Матричные вычисления с использованием pyMIC

К. О. Петрищев, e-mail: vrn.kostyan.p@mail.ru
А. В. Романов, e-mail: romanov_av@cs.vsu.ru

Федеральное государственное бюджетное образовательное учреждение высшего образования «Воронежский государственный университет»

Аннотация. *В работе рассматриваться использование инструмента PyMIC для разгрузки математических вычислений на сопроцессор Xeon Phi. В качестве модельной задачи выбрано перемножение матриц, как одна из классических задач для распараллеливания программного кода. Представлены примеры разгрузки на сопроцессор Xeon Phi на языке C через инструмент PyMIC.*

Ключевые слова: *Xeon Phi, программирование, матричные вычисления, pyMIC, intel MKL*

Введение

Сопроцессоры Intel Xeon Phi часто используются для повышения производительности суперкомпьютеров собранных до 2018 . Согласно выпуску рейтинга TOP500 за июнь 2017 года, 14 систем в списке используют Xeon Phi в качестве ускорителя, а 13 — в качестве загружаемых автономных процессоров (включая текущую систему Cori с производительностью 27.8 PFLOP/s в NERCS и 24.9 PFLOP/s в Oakforest-PACS в JCAHPC в Японии, занимающие шестую и седьмую позиции в списке соответственно). В Воронежском Государственном Университете вычислительный кластер содержит 14 сопроцессоров Intel Xeon Phi 7150P.

Модели программирования для сопроцессора Xeon Phi включают так называемый режим разгрузки. В этом случае основное приложение выполняется на центральном процессоре, а на сопроцессоре выполняется только часть кода, обозначенного программистом. Для этой цели можно использовать несколько сред выполнения и библиотек. Например, прагмы openmp компилятора intel, Language Extensions for Offload (LEO), включенные в компилятор Intel. Целевые прагмы, похожие на LEO, теперь также включены в стандарт OpenMP 4.5. Другой подход заключается в использовании API низкоуровневого интерфейса разгрузки сопроцессора (COI) или Симметричный коммуникационный интерфейс (SCIF) оба предоставлены Intel. Кроме того, можно использовать внешние

библиотеки, такие как библиотека Hetero Streams Library (hStreams) или библиотека Heterogeneous Active Messages (HAM).[1] Однако зачастую, удобно использовать связку языков C и Fortran с высокоуровневыми инструментами таким как Python.

В то время как первое поколение карт Intel Xeon Phi под кодовым названием Knights Corner работает как сопроцессор и позволяет пользователям разгружать вычисления по шине PCIe, текущее поколение Knights Landing используется в качестве автономных процессоров. Однако модель программирования разгрузки расширяется за счет разгрузки по структуре если ускорители не подключены напрямую к хосту через PCIe. Существующие коды, использующие стратегию разгрузки, совместимы с разгрузкои поверх структуры и поэтому могут запускаться с минимальными изменениями на суперкомпьютерах, где узлы Xeon и Xeon Phi создают отдельные подкластеры (clusterbooster).

1. Архитектуры сопроцессора Intel Xeon Phi

Intel Xeon Phi состоит из 61 ядра, соединенных высокопроизводительным двунаправленным соединением на кристалле. Сопроцессор работает под управлением операционнои системы Linux и поддерживает все важные инструменты разработки Intel, такие как компилятор C/C++ и Fortran, MPI и OpenMP, высокопроизводительные библиотеки, такие как MKL, отладчик и инструменты трассировки, такие как Intel VTune Amplifier XE. Традиционные инструменты UNIX на сопроцессоре поддерживаются через BusyBox, которыи объединяет крошечные версии многих распространенных утилит UNIX в один небольшои исполняемыи фаил. Сопроцессор подключен к процессору Intel Xeon — «хосту» — через шину PCI Express (PICe). Осуществление виртуализированныи стек TCP/IP позволяет получить доступ к сопроцессору как к сетевому узлу. Далее мы приводим наиболее важные свои ства архитектуры MIC:

Основные:

• Процессорное ядро (скалярная единица) представляет собои упорядоченную архитектуру (на основе семеи ства процессоров Intel Pentium)

• Извлекает и декодирует инструкции из четырех аппаратных потоков

• Поддерживает 64-битную среду исполнения, а также инструкции Intel Initial Many Core.

• Не поддерживает какие-либо предыдущие расширения Intel SIMD, такие как инструкции MME, SSE, SSE2, SSE3, SSE4.1, SSE4.2 или AVX.

