Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики

Образовательный комплекс «Введение в принципы функционирования и применения современных мультиядерных архитектур (на примере Intel Xeon Phi)»

Лекция №3
Выполнение программ на Intel Xeon Phi. Модели организации вычислений с использованием Intel Xeon Phi

Линев А.В.

При поддержке компании Intel

Нижний Новгород
2013

Содержание

1. Архитектура и состав ПО, обеспечивающего выполнение программ на Intel Xeon Phi

1.1. Intel Manycore Platform Software Stack (MPSS)

Архитектура и состав программного обеспечения для сопроцессора Intel Xeon Phi ориентированы на выполнение высокопроизводительных приложений, способных максимально использовать возможность одновременного выполнения сотен потоков. Встроенное ПО позволяет использовать его в системах с шиной PCI Express, работающих под управлением операционных систем Linux или Windows.

Сопроцессор Intel Xeon Phi – это высокопроизводительная многоядерная система, реализованная в виде карты расширения PCI Express и соответствующая всем спецификациям PCI Express. В частности, сопроцессор реализует все необходимые интерфейсы и обслуживает запросы основной системы по опросу его состояния и конфигурированию. Базовая операционная система использует драйвер, взаимодействующий с сопроцессором как с устройством PCE Express.

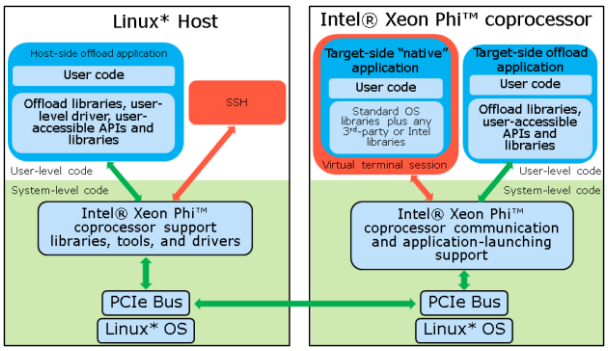

Рис. 1 Архитектура ПО сопроцессора Intel Xeon Phi

Библиотеки обеспечивают базовую функциональность, такую как определение сопроцессоров в системе, передачу данных между хостом и сопроцессором, загрузку исполняемых файлов на Xeon Phi и их запуск. Инструментальные средства позволяют управлять настройкам сопроцессора, получать информацию о его состоянии, обновлять его флеш-память и т.д.; с помощью ssh можно получить терминальный доступ к сопроцессору для запуска на нем программ.

С точки зрения базовой ОС сопроцессор представляет собой отдельный вычислительный SMP-домен, слабо связанный с основными процессорами системы. Поскольку сопроцессор интегрирован в базовую систему в виде

карты расширения PCI Express, взаимодействие между базовой системой и сопроцессорами можно осуществлять несколькими различными способами, что позволяет реализовать и предоставить разработчикам несколько программных моделей использования сопроцессора. Для обеспечения разработки высокопроизводительных приложений доступны реализации ряда стандартных API: сокеты TCP/IP, MPI, OpenCL. Также доступны специализированные интерфейсы, позволяющие получать доступ к возможностям сопроцессора, например, SCIF API (Symmetric Communication Interface API) обеспечивает взаимодействие между базовой системой и сопроцессором, и API высокого уровня используют именно его для организации пересылок. Следующий рисунок иллюстрирует взаимосвязи между различными API, реализованными в MPSS (Intel MIC Architecture Manycore Platform Software Stack).

