ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 102 за 2015 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Ефимкин К.Н.

Применение языка НОРМА для реализации некоторых тестов пакета NPB

Рекомендуемая форма библиографической ссылки: Применение языка НОРМА для реализации некоторых тестов пакета NPB / А.Н.Андрианов [и др.] // Препринты ИПМ им. М.В.Келдыша. 2015. № 102. 18 с.
URL: http://library.keldysh.ru/preprint.asp?id=2015-102

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

А.Н. Андрианов, Т.П. Баранова, А.Б. Бугеря, К.Н. Ефимкин

Применение языка НОРМА для реализации некоторых тестов пакета NPB

Москва — 2015

Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Ефимкин К.Н.

Применение языка НОРМА для реализации некоторых тестов пакета NPB

В работе рассмотрено применение языка НОРМА для реализации известных тестов из открытого пакета измерения производительности кластерных систем NAS Parallel Benchmarks (NPB). Исследована возможность реализации на языке НОРМА указанных тестов и описана процедура портирования двух тестов с языка Си на язык НОРМА. Приводится сравнение эффективности получаемых автоматически последовательных и параллельных программ по сравнению с оригинальными тестами.

Ключевые слова: суперкомпьютеры, параллельное программирование, НОРМА, NAS Parallel Benchmarks

Alexander Nikolaevich Andrianov, Tatiana Petrovna Baranova, Alexander Borisovich Bugerya, Kirill Nikolaevich Efimkin

NORMA language usage for implementing some NPB tests

The possibility to use NORMA language for implementing some well known tests from open source test suite for cluster’s performance estimation NAS Parallel Benchmarks (NPB) has been investigated. The porting procedure for two of these tests from C language into NORMA language is described. The performance of automatically generated serial and parallel programs is estimated and compared to performance of original tests.

Key words: HPC, parallel programming, NORMA language, NAS Parallel Benchmarks

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 15-01-03039-а.

Введение

В современном научном сообществе задача разработки эффективных параллельных программ имеет важное стратегическое значение. Наверное, уже не осталось ни одной области науки или отрасли промышленности, так или иначе не связанной со сферой высокопроизводительных вычислений. Несмотря на то, что параллельное программирование появилось уже достаточно давно, успешно развивается и проводится масса исследований на эту тему, вопрос «как создать эффективную параллельную программу для решения такой-то задачи» до сих пор крайне актуален для программистов и математиков.

Достаточно быстрое развитие новых аппаратных возможностей для поддержки параллельных вычислений, наблюдаемое в последнее время, еще больше усложняет проблему. Например, появление массово доступных многоядерных процессоров поставило вопрос об эффективном программировании для них. Практически одновременно появились массово доступные графические ускорители (графические процессоры), а затем – ускорители Xeon Phi. И для каждого типа ускорителей опять возникает вопрос об эффективном программировании для них. Агрессивное продвижение своих решений фирмами-производителями таких аппаратных средств часто дезориентирует прикладных специалистов, разрабатывающих параллельные вычислительные программы, толкает их на изменение средств разработки программ, хотя ясные и достаточно убедительные аргументы в пользу таких изменений отсутствуют. Так, применение технологии CUDA для эффективного программирования для графических ускорителей в первых своих версиях являлось, фактически, программированием на уровне ассемблера, с учетом тонких особенностей аппаратуры.

С помощью такого ручного низкоуровневого программирования за последние годы некоторые расчетные прикладные пакеты и математические библиотеки были портированы для использования на вычислительных системах с графическими процессорами. Это, несомненно, существенно облегчает задачу прикладному специалисту, но только в том случае, если все его потребности в высокопроизводительных вычислениях полностью покрываются уже имеющимися распараллеленными пакетами и/или библиотеками. Если же с помощью таких готовых средств построить решение для своей задачи не удаётся, то иного пути, кроме как изучить специальные инструменты программирования для целевой архитектуры и начать реализовывать свой алгоритм этими средствами на столь низком уровне, у прикладного специалиста нет.

Надежды на автоматическое распараллеливание уже написанных последовательных программ на различные виды параллельных архитектур пока совершенно не оправдываются, несмотря на то, что фирмы-производители таких аппаратных решений давно активно поддерживают данное направление исследований.

В данной работе рассматриваются возможности использования декларативного непроцедурного языка НОРМА для автоматического распараллеливания последовательной программы на примере исследования возможности портирования отдельных тестов известного тестового пакета NPB, имитирующего поведение реальных вычислительных задач, на язык НОРМА.

