Ганкевич И. Г.
Санкт-Петербургский государственный университет

Эффективность MIC архитектуры в задаче решения стохастических дифференциальных уравнений

Рекомендовано к публикации профессором Дегтяревым А. Б.

Введение. Стохастические дифференциальные уравнения представляют собой дифференциальные уравнения в частных производных, в которых один или более членов, а также решение описывают стохастические процессы, и одним из таких уравнений является уравнение Блека — Шоулза. Среди методов численного решения стохастических дифференциальных уравнений наиболее простым считается метод Монте-Карло, в котором в случае уравнения Блека — Шоулза с помощью псевдослучайных чисел моделируется процесс изменения цен на опции. Данная модель была выбрана в качестве примера реальной задачи, а метод решения был выбран как самый простой в реализации на параллельных архитектурах компьютеров. Описанию методики преобразования исходного кода программы и обсуждению результатов тестов производительности и посвящена данная статья.

Методика преобразования и тестирования кода программы для MIC архитектуры. Методика тестирования программы включала в себя компиляцию и запуск программы под x86 и MIC архитектуры, которые проводились с помощью специализированных инструментальных средств, и состояла из трех этапов. Первый этап состоял в отладке последовательной версии кода и его оптимизации под x86 архитектуру. Компилятор Intel использовался для тестирования различных вариантов опций, а инструмент Intel Vtune Amplifier — для измерения производительности отдельных частей исходного кода программы. Целью второго этапа был анализ зависимостей по данным и последующее распараллеливание и векторизация кода, обрабатывающего независимые структуры данных, и цель была достигнута путем использования библиотеки Intel OpenMP, «нотаций массивов» (Intel Cilk Array Notation), а также специфических директив компилятора Intel. Наконец, третий этап тестирования состоял в оптимизации программы под MIC архитектуру и сравнении

производительности с x86 архитектурой. В целом, исходная программа была оптимизирована под архитектуру MIC путем выбора определенных опций компиляции, путем векторизации исходного кода, и производительность измерялась с помощью стандартных инструментов на каждом этапе тестирования.

Этап I. Первый этап преобразования незначительно повлиял на производительность программы, однако он создал основу для оптимизаций на следующих этапах. Во-первых, все файлы, содержащие исходных код, были объединены в один файл для включения межпроцедурной оптимизации (опция `-ip`), и опции `-Wall -Wextra -std=c99` были использованы для исключения отмеченных компилятором подозрительных участков кода и для его преобразования в соответствие со стандартом C99. Далее, опции `-strict-ansi -fno-alias` и ключевое слово restrict были использованы для исключения пересечения массивов в памяти устройства. Также подпрограмма `posix_memalign` и внутренние директивы компилятора `__assume_aligned` и `_assume` были использованы для выравнивания массивов по границам в 16 байт (64 байта для MIC архитектур) для исключения неэффективных операций доступа к памяти [**?**]. Несмотря на проделанные оптимизации, опции отчетов `-vec-report -opt-report-phase=hlo -opt-report 3` показали, что они имеют незначительный эффект на результирующий исполняемый файл, и не все циклы были векторизованы в автоматическом режиме. Таким образом, получившийся код требовал более детального исследования и ручной оптимизации, что и стало целью второго этапа тестирования.

Этап II. Детальное исследование исходного кода показало, что большая часть процессорного времени тратится на выполнение одной функции в цикле, и эта функция была модифицирована для увеличения производительности. Анализ «горячих точек» в Intel Vtune Amplifier XE определил, что большая часть времени тратится на исполнение функции `current_solution`, в которой компилятор не векторизовал код. Ручной анализ кода показал, что между итерациями внешнего цикла функции нет зависимости по данным, и этот цикл был распараллелен с помощью директивы OpenMP `#pragma omp for schedule(dynamic)`; динамическое планирование итераций цикла было выбрано после тестирования производительности. Такое преобразование кода сократило время выполнения про-

граммы почти в два раза на двухпроцессорном компьютере (см. таблицу 1). Далее в коде были определены дополнительные массивы, необходимые для векторизации тела внешнего цикла, и это преобразование сократило время выполнения еще примерно в два раза при длине вектора, равной четырем. Для получения максимальной производительности были проведены эксперименты с разными инструментами векторизации, такими как Intel Cilk array notation [**?**], `#pragma simd` и автоматической векторизацией компилятором, что позволило получить четырехкратное ускорение по сравнению с последовательной версией программы. После проведения описанных оптимизаций анализ параллелизма в Intel Vtune Amplifier показал, что только 0,057 секунд тратится на синхронизацию независимых потоков исполнения, что свидетельствует о хорошем уровне оптимизации программы. В целом, с помощью распараллеливания на уровне потоков и последующей векторизации удалось получить четырехкратное ускорение на двухъядерном x86 процессоре.

