать коды с нечетными значениями кодового расстояния (выделены темным цветом). Из таблицы видно, что ячейки с нечетными числами расположены в виде надломленных столбцов единичной ширины. При этом, чем ниже расположена ячейка в соответствующем столбце, тем выше эффективность соответствующего кода. В результате анализа полученных данных приходим к следующим заключениям. Значительная часть ячеек, содержащая четные числа, соответствует таким значениям n и k, которые не представляют интереса для создания помехоустойчивого кода. Следовательно, поиск матриц для этих размерностей не имеет смысла. Иными словами, представляется целесообразным искать порождающие матрицы только в конкретных областях (столбцах, содержащих нечетные числа), которые могут быть предсказаны в определенном порядке. Учет выявленной закономерности позволит еще больше сократить количество вычислений при поиске блоковых кодов.

Заключение

В работе на основе предложенного упрощенного алгоритма найдены значения нижней оценки d^{**}_{max} максимального кодового расстояния по Хеммингу d^{*}_{max} и соответствующие им порождающие матрицы линейного блокового кода (n, k).

Таблица 1

Рассчитанные значения максимального кодового расстояния

k \ n-k	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
10	6	**7**	8	8	8	8	8	8	**9**	10	10	**11**	12	12	12	12	**13**	14	14	**15**	16	16	16
11	6	**7**	8	8	8	8	8	8	**9**	10	10	**11**	12	12	12	12	12	**13**	14	14	**15**	16	16
12	6	**7**	8	8	8	8	8	8	**9**	10	10	10	**11**	12	12	12	12	12	**13**	14	**15**	16	16
13	**5**	6	6	6	**7**	8	8	8	8	**9**	10	10	**11**	12	12	12	12	12	**13**	14	14	**15**	16
14	**5**	6	6	6	6	**7**	8	8	8	**9**	10	10	10	**11**	12	12	12	12	12	**13**	14	**15**	16
15	**5**	6	6	6	6	**7**	8	8	8	8	**9**	10	10	**11**	12	12	12	12	12	**13**	14	14	**15**
16	**5**	6	6	6	6	**7**	8	8	8	8	**9**	10	10	10	**11**	12	12	12	12	12	**13**	14	15
17	**5**	6	6	6	6	6	**7**	8	8	8	**9**	10	10	10	**11**	12	12	12	12	12	**13**	14	14
18	**5**	6	6	6	6	6	**7**	8	8	8	8	**9**	10	10	**11**	12	12	12	12	12	12	**13**	14
19	**5**	6	6	6	6	6	**7**	8	8	8	8	**9**	10	10	**11**	12	12	12	12	12	12	**13**	14
20	4	**5**	6	6	6	6	**7**	8	8	8	8	**9**	10	10	**11**	12	12	12	12	12	12	**13**	14
21	4	**5**	6	6	6	6	**7**	8	8	8	8	**9**	10	10	10	**11**	12	12	12	12	12	12	**13**
22	4	**5**	6	6	6	6	**7**	8	8	8	8	8	**9**	10	10	10	**11**	12	12	12	12	12	**13**
23	4	**5**	6	6	6	6	**7**	8	8	8	8	8	**9**	10	10	10	**11**	12	12	12	12	12	12
24	4	**5**	6	6	6	6	6	**7**	8	8	8	8	**9**	10	10	10	10	**11**	12	12	12	12	12
25	4	**5**	6	6	6	6	6	**7**	8	8	8	8	**9**	10	10	10	10	**11**	12	12	12	12	12
26	4	**5**	6	6	6	6	6	**7**	8	8	8	8	8	**9**	10	10	10	10	**11**	12	12	12	12
27	4	**5**	6	6	6	6	6	**7**	8	8	8	8	8	**9**	10	10	10	10	**11**	12	12	12	12
28	4	4	**5**	6	6	6	6	**7**	8	8	8	8	8	**9**	10	10	10	10	**11**	12	12	12	12
29	4	4	**5**	6	6	6	6	**7**	8	8	8	8	8	**9**	10	10	10	10	10	**11**	12	12	12
30	4	4	**5**	6	6	6	6	**7**	8	8	8	8	8	8	**9**	10	10	10	10	**11**	12	12	12
31	4	4	**5**	6	6	6	6	**7**	8	8	8	8	8	8	**9**	10	10	10	10	**11**	12	12	12
32	4	4	**5**	6	6	6	6	**7**	8	8	8	8	8	8	**9**	10	10	10	10	10	**11**	12	12