Язык НОРМА и система программирования НОРМА [1-3] разрабатываются в ИПМ им. М.В.Келдыша достаточно давно и предназначены для автоматизации решения вычислительных сеточных задач на параллельных компьютерах. Прикладной специалист записывает свои математические формулы на языке НОРМА, а затем компилятор языка НОРМА генерирует программу для различных параллельных архитектур и различных языков программирования (Фортран или Си). Язык НОРМА позволяет описывать решения широкого класса задач математической физики. Программа на языке НОРМА имеет очень высокий уровень абстракции и отражает метод решения, а не его реализацию при конкретных условиях. Такое описание не ориентировано на конкретную архитектуру вычислительной системы, поэтому оно предоставляет гораздо большие возможности для выявления естественного параллелизма алгоритма и организации вычислений, чем какая-либо реализация алгоритма в виде программы для какой-то конкретной вычислительной системы.

В данной работе также рассматривается вопрос оценки производительности получаемого программного кода для целевой вычислительной архитектуры. Для этого для тех тестов из пакета NPB, для которых были созданы соответствующие аналоги на языке НОРМА, было проведено сравнение эффективности программ, получающихся в результате автоматической трансляции с языка НОРМА, для различных вычислительных архитектур с результатами исполнения оригинальных тестов, написанных вручную для тех же самых вычислительных архитектур.

Пакет измерения производительности кластерных систем NPB

Пакеты измерения представляют собой специальные наборы тестов. Тесты (benchmarks) – это программа или программный комплекс, отвечающие ряду определенных требований и предназначенные для оценки тех или иных параметров системы. Тесты классифицируются на ядра (kernels) и приложения (applications benchmarks). Ядра – это фрагменты кода, взятые из реальных приложений, позволяющие измерять скорость исполнения реальной программы на разных платформах. Приложения – это программы, созданные на основе реальных приложений и адаптированные для задач тестирования. Спецификой ядер и приложений является то, что они отражают часть программы, которая занимает наибольшее время выполнения (алгоритмы численных методов,

перемножение матриц, векторов). Зная, какие задачи будут решаться на кластере, можно выбрать тот или иной тест. Наиболее широко тесты для измерения производительности кластерных систем представлены в пакете NAS Parallel Benchmarks (NPB) [4], обладающим помимо объективных достоинств еще одним необъективным - авторитетностью программ и их авторов. Это послужило выбором данного теста для исследования возможности реализации тестов на языке НОРМА и проверки эффективности кода, генерируемого компилятором с языка НОРМА.

Состав тестового пакета NPB. Тестовый пакет состоит из «простых» синтетических задач (ядер и приложений), эмулирующих вычисления на реальных задачах (в том числе в области вычислительной гидродинамики). Вычисления производятся в определенных классах задач. Под классом понимается размерность основных массивов данных, используемых в тесте: «Sample code», «Class A» (маленькие), «Class B» (большие), «Class C» (очень большие), «Class D» (огромные). В тестовом пакете существует 5 ядер: EP, MG, CG, FT, IS.

Ядро **EP** (Embarrassingly Parallel). Ядро основано на порождении пар псевдослучайных чисел (Гауссово распределение). Применяется если на кластере будут решаться задачи с использованием метода Монте-Карло.

Ядро **MG** (simple 3D Multi-Grid). Ядро использует приближенное решение трехмерного уравнения Пуассона Du=f на сетке NxNxN, где N определяется классом теста.

Ядро **CG** (Conjugate Gradient method). Метод сопряженных градиентов используется для нахождения приближенного значения наименьшего собственного числа матрицы.

Ядро **FT** (Fourier Transform). Численное решение уравнения в частных производных с использованием быстрого преобразования Фурье.

Ядро **IS** (Integer Sort). Параллельная сортировка N целых чисел.

Кроме ядер, в тестовый пакет входят 3 приложения (эмуляторы реальных программ по вычислительной гидродинамике): LU, SP, BT, алгоритмы которых используют приведенные выше ядра.

Тестовый пакет NPB имеет реализации на языках ФОРТРАН 77 и Си для следующих целевых архитектур:

- Однопроцессорный последовательный код
- Многоядерные вычислительные архитектуры, использование технологии OpenMP
- Распределённые вычислительные архитектуры, использование технологии MPI

Тестовый пакет NPB предназначен для работы в UNIX подобных операционных системах. Его особенностью является то, что для каждого класса задачи необходимо компилировать отдельную программу, которые при выполнении, независимо друг от друга, показывают производительность

различных аспектов вычислительной системы, на которой они выполняются, при решении типичных задач математической физики.