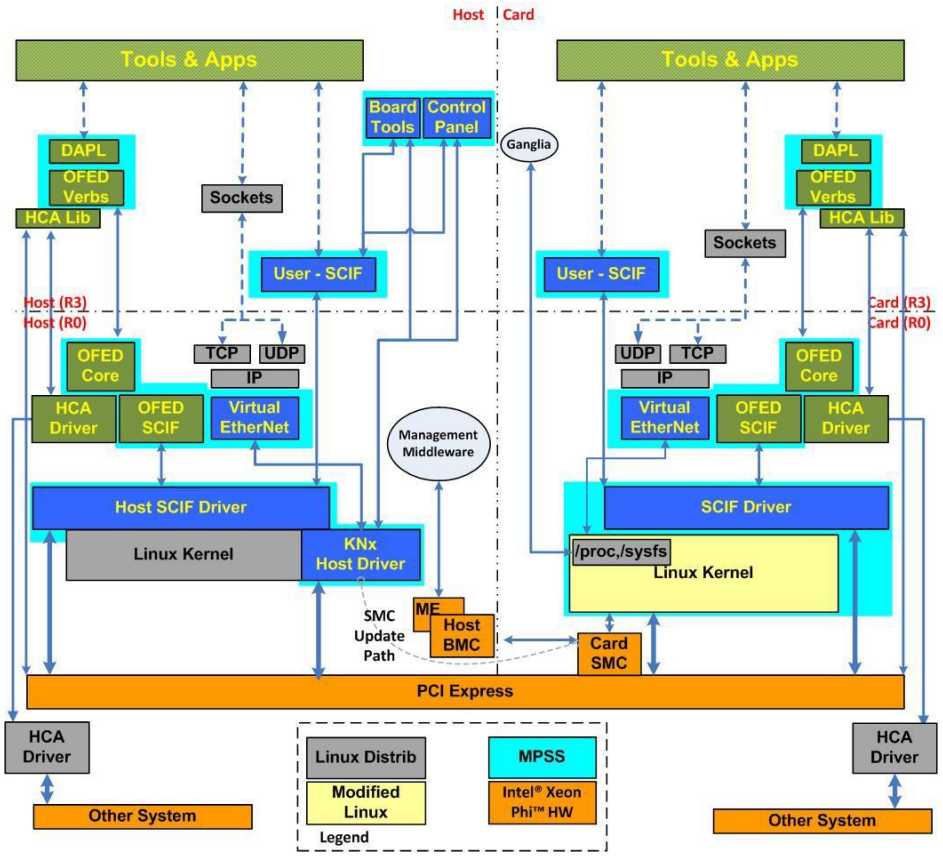

Рис. 2 Компоненты ПО сопроцессора Intel Xeon Phi

На левой стороне схемы отображены программные компоненты, выполняющиеся на стороне базовой системы. В качестве операционной системы используется ядро Linux. На правой стороне приводятся программные

компоненты, работающие на сопроцессоре; исполняющееся на нем ядро Linux адаптировано для архитектуры Intel Xeon Phi. Схема отражает структуру программного обеспечения по окончании загрузки базовой операционной системы и ОС сопроцессора.