Приведены результаты расчетов для n = 20 … 64 и k = 10 … 32. В результате анализа полученных данных выявлена закономерность, позволяющая уменьшить количество вычислительной работы при дальнейшем поиске практически полезных кодов.

Список литературы

1. Foster I. The Anatomy of the Grid: Enabling Scalable Virtual Organizations / I. Foster, C. Kesselman, S. Tuecke // International J. Supercomputer Applications, 15(3), 2001.

2. Блейхут.Р. Теория и практика кодов, контролирующих ошибки / Р. Блейхут.. — М.: Мир, 1986. – 576 с.

3. Винничук С.Д. Нижняя оценка максимального кодового расстояния для линейных блоковых кодов (n, k) над полем GF(2) / С.Д. Винничук, А.Н. Давиденко, С.Я. Гильгурт, А.С. Потенко // Тез. доп. Міжнар. НТК «Моделювання-2012». – К.: ІПМЕ ім. Г.Є.Пухова, 2012. – С. 150–153.

4. Петренко А.І. Практикум з грід-технологій: навчальний посібник / А.І. Петренко, С.Я. Свістунов, Г.Д. Кисельов – К.: НТУУ «КПІ», 2011. – 580 с.

5. Advanced Resource Connector / Nordugrid. Grid Solution for Wide Area Computing and Data Handling [Электронный ресурс] – Режим доступа: http://www.nordugrid.org /middleware. – Загл. с экрана. – 30.05.2013).

Поступила в редколлегию 25.07.2013

Рецензент: д-р техн. наук, проф. С.Е. Саух, Институт проблем моделирования в энергетике им. Г.Е. Пухова, Киев.

ЗАСТОСУВАННЯ ГРІД-СИСТЕМИ ПРИ ДОСЛІДЖЕННІ ЛІНІЙНИХ БЛОКОВИХ КОДІВ

С.Д. Винничук, А.М. Давиденко, С.Я. Гільгурт, О.С. Потенко

Для створення перешкодозахищених цифрових систем з виправленням помилок на основі лінійних блокових кодів пропонується підхід, заснований, з одного боку, на використанні низки способів зменшення ресурсоміcткості алгоритмів пошуку породжуючих матриць, з іншого – на застосуванні грід-систем у якості високопродуктивного обчислювального засобу. Для виконання розрахунків було застосовано ресурси Українського національного гріду. Аналіз результатів дозволив виявити закономірність, яка надає можливість ще більше скоротити кількість обчислень при подальшому пошуку практично корисних блокових кодів.

Ключові слова: *лінійний блоковий код, відстань по Хеммингу, прискорений алгоритм, грід.*

USE OF GRID-SYSTEM IN RESEARCHING LINEAR BLOCK CODES

S.D. Vinnichuk, A.N. Davydenko, S.Ya. Hilgurt, A.S. Potenko

In order to create noise-eliminating digital error control systems, an approach is proposed based on some methods to reduce compute intensiveness of algorithms of finding the generating matrices on the one hand and Grid-systems as a high-performance computing tool on the other hand. To fulfill computations, the resources of the Ukraine National Grid were used. The analysis of results has discovered a rule, which allows much more reducing the computation amount during further search for practically useful block codes.

Keywords*: linear block code, Hamming distance, fast algorithm, Grid.*