Таблица 1. Спецификация программно-аппаратной вычислительной платформы, использованной на первом и втором этапе тестирования

Операционная система	CentOS release 5.8 (Final)
Компилятор	Intel(R) C Intel(R) 64 Compiler XE 12.1.1.256 Build 20111011
Вычислитель	2x Intel(R) Xeon(R) CPU X5670 @ 2.93ГГц (2 ядра, виртуальная машина VMware)

Этап III. В заключительной фазе тестирования оптимизированный код программы был перенесен на платформу с ускорителем Xeon Phi (таблица 2) и настроен под архитектуру MIC, однако многоядерный процессор показал большую производительность, чем ускоритель MIC. Во-первых, последовательная и параллельная версии программы были перезапущены на новой платформе (таблица 2), и показали ускорение в 23,62 раза на 24 ядрах для сетки `TIME_GRID=64`. Далее, путем использования опции `-mmic` и команды `micnativeloadex` [**?**] исходный код был скомпилирован под архитектуру ускорителя и запущен на нем, показав ускорение в 87,82 раза на 60 ядрах для сетки `TIME_GRID=64`. Затем параллельная версия программы были запущены еще раз с увеличенным размером сетки `TIME_GRID=256` и завершились через 247 и 903 секунд на x86 и MIC вычислителях. Таким образом, переход от x86 к MIC архитектуре увеличил ускорение относительно последовательной версии

программы ввиду большего размера вектора (16 против 4 элементов), но 24 ядра архитектуры x86 показали меньшее общее время выполнения по сравнению с 60 ядрами MIC (таблица 3).

Таблица 2. Спецификация программно-аппаратной вычислительной платформы, использованной на третьем этапе тестирования

Операционная система	Red Hat Enterprise Linux Server release 6.1 (Santiago)
Компилятор	Intel(R) C Intel(R) 64 Compiler XE 13.0.0.079 Build 20120731
Вычислитель	2x Intel(R) Xeon(R) CPU E5-2640 0 @ 2.50ГГц (24 ядра) Intel XEON PHI-5110P @ 1.053ГГц (60 ядер)

Результаты. Целью тестирования было сравнение производительности вычислителей MIC с традиционными x86 компьютерами на примере легко распаралелливаемых и векторизуемых расчетов Монте-Карло. Несмотря на проделанные автоматические и ручные оптимизации, проводившиеся под контролем анализатора производительности, общее время выполнения программы на ускорителе больше, чем на x86 компьютере. Таким образом, использование ускорителя MIC в качестве замены основному процессору x86 неэффективно, а эффективность его использования в качестве сопроцессора требует дополнительных исследований.

Таблица 3. Сравнение производительности и ускорения последовательной и параллельной версий программы на архитектурах MIC и x86

Арх.	Сетка	Ядра	Время, с.	Ускорение	Эффективность, %
x86	64	1	189,06	1	100
x86	64	24	8	23,62	98
x86	256	1	6160,52	1	100
x86	256	24	247,18	24,92	104
MIC	64	1	2206,48	1	100
MIC	64	60	25,12	87,82	146
MIC	256	1	75248,89	1	100
MIC	256	60	903,18	83,32	139

Литература

1. Doerner Wendy. Advanced Optimizations for Intel MIC Architecture. `http://software.intel.com/en-us/articles/advanced-optimizations-for-intel-mic-architecture`

2. Green Ronald W. Vectorization Essentials. http://software.intel.com/en-us/articles/vectorization-essential

3. Intel Xeon Phi Coprocessor System Software Developers Guide. http://software.intel.com/sites/default/files/article/334766/intel-xeon-phi-systemsoftwaredevelopersguide.pdf