• Новые векторные инструкции, предоставляемые набором инструкции сопроцессора Intel Xeon Phi, используют выделенныи 512-разрядныи векторныи модуль с плавающеи запятои (VPU), которыи предусмотрен для каждого ядра.

• Высокопроизводительная поддержка операции обратного вычисления, извлечения квадратного корня, возведения в степень и экспоненты, функции разброса/сбора и потокового сохранения для достижения более высокои эффективнои пропускнои способности памяти

• Может выполнять 2 инструкции за такт, одну по U-каналу и одну по V-каналу (не все типы инструкции могут выполняться по V-каналу, например, векторные инструкции могут выполняться только по U-каналу)

2. Нативная компиляция

Для достижения хорошеи производительности следует помнить о следующих пунктах.

• Данные должны быть выровнены до 64 баи т (512 бит) для архитектуры MIC, в отличие от 32 баи т (256 бит) для AVX и 16 баи т (128 бит) для SSE.[2]

• Из-за большои ширины SIMD (64 баи та). Векторизация для архитектуры MIC даже важнее, чем для Intel Xeon. Архитектура MIC предлагает новые инструкции, такие как сбор/распределение, плавное умножение-сложение, маскированные векторные инструкции и т.д., которые позволяют распараллелить больше циклов на сопроцессоре, чем на хосте на базе Intel Xeon.

• Используи те прагмы, такие как pragma ivdep, #pragma vector always, #pragma vector aligned, #pragma т.д. Достижения автовекторизации. Автовекторизация включена на уровне оптимизации по умолчанию.

• Разрешить компилятору генерировать отчеты о векторизации с помощью опции компилятора --vecreport2 чтобы увидеть, были ли циклы векторизованы для MIC (сообщение «*MIC* Loop был векторизован» и т. д.). Варианты -opt-report-phase hlo (Отчет оптимизатора высокого уровня) или -opt-report-phase ipo_inl (Встраивание отчета) также может быть полезным.

• Явное векторное программирование также возможно с помощью языковых расширении Intel Cilk Plus (обозначение массива C/ C++,

векторные элементарные функции) или новых конструкции SIMD из OpenMP 4.0 RC1.

Элементарные функции вектора могут быть объявлены с помощью __attributes__((vector)) .Затем компилятор генерирует векторизованную версию скалярнои функции, которую можно вызвать из векторизованного цикла.

3. Intel MKL

(библиотека оптимизированных математических процедур)

Сопроцессор Intel Xeon Phi поддерживается, начиная с версии MKL 11.0. Полезная информация содержится в зоне разработчиков MKL [3]. Все функции можно использовать на Xeon Phi, однако уровень оптимизации для более широких 512-битных инструкции SIMD отличается.).

Начиная с Intel MKL 11.0 Update 2, следующие функции оптимизированы для сопроцессора Intel Xeon Phi:

BLAS Уровень 3 и большая часть Уровнеи 1 и 2

Разреженные BLAS: ?CSRMV, ?CSRMM

Некоторые важные процедуры LAPACK (LU, QR, Cholesky)

Быстрые преобразования Фурье

Библиотека векторнои математики

Генераторы случаи ных чисел в векторнои статистическои библиотеке

Intel планирует оптимизировать более широкии набор функции в будущих выпусках MKL.

4. Режимы использования MKL

Для Xeon Phi доступны следующие 3 модели использования MKL:

Автоматическая разгрузка (AO)

В случае автоматическои разгрузки пользователю вообще не нужно менять код. Для функции с автоматическои разгрузкои среда выполнения может автоматически загружать данные в сопроцессор Xeon Phi и выполнять там (все или часть) вычисления. Передача данных и управление выполнением полностью автоматизированы и прозрачны для пользователя.

Разгрузка с помощью компилятора (CAO)).[4]

В этом режиме MKL разгрузка явно контролируется прагмами или директивами компилятора. В отличие от режима автоматическои разгрузки все функции MKL могут быть разгружены в режиме CAO.

Большим преимуществом этого режима является то, что он позволяет сохранять данные на устрои стве.

Для компиляторов Intel возможно использование AO и CAO в однои□ и тои□ же программе, однако в этом случае разделение работы должно быть явно установлено для AO. В противном случае все вызовы MKL AO выполняются на хосте. [5]

Собственное исполнение

В этом режиме MKL сопроцессор Intel Xeon Phi используется как независимыи□ вычислительныи□ узел. Для сборки программы для собственного режима необходимо использовать следующие настрои□ки компилятора:

```
icc -O3 -mkl -mmic file.c -o file
```

Затем двоичныи□ фаи□л необходимо вручную скопировать в сопроцессор через ssh и запустить непосредственно на сопроцессоре.