После анализа исходных текстов программ тестового пакета NPB для эксперимента по портированию данного тестового пакета на язык НОРМА были выбраны ядра CG и MG, как наиболее подходящие под концепцию языка НОРМА. И, в то же время, был сделан вывод, что такие ядра, как IS, никак не могут быть эффективно реализованы на языке НОРМА. Тесты CG и MG используют вычисления величин на сетках – именно то, для чего предназначен язык НОРМА. В них не используются такие конструкции индексов или циклы, которые не поддерживаются языком НОРМА, или видны пути альтернативных решений для таких конструкций. А такие задачи, как сортировка на языке НОРМА не решаются принципиально – в силу таких ограничений языка, как однократное присваивание, отсутствие условных операторов, операций с памятью и т.п.

Портирование CG

Тестовое ядро CG (Conjugate Gradient) осуществляет приближение к наименьшему собственному значению большой разреженной симметричной положительно определенной матрицы с использованием метода обратной итерации вместе с методом сопряженных градиентов в качестве подпрограммы для решения СЛАУ. Тест применяется для оценки скорости передачи данных при отсутствии какой-либо регулярности.

Структура теста

После визуального анализа исходного текста последовательной программы на языке Си и исследования под отладчиком скомпилированной из него исполняемой программы в тесте CG были выделены следующие основные компоненты:

1. Подготовка данных для теста. Производится инициализация вектора косвенной адресации (функция **makea**()).
2. Инициализация расчётных векторов и прогон одной итерации теста без учёта времени её выполнения, исключительно для инициализации страниц виртуальной памяти.
3. Повторная инициализация расчётных векторов.
4. Осуществление заданного количества итераций теста с замером времени (основной рабочий цикл).
5. Основная рабочая функция **conj_grad**(), осуществляющая реализацию метода сопряженных градиентов и вызываемая на каждом шаге итерации, как при инициализации, так и в основном рабочем цикле.
6. Анализ правильности полученных результатов и вывод статистических данных о прохождении теста.

После анализа производимых на каждом этапе операций было принято решение переписать на языке НОРМА функцию **conj_grad**() (этап 5) – естественно, как основную рабочую функцию теста – а также этапы инициализации (2 и 3) и основной рабочий цикл (этап 4). В то же время стало очевидно, что этап 1 – инициализация вектора косвенной адресации – плохо подходит для реализации средствами языка НОРМА. Этап 6 также было решено оставить в оригинальном виде на Си для неоспоримости факта правильности анализа полученных результатов оригинальным кодом и сохранения формата вывода. Фактически, получается что тест начинает свою работу и заканчивает её кодом на Си, а всё «сердце» теста, в котором производятся все вычисления и при выполнении которых производится засечка времени и делается вывод о производительности системы – реализуется на языке НОРМА. Поэтому в оставшемся коде на Си не нужно делать никаких предположений об архитектуре целевой системы и об используемых инструментах параллельного программирования. Этот код так и остаётся обычным последовательным кодом на Си и на быстродействие теста никак не влияет.

Таким образом, получилась следующая структура теста с использованием языка НОРМА:

- раздел **conj_grad_N**, написанный на языке НОРМА и реализующий, как и оригинальная функция **conj_grad**(), метод сопряженных градиентов;
- раздел **init**, написанный на языке НОРМА и реализующий этапы 2 и 3 теста;
- раздел **bench**, написанный на языке НОРМА и реализующий этап 4 теста;
- оригинальный код теста на языке Си, в котором код, относящийся к этапам 2 и 3, заменён на вызов **init**(), а код, относящийся к этапу 4, заменён на вызов **bench**() в обрамлении операторов засечки времени. Код функции **conj_grad**() (этап 5) убран из теста и ниоткуда не вызывается.

Вызов разделов, написанных на языке НОРМА, из оригинального кода теста на языке Си, осуществляется так же, как и любых других функций на языке Си. При этом параметры вызываемого раздела на языке НОРМА должны передаваться по адресу, если это величины на области и по значению, если это скалярные величины. Параметры, являющиеся результатами раздела, всегда должны передаваться по адресу.

Следует отметить, что раздел **conj_grad_N** вызывается только из разделов **init** и **bench**, а из кода на языке Си не вызывается. В оставленном оригинальном коде теста на языке Си не потребовалось делать никаких предположений об архитектуре вычислительного комплекса, на котором этот тест будет выполняться и об используемых средствах параллельного программирования. Это обычный последовательный Си. Вся работа по распараллеливанию

вычислений производится компилятором с языка НОРМА при трансляции разделов **conj_grad_N**, **init** и **bench** на целевую параллельную вычислительную систему.