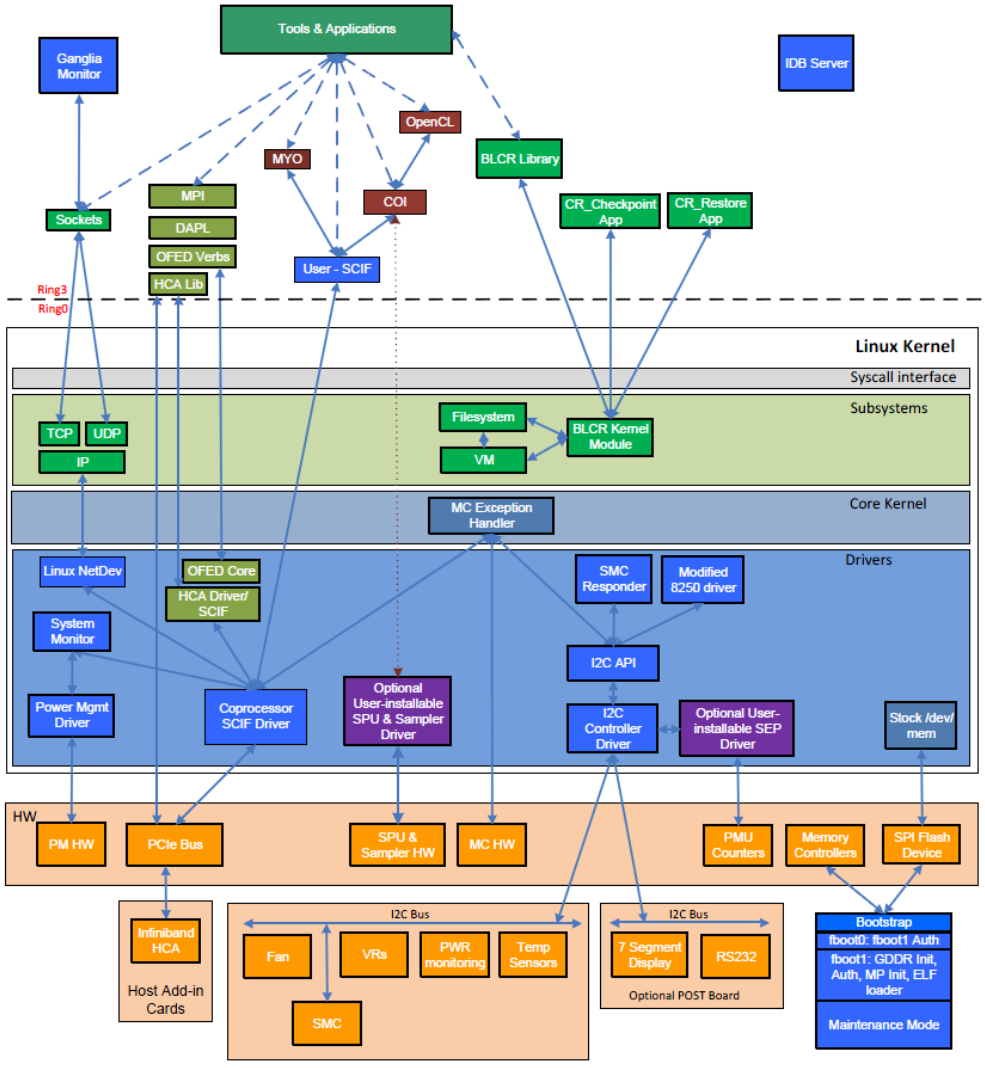

Рис. 3 Intel Xeon Phi Coprocessor Software Stack

Поскольку архитектура Intel Xeon Phi похожа на другие архитектуры Intel, процесс старта и загрузки операционной системы сопроцессора похож на загрузку ОС базовой системы. Аналогом системного BIOS является Bootstrap, который начинает выполняться при подаче электрического питания на сопроцессор или его перезагрузке ответственен за загрузку операционной системы сопроцессора. Bootstrap состоит из двух частей (fboot0 и fboot1), первая из которых расположена в ROM сопроцессора и не может

быть изменена, а вторая располагается во флеш-памяти и допускает обновление. В ходе работы второй части Bootstrap выполняется загрузка операционной системы сопроцессора из основной системы и передача управления ее загрузчику (Linux Loader).

Операционная система сопроцессора Intel Xeon Phi базируется на стандартном ядре Linux (kernel.org), в которое были внесены минимально возможные изменениями, требуемые для поддержки новой архитектуры и учитывающие отсутствие в сопроцессоре ряда стандартных для вычислительных систем аппаратных компонентов. ОС сопроцессора обеспечивает стандартные возможности, такие как создание/завершение процессов, планирование исполнения потоков, управление памятью, электропитанием, конфигурацией и т.д. Для обеспечения управления специфичными компонентами сопроцессора используется специальный драйвер.

1.2. Симметричный коммуникационный интерфейс

Симметричный коммуникационный интерфейс (Symmetric Communication Interface, SCIF) – базовый механизм взаимодействия между процессорами основной системы и сопроцессором. Он обеспечивает взаимодействие между основными процессорами системы и сопроцессорами, а также между несколькими сопроцессорами, установленными в одной системе. SCIF использует возможности механизма прямого доступа к памяти (DMA) со стороны сопроцессора, а также возможность отображения физической памяти системы или любого сопроцессора в виртуальное адресное пространства любого процесса, выполняющегося в базовой системе или на любом сопроцессоре. Взаимодействие между парой клиентов SCIF основано на прямом доступе к памяти друг друга, в частности, взаимодействие между двумя сопроцессорами производится без использования оперативной памяти основной системы.

Набор драйверов, используемых для управления сопроцессором на стороне базовой операционной системы, называется базовым драйвером (Host driver). Его первичной задачей является инициализация сопроцессоров Intel Xeon Phi, что включает загрузку на них операционных систем с необходимыми параметрами. По завершении загрузки сопроцессоров базовый драйвер служит основой SCIF-коммуникаций. Он непосредственно не обеспечивает никаких средств межпроцессного взаимодействия, но все высокоуровневые механизмов (сокеты, MPI и т.д.) базируются на предоставляемой им функциональности.

Для передачи данных между сопроцессором и основной системой можно использовать либо копирование памяти, либо DMA-передачу. Копирование памяти оптимизировано для многопоточного использования, обеспечивая параллельное обслуживание множества запросов для максимального

использования пропускной способности шины PCI Express. Однако, когда необходимо уменьшить нагрузку на процессоры основной системы или размер передаваемых данных превышает размер пакета PCI Express, предпочтительней использовать прямой доступ к памяти. Также необходимо отметить, что DMA-передачи, инициированные процессором основной системы, имеют меньшую латентность, чем передачи, инициированные сопроцессором. Приложения могут явно указывать, какой способ передачи следует использовать, либо оставить это на усмотрение драйвера.