5. PyMIC

Python один из наиболее популярных языков программирования в индустрии, использовать его достаточно легко, а скорость разработки на нём высока.

В последние годы Python привлекает особое внимание в сообществах специализирующихся на высокопроизводительных вычислениях или HPC (high performance computing).

Такие дополнения и библиотеки как Numpy и SciPy предоставляют эффективную реализацию ключевых алгоритмов и структур дынных по эффективности сравнимой такими языками как C или Fortran, что позволяет на ставить под сомнения Python как язык для высокопроизводительный вычислений.

Потребность HPC в скорости порождает необходимость в сопроцессорном оборудовании, которое бы ускоряло многочисленные вычисления с плавающей точкой.

Вычислительные блоки общего назначения (GPGPU) и сопроцессор Intel Xeon Phi являются примерами дискретных плат расширения для получения дополнительной вычислительной мощности сверх ЦП.

Приведенные выше модули предоставляют общий функционал, в свою очередь модулю pyMIC фокусируется конкретно на отправке на локальные сопроцессоры задач связанных с Numpy и SciPy.

Основная логика работы в системе pyMIC заключается в следующем:

Математический код написанный на языках C или Fortran собирается в разделяемую библиотеку. В операционной системе типа Linux такие библиотеки имеют расширение .so.

Ниже представлена команда, компилирующая разделяемую библиотеку на языке C.

icc -I../../include -I$(MKLROOT)/include -fPIC -shared -mmic -g -O2 -L$(MKLROOT)/lib/mic -lmkl_intel_lp64 -lmkl_core -lmkl_intel_thread -lpthread -o libdgemm.so dgemm.c

После этого необходимо выполнить разгрузку вычислений на математический сопроцессоре. Приведенные ниже команды соответствуют инициализации объектов устройств, раннее созданных разделяемых библиотек и потоков по умолчанию.

Листинг 1

```
device = mic.devices[0]
library = device.load_library("libdgemm.so")
stream = device.get_default_stream()
```

Для дальнейшей работы и взаимодействия, необходимо связать оперативную память компьютера и память математического сопроцессора,

Листинг 2

```
    offl_a = stream.bind(a)
offl_b = stream.bind(b)
offl_c = stream.bind(c)
```

Вызова функции из собранной библиотеки, передачи дынных и синхронизации памяти. Выглядит следующем образом:

Листинг 3

```
stream.invoke(library.dgemm_kernel, offl_a, offl_b, offl_c,
              m, n, k, alpha, beta)
stream.sync()
```

Ниже приведены результаты перемножение матриц 4096x4096 с помощью библиотеки NumPy, которая базируется на BLAS из языка Fortran и библиотеки MKL, имеющих высокую степень оптимизации под сопроцессор производства Intel.

Параметры машины, на которой выполнялись вычисления, следующие: Intel i7-3820 имеет 4 ядра и 8 потоков и Intel Xeon Phi 7150P имеет 64 ядра и 244 потока.

Performance:

MxM (numpy)	1.62 sec
MxM (offload)	1.03 sec

Библиотеки NumPy и MKL являются примерами хорошо оптимизированного кода, интересно посмотреть, как ведет себя система PyMIC и Intel Xeon Phi с более простым кодом. Для оценки возможностей использования связки PyMIC и Intel Xeon Phi с менее оптимизированными приложениями, нами было реализовано перемножение матриц без глубокой оптимизации и векторизации данных.

Листинг 4

```
#include <pymic_kernel.h>
#include <omp.h>


PYMIC_KERNEL
void multiplication(const double *A, const double *B,
                  double *C, const long int *nrows, const long
int *ncols){

#pragma omp parallel for collaps(2)

    for(int i = 0; i < *nrows; i++)
        for(int j = 0; j < *ncols; j++)
        {
           for(int k = 0; k < *ncols; k++)
               *(C+i*1024+j) += *(A+i*1024+k) *
*(B+k*1024+j);
        }

    }
```

```
#include <pymic_kernel.h>
#include <omp.h>
```

Листинг 5

```
PYMIC_KERNEL
void multiplication(const double *A, const double *B,
                  double *C, const long int *nrows, const long
int *ncols){
```

```
    for(int i = 0; i < *nrows; i++)
        for(int j = 0; j < *ncols; j++)
        {
#pragma omp parallel for

            for(int k = 0; k < *ncols; k++)
                *(C+i*1024+j) += *(A+i*1024+k) *
*(B+k*1024+j);
        }

    }
```

Performance:

MxM (numpy)	0.058 sec
MxM (offload)	1.82 sec

Полученные результаты показывают, что скорость работы более оптимизированной программы с использованием NumPy превосходят написанную нами программу без оптимизации и векторизации, несмотря на использование Intel Xeon Phi и распараллеливание с помощью OpenMP. Отсутствие векторизации являться большой проблемой для Xeon Phi.