Портирование отдельных конструкций теста

В этом разделе на примере переноса основной расчётной части теста CG мы постараемся показать, каким образом можно портировать код обычной последовательной программы в непроцедурные спецификации на языке НОРМА. Для этого поэтапно, шаг за шагом, будет приведён ход процедуры портирования с примерами конструкций исходного текста последовательной программы и соответствующими полученными конструкциями программы на языке НОРМА.

Вначале были проанализированы константные параметры, зависящие от класса задачи, и, независимо от других таких параметров, влияющие на размеры используемых в программе векторов или напрямую используемые в программе. Таких параметров в тесте CG оказалось всего 4. Они были вынесены в отдельный файл на языке НОРМА. За его малостью приведём его полностью:

```
/* CLASS = A */
DOMAIN PARAMETERS NA = 14000, NONZER = 11, NITER = 15.
CONSTANT SHIFT = 20.0.
/* CLASS = B */
//DOMAIN PARAMETERS NA = 75000, NONZER = 13, NITER = 75.
//CONSTANT SHIFT = 60.0.
/* CLASS = C */
//DOMAIN PARAMETERS NA = 150000, NONZER = 15, NITER = 75.
//CONSTANT SHIFT = 110.0.
```

Таким образом, комментируя и раскомментируя 2 строчки в отдельном файле, мы получили возможность менять класс задачи.

Все остальные параметры оказались вычисляемыми на основе 4-х вышеприведённых. Поэтому их описание было помещено в самое начало программы на языке НОРМА, в глобальной зоне видимости, и, таким образом, стало доступно всем 3-м создаваемым разделам.

Затем были проанализированы все вычислительные циклы (не итерационные – из итерационных должны получиться соответствующие итерации) в портируемых участках теста, и на основе этой информации были созданы описания областей – также в глобальной зоне видимости программы на языке НОРМА. Фактически, каждый цикл или набор вложенных циклов должен соответствовать описанию области на языке НОРМА. А сами вычислительные циклы в программе на языке НОРМА были заменены на оператор **ASSUME**. Так, например, циклу

```
for (i = 0; i < NA; i++) x[i] = 1.0;
```

стала соответствовать область **oNA** и затем оператор **ASSUME**

```
oNA:(i=1..NA).
FOR oNA ASSUME x = 1.0.
```

Также среди счётных циклов были выделены операции редукции и заменены на вызовы соответствующих функций редукции языка НОРМА. Так, например, операторы

```
rho = 0.0;
for (j = 0; j < LASTCOL - FIRSTCOL + 1; j++) {
  rho = rho + r[j]*r[j];
}
```

были трансформированы в оператор

```
rho = SUM((oLCminFCp1) r * r).
```

Итерационные циклы, которых в каждом разделе оказалось по одному, даже в разделе **init** (цикл из одной итерации для полной эмуляции рабочего цикла), были преобразованы в итерационные конструкции языка НОРМА. Были выявлены итерационные переменные – такие переменные, которые на каждом шаге итерации вычисляют свои значения с использованием значений предыдущего шага. Для примера приведём фрагмент того, как был преобразован итерационный цикл из основной рабочей функции **conj_grad**():

```
for (cgit = 1; cgit <= CGITMAX; cgit++) {
    ........................
    rho0 = rho;
    rho = 0.0;
    for (j = 0; j < LASTCOL - FIRSTCOL + 1; j++) {
      rho = rho + r[j]*r[j];
    }
    beta = rho / rho0;
}
```

В разделе **conj_grad_N** соответствующий код выглядит следующим образом:

```
ITERATION r, p, z, rho ON cgit.
    ........................
    rho = SUM((oLCminFCp1) r * r).
    beta = rho / rho[cgit-1].
    EXIT WHEN (cgit = CGITMAX).
END ITERATION cgit.
```

Там, где языка НОРМА недостаточно

Среди конструкций, которые необходимо было портировать на язык НОРМА оказались и такие, что не могут быть выражены средствами языка НОРМА. В первую очередь речь идёт о «сердце» теста - использовании косвенной адресации данных. Собственно, используется она один раз в операторе внутри итерационного цикла в основной рабочей функции **conj_grad**():

```
for (j = 0; j < LASTROW - FIRSTROW + 1; j++) {
  sum = 0.0;
  for (k = rowstr[j]; k < rowstr[j+1]; k++) {
    sum = sum + a[k]*p[colidx[k]];
  }
  q[j] = sum;
}
```

Такие конструкции как `p[colidx[k]]`, также как и циклы с переменными границами, никак не могут быть записаны на языке НОРМА. Но выход есть! Язык НОРМА поддерживает вызовы так называемых «внешних» функций, которые могут быть написаны на целевом языке программирования. Задача по реализации косвенной адресации и цикла с переменными границами возлагается на такую вспомогательную внешнюю функцию, а в программе на языке НОРМА в каждой точке области, соответствующей внешнему циклу в вышеприведённом примере, результат выполнения этой внешней функции в данной точке присваивается результирующей переменной. И тогда вся конструкция в программе на языке НОРМА принимает следующий вид:

```
PLAIN EXTERNAL FUNCTION conj_grad_HF1 DOUBLE.
FOR oLRminFRp1 ASSUME q = conj_grad_HF1(rowstr[i=i],
   rowstr[i=i+1], a ON oNZ, p[cgit-1] ON oNA, colidx ON oNZ).
```

А вспомогательная внешняя функция conj_grad_HF1() выглядит так:

```
double conj_grad_HF1(int rowstr_i, int rowstr_ip1, double *a,
    double *p, int *colidx) {
  double sum = 0.0;
  int k;
  for(k = rowstr_i; k < rowstr_ip1; k++) {
    sum = sum + a[k]*p[colidx[k]];
  }
  return sum;
}
```

Очевидно, что так функция **conj_grad_HF1**() может выглядеть только для программ, работающих с общей памятью (последовательная программа, OpenMP и CUDA). В случае использования распределённой памяти (скажем, при использовании MPI) при реализации функции **conj_grad_HF1**() встанет вопрос о получении данных из другого узла. При этом где именно будут находиться данные при написании внешней функции ещё неизвестно – эта задача возлагается на компилятор. Планируется, что в системе поддержки выполнения программ на языке НОРМА (NORMA Run Time Library) для распределённых систем будут созданы соответствующие функции, с помощью которых можно будет запрашивать данные из других узлов по имени и набору индексов. С помощью таких функций реализация функции **conj_grad_HF1**() для распределённых систем будет не сильно сложнее, чем приведённая выше для систем с общей памятью.

Также в виде внешней функции **printZeta**() была оформлена печать промежуточных результатов на каждом шаге итерации основного рабочего цикла. Сделано это было исключительно для сохранения оригинального формата вывода.

Таким образом, в результате проведённых преобразований, структура теста CG с использованием языка НОРМА получилась следующая (рис. 1):

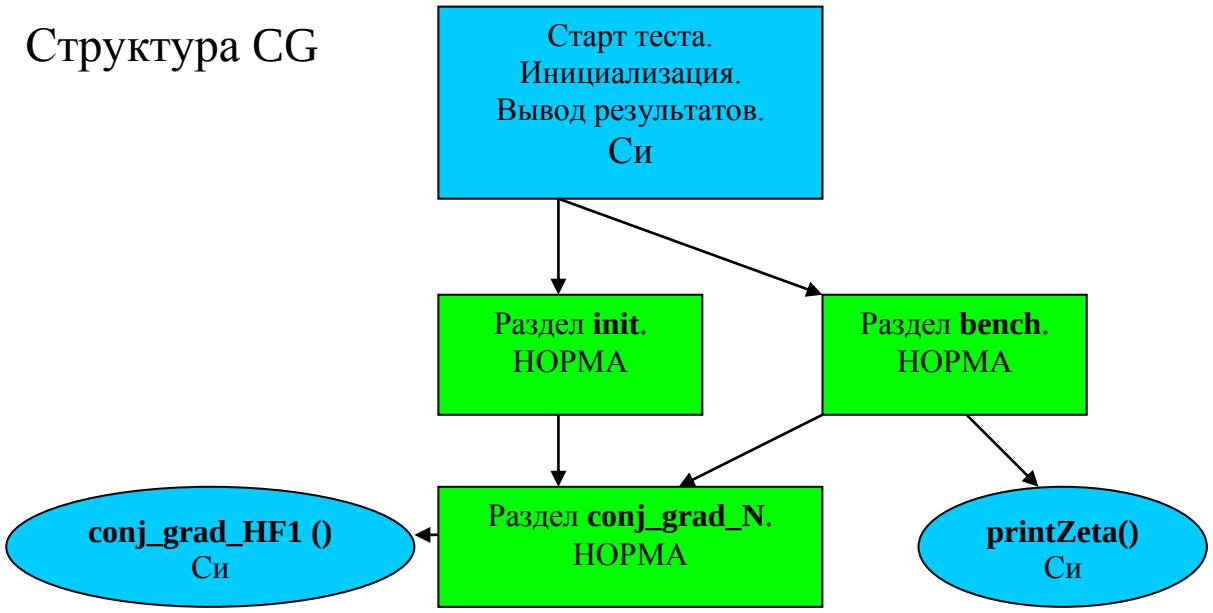

Рис. 1. Структура теста CG с использованием языка НОРМА.