Листинг 6

Код скрипта для PyMIC

```
#!/usr/bin/python
from __future__ import print_function

import pymic as mic
import numpy as np
import sys
import time

# load the library with the kernel function (on the target)
device = mic.devices[0]
library = device.load_library("libdgemm.so")

# use the default stream
stream = device.get_default_stream()

ds = 1024
m, n, k = ds, ds, ds
if len(sys.argv) > 1:
    sz = int(sys.argv[1])
    m, n, k = sz, sz, sz
```

```
# construct some matrices
np.random.seed(10)
a = np.random.random(m * k).reshape((m, k))
b = np.random.random(k * n).reshape((k, n))
c = np.zeros((m, n))

# associate host arrays with device arrats
offl_a = stream.bind(a)
offl_b = stream.bind(b)
offl_c = stream.bind(c)

# convert a and b to matrices (eases MxM in numpy)
Am = np.matrix(a)
Bm = np.matrix(b)

# print the input
print("input:")
print("---------------------------------------")
print("a=", a)
print("b=", b)
print()
print()

# print the input of numpy's MxM if it is small enough
np_mxm_start = time.time()
Cm = Am * Bm
np_mxm_end = time.time()
print("numpy gives us:")
print("---------------------------------------")
print(Cm)
print("checksum:", np.sum(Cm))
print()
print()

print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++')

# invoke the offloaded dgemm
c[:] = 0.0
offl_c.update_device()
np_mic_start = time.time()
stream.invoke(library.dgemm_kernel, offl_a, offl_b, offl_c,
              m, n, k, alpha, beta)
stream.sync()
np_mic_end = time.time()


offl_c.update_host()
stream.sync()
print(offl_c)
print("_____________________")
# print the performance information
```

```
flops = (2 * m * n * k) / 1000 / 1000 / 1000
np_mxm_time = np_mxm_end - np_mxm_start
np_mic_time = np_mic_end - np_mic_start
print("performance:")
print("---------------------------------------")
print("MxM (numpy)            {0:>6.3} sec    "
      "{1:>6.3} GFLOPS".format(np_mxm_time, flops /
np_mxm_time))
print("MxM (offload)          {0:>6.3} sec    "
      "{1:>6.3} GFLOPS".format(np_mic_time, flops /
np_mic_time))

print()
sum1 = np.sum(Cm)
sum2 = np.sum(offl_c.array)
if sum1 != sum2:
    print('Validation failed: ', sum1, '!=', sum2, 'diff',
abs(sum1-sum2))
else:
    print('Validation succeeded: ', sum1, '==', sum2)
```

Заключение

В данной статье были рассмотрены разные примеры выполнения кода с использованием Xeon Phi. В ходе проведенных вычислений, можем сказать что PyMIC в связки с MKL позволяет добиться прироста в 50% на сопроцессоре относительно библиотеки NumPy на обычном процессоре. Но выполнения обычного кода на сопроцессоре, в разы уступает NumPy на обычных процессах. Это происходить из-за сложности оптимизации кода под Xeon Phi, с следствие чего использование сопроцессор не имеет большой выгоды относительно оптимизированных программ под процессоры.

Список литературы

1. Best Practice Guide - Intel Xeon Phi / Nevena Ilieva, Michael Schliephake, Sami Saarinen, Volker Weinberg - LRZ Germany 2014. - 49 с.
2. Optimizing Xeon Phi for Interactive Data Analysis / Chansup Byun, Jeremy Kepner, William Arcand [и др.] // MIT Lincoln Laboratory Supercomputing Center. - 2019 - С. 1-6.
3. Evaluation of the Intel Xeon Phi offload runtimes for domain decomposition solvers / Lukas Maly, Jan Zapletala, Michal Mertaa, [и др.] // Advances in Engineering Software. - 2018. -С. 1-9.
4. GPAW: DFT and beyond within the projector-augmented wave method: сайт. -URL: https://wiki.fysik.dtu.dk/gpaw (дата обращения: 15.5.2022). – Текст: электронный.

5. Intel Corporation. Intel R Manycore Platform Software Stack (MPSS), 2014.: сайт. -URL: https://software.intel.com/en-us/articles/ intel- manycore- platform- software- stack- mpss. (дата обращения: 20.5.2022). – Текст: электронный.