Результаты выполнения теста CG с использованием языка НОРМА

На текущий момент компилятор программ с языка НОРМА поддерживает следующие виды выходных программ:

- последовательная программа на языках Фортран или Си;
- программа для вычислительных систем с общей памятью с использованием технологии OpenMP на языках Фортран или Си;
- программа для вычислительных систем с графическим процессором с использованием технологии CUDA.

После того, как портирование теста CG было завершено, настало время проверить корректность получающихся выходных программ на различных архитектурах и сравнить их эффективность с соответствующими оригинальными вариантами теста, написанными полностью вручную на целевом языке. Задача проверки корректности решалась автоматически благодаря тому, что тест CG после выполнения расчётных задач проверяет полученный результат. Процедура проверки полученного результата была оставлена в оригинальной программе на языке Си, без изменений. Таким образом, заключение этой процедуры о корректности полученного результата

можно считать за признание всей полученной исполняемой программы корректной в целом.

Для оценки эффективности получающихся выходных программ сравнивался показатель скорости выполнения теста программами, полученными с помощью компилятора с языка НОРМА, с соответствующими оригинальными программами, написанными полностью вручную на целевом языке. Показатель скорости выполнения теста рассчитывается в конце теста в миллионах операций в секунду и выводится на печать в итоговых результатах.

Запуск программ теста CG, как оригинальных, так и полученных в результате компиляции программ с использованием языка НОРМА производился для последовательной версии и для версии OpenMP на персональном компьютере с процессором Intel Core i7-3770 (4 ядра, 8 потоков, тактовая частота 3.4 GHz) под управлением операционной системы Windows 7 x64 SP1. Компиляция исходных текстов программ осуществлялась в среде Visual Studio 2008 в 32-х битном режиме. В таблице 1 приведены результаты запуска тестов классов A, B и C. Во всех случаях проверка результата прошла успешно и приведённые цифры – количество миллионов операций в секунду (Mop/s), основной показатель скорости выполнения теста.

Таблица 1

Результаты запуска теста CG (Mop/s)

	Class A		Class B		Class C	
	Си	Си + НОРМА	Си	Си + НОРМА	Си	Си + НОРМА
Последовательная программа	1648	1710	784	773	723	711
OpenMP программа	3177	2540	2523	2585	2502	2452

Как можно заметить, результаты выполнения теста CG с использованием языка НОРМА практически идентичны (видимо, с точностью до погрешности измерений) соответствующим результатам оригинального теста CG, написанного вручную. Это даёт нам право утверждать, что тест CG прекрасно подошёл для записи его на языке НОРМА. И что задачи, имитирующиеся этим тестом, также должны хорошо подходить для программирования их (или по крайней мере их основных вычислительных ядер) на языке НОРМА. Ничуть не проиграв по скорости выполнения получившихся программ, компилятор программ с языка НОРМА полностью взял на себя задачу по автоматическому распараллеливанию выходной программы, и успешно её выполнил.

Существенно более быстрое выполнения задач класса A по сравнению с классами B и C объясняется, видимо, малым размером задач класса A и, как следствие, гораздо лучшим попаданием в процессорный кэш. Этим же можно

объяснить некую разницу в этом же классе задач между оригинальным тестом и тестом с использованием языка НОРМА.

Портирование MG

Ядро использует приближенное решение трехмерного уравнения Пуассона Du=f на сетке NxNxN, где N определяется классом теста.

Структура теста

Текст последовательной программы теста на языке Си состоит основной компоненты (main) и ряда вызываемых функций.

В основной компоненте производится:

1. Подготовка данных для теста (очистка временных счетчиков, установка признака подсчета времени для расчетных функций, установка границ размерностей массивов), инициализация необходимых для расчета в определенном классе коэффициентов и прогон одной итерации теста. Распечатывается время выполнения блока инициализации.
2. Выполнение заданного количества итераций ядра теста с замером общего времени (основной рабочий цикл). В каждой итерации производится последовательный вызов основных функций с различными значениями аргументов. Как правило, аргументами являются адреса массивов и их размерности.
3. Анализ полученных результатов и вывод статистических данных о прохождении теста.

После визуального анализа текста функций и производимых в них операций были выявлены те функции, в которых присутствовали вычислительные операции и циклы, в которых отсутствовали зависимости. В результате было принято решение переписать на языке НОРМА следующие функции: **setup()**, **psinv()**, **resid()**, **comm3()**, **norm2u3()**, **zero3()**. Остальные функции были оставлены в оригинальном виде на языке Си.

Основной рабочий цикл – итерационный вызов функций – на язык НОРМА не переводился.

Таким образом, структура теста не изменилась, за исключением того, что ряд функций оказался написанным на другом языке и работа сводилась к соблюдению интерфейса между частями программы, написанными на разных языках.

Портирование отдельных функций теста

Для каждой портируемой функции были проанализированы параметры, зависящие от класса задачи, и влияющие на размеры используемых в программе массивов.

Были проанализированы вычислительные циклы и на основе этой информации были созданы статические описания областей для расчета, определены переменные на этих областях и формулы для расчета. Каждому циклу (или набору вложенных циклов) должно соответствовать описание области на языке НОРМА. Сами вычислительные циклы в программе на языке НОРМА заменяются оператором **ASSUME**.

Итерационные циклы были преобразованы в итерационные конструкции языка НОРМА. Были выявлены итерационные переменные – такие переменные, которые на каждом шаге итерации вычисляют свои значения с использованием значений предыдущего шага.

Заметим, что в исходной программе при вызове функций в качестве аргументов передаются адреса и размерности массивов, что позволяет одной функции работать с массивами различной величины.

В тесте для разных классов используются следующие размерности массивов:

CLASS A	CLASS B	CLASS C	CLASS D
256x256x256	256x256x256	512x512x512	1024x1024x1023

Поскольку язык НОРМА допускает только статические области определения переменных, то один и тот же алгоритм пришлось повторить на языке НОРМА столько раз, сколько размерностей (и, соответственно, количество статических областей) использовалось в исходном тексте. Так, например, для исходной функции **resid()** используются в вычислениях величины размерностей 4, 6, 10, 18, 32, 66, 130, 258, 514 и, соответственно, в программе на языке НОРМА было создано 9 функций**: residn4**, **residn6**, **residn10**, **residn18**, **residn32**, **residn66**, **residn130**, **residn258**, **residn514**.

Аналогично и для других функций было создано по 9 копий в программе на языке НОРМА.

Приведем пример перевода цикла из функции **resid**() (где n1, n2, n3 могут принимать значения 4 и 6):

```
for (i3 = 1; i3 < n3-1; i3++) {
 for (i2 = 1; i2 < n2-1; i2++) {
   for (i1 = 0; i1 < n1; i1++) {
      u1[i1] = u[i3][i2-1][i1] + u[i3][i2+1][i1] +
               u[i3-1][i2][i1] + u[i3+1][i2][i1];
   }
 }
}
```

Для этого цикла была определена область и оператор **ASSUME** на этой области:

```
oi25j25k:(oi25: i=2..5; oj25: j=2..5; ok6: k=1..6).
FOR oi25j25k ASSUME u1=u[i,j-1,k]+u[i,j+1,k]+u[i-1,j,k]+
                      u[i+1,j,k];
```

В тех циклах, где были выявлены операции редукции, эти операции были заменены на вызовы соответствующих функций редукции языка НОРМА. Так, например, операторы исходного текста

```
for (i3 = 1; i3 < n3-1; i3++) {
    for (i2 = 1; i2 < n2-1; i2++) {
      for (i1 = 1; i1 < n1-1; i1++) {
        s = s + pow(r[i3][i2][i1], 2.0);
        a = abs(r[i3][i2][i1]);
      }
    }
}
```

были преобразованы в операторы на языке НОРМА

```
s = SUM((oi33j33k33) r*r ).
a = MAX((oi33j33k33) abs(r)).
```

Таким образом, основные расчётные функции теста были переведены на язык НОРМА. Но вызываются они по-прежнему из программы на языке Си, что часто оказывается полезно, когда нет необходимости переносить на язык НОРМА всю исходную программу. Кроме того, такой «гибридный» подход позволяет обходить специфические ограничения языка НОРМА, такие как однократное присваивание, вид индексных выражений и способы описания индексных пространств, и т.п., что важно для расширения области практической применимости языка НОРМА.

Те функции исходного теста, которые были переписаны, были скомпилированы с помощью компилятора НОРМА в функции на языке Си для последовательной модели и для моделей параллельного программирования OpenMP и CUDA. Соответственно, в исходном коде теста на языке Си были вставлены анализ величины размерности массива и вызовы соответствующих функций, полученных компилятором НОРМА.

Результаты выполнения теста

С помощью компилятора gcc (для последовательной модели и модели OpenMP) и nvcc (для модели CUDA) были созданы варианты теста для классов B и C. На этих вариантах проводились замеры времени исполнения.

Запуск всех программ производился на вычислительном кластере К100 [5] с процессором 2 x Intel Xeon X5670, с 3 графическими ускорителями nVidia Fermi C2050.

Компиляция исходных текстов программ осуществлялась в среде UNIX.

В таблице 2 приведены результаты запуска тестов, время выполнения в секундах (sec).

Таблица 2

Результаты запуска теста MG (sec)

	Class B		Class C	
	Си	Си + НОРМА	Си	Си + НОРМА
Последовательная программа	8.62	14.56	74.02	120.60
OpenMP программа	1.7	6.19	13.57	46.87
CUDA программа		36.41		

Как видно из приведенной таблицы, характеристики работы теста с функциями, созданными компилятором НОРМА, хуже, чем в первоисточнике. Это объясняется тем, что в компиляторе с языка НОРМА реализованы пока только общие схемы отображения исходной программы в выходную, не учитывающая оптимизации, связанные, в частности, с особенностями конкретных параллельных архитектур и «тонкими» методами программирования для них. Отсюда не всегда эффективный выходной код, появление дополнительных массивов и затратная работа с памятью, увеличение количества операторов цикла. Работа по включению в компилятор возможностей такой оптимизации в настоящее время ведется.

Кроме того, известно, что производительность программ, созданных «ручным» программированием для целевой платформы, дает, как правило, лучшие результаты, чем при автоматической генерации кода.

У языка НОРМА есть свои достоинства: он позволяет снять с программиста, который не использует терминов параллельного программирования, работу, связанную с решением рутинных и довольно тонких проблем по синтезу параллельной программы, и возложить ее на компилятор. Наиболее важным, по-видимому, является возможность автоматического переноса одной и той же программы на языке НОРМА на различные параллельные архитектуры.

Заключение

Приведенные выше примеры переноса на язык НОРМА тестов известного пакета NPB для измерения производительности кластерных систем показывает ряд новых возможностей применения языка НОРМА для решения прикладных вычислительных задач. Приведенные способы переноса уже существующих последовательных программ на язык НОРМА, в том числе способы создания “гибридных” программ – объединения кода на языке НОРМА с кодом на последовательном языке (например, Си) – важны для расширения области практической применимости языка НОРМА.

Приведенные результаты выполнения портированных на язык НОРМА тестов NPB показывают корректность алгоритмов автоматического синтеза компилятором параллельных программ для различных вычислительных архитектур. Созданные и реализованные методы трансляции можно рассматривать как еще одно практическое подтверждение жизнеспособности подхода к автоматическому синтезу параллельных программ по непроцедурной (декларативной) программе на языке НОРМА [6].

Отсутствие в настоящее время в компиляторе с языка НОРМА достаточно «сильных» оптимизаций, учитывающих особенности вычислительной архитектуры и особенностей программирования для нее, (что особенно заметно при отображении в CUDA) препятствует, в ряде случаев, получению эффективного исполняемого кода. Работа над включением в компилятор таких оптимизаций в настоящее время ведется.

Также в настоящее время ведётся разработка новых возможностей языка НОРМА и началась работа по созданию компилятора с языка НОРМА для гибридных архитектур (MPI + OpenMP + CUDA).

Литература

1. Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н., Задыхайло И.Б. НОРМА. Описание языка. Рабочий стандарт. — М.: Препринт ИПМ им.М.В.Келдыша РАН. — 1995. — № 120. — 52с.
2. Андрианов А.Н., Бугеря А.Б., Гладкова Е.Н., Ефимкин К.Н., Колударов П.И. Простые вещи. — Суперкомпьютеры, 2014, № 2(18) — Москва: Изд-во СКР-Медиа , 2014. — с. 58-61.
3. Система НОРМА. URL: http://www.keldysh.ru/pages/norma
4. NAS Parallel Benchmarks, URL: http://www.nas.nasa.gov/publications/npb.html
5. Гибридный вычислительный кластер К-100. URL: http://www.kiam.ru/MVS/resourses/k100.html
6. Андрианов А.Н. Система Норма. Разработка, реализация и использование для решения задач математической физики на параллельных ЭВМ. М., автореф. дис. докт. физ.-мат. наук, 2001

Оглавление