Ganglia – масштабируемая распределенная система мониторинга для высокопроизводительных вычислительных систем. Существуют версии для различных архитектур и операционных систем, и она используется на тысячах кластеров. Для интеграции с ganglia и подобным ей системам MPSS предоставляет стандартный интерфейс доступа к данным мониторинга на всех сопроцессорах Intel Xeon Phi (виртуальные файловые системы proc или sysfs), а также предоставляет образцы плагина для сбора дополнительных метрик и настроек системы мониторинга. Каждый сопроцессор представляется в системе мониторинга как отдельный узел.

Для обеспечения эффективности работы высокопроизводительных приложений, для сопроцессора разработан OFED (OpenFabrics Enterprise Distribution) – комплекс программных средств для использования RDMA (Remote Direct Memory Access, удаленный прямой доступ к памяти) и выполнения приема/передачи данных без вызовом ядра ОС. OFED широко используется в приложениях, требующих эффективной работы с сетью и хранилищами данных, и при организации параллельных вычислений. Также, OFED является предпочтительным механизмом передачи для библиотеки Intel MPI, которая может в таком случае использовать интерфейс SCIF для организации взаимодействия между сопроцессорами Intel Xeon Phi в рамках одного узла и между сопроцессором и процессорами базовой системы. При таком взаимодействии с точки зрения MPI каждый сопроцессор считается отдельным узлом.

Реализация библиотеки Intel MPI для архитектуры MIC поддерживает работу только с системой управления параллельными заданиями Hydra. Каждый узел и каждый сопроцессор Intel Xeon Phi идентифицируются уникальным символьным именем или IP-адресом, что позволяет выбрать для каждого процесса исполняемый файл, соответствующий архитектуре, на которой он будет выполняться.

Intel Debugger (idb) – средство отладки гетерогенных приложений, позволяющее отлаживать как приложения, работающие только на сопроцессоре, так и приложения, использующие и центральный процессор, и сопроцессор. Для отладки приложений, использующих Intel Xeon Phi, можно также использовать GNU Project Debugger (gdb).

2. Модели использования сопроцессора Intel Xeon Phi

Архитектура Intel Xeon Phi поддерживает несколько режимов использования сопроцессора, которые можно комбинировать для достижения максимальной производительности в зависимости от характеристик решаемой задачи. Процесс может быть запущен как в операционной системе базовой системы, так и в ОС сопроцессора; в зависимости от режима использования могут использоваться вычислительные мощности только процессоров базовой системы, либо только сопроцессора, либо процессоров базовой системы и сопроцессора совместно.

Библиотека Intel MPI для ОС Linux соответствует версии 2.1 стандарта MPI (MPI-2.1) и базируется на реализациях MPICH2 и MVAPICH2. Intel планирует обеспечить полную функциональность библиотеки для использования на любых конфигурациях процессоров Intel Xeon и Intel Xeon Phi, чтобы обеспечить разработчикам единообразное окружение разработки и исполнения MPI-приложений. В настоящий момент не поддерживается ряд разделов спецификации: динамическое управление процессами, файловый ввод-вывод MPI, односторонние передачи пассивному получателю (получатель не вызывает функции MPI).

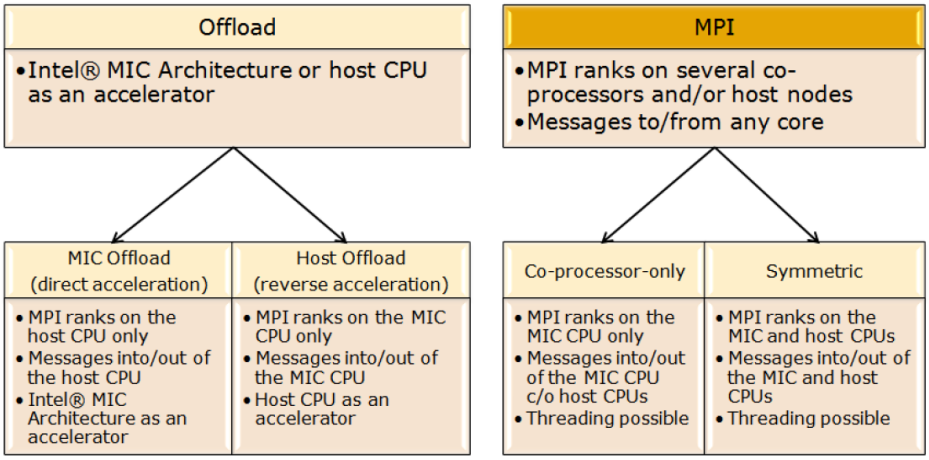

Рис. 4 Режимы и модели программирования для Intel Xeon Phi

Поддерживается два режима выполнения приложений: режим Offload и режим MPI.

При работе в режиме Offload используется один из следующих подходов:

- Процессы MPI выполняются на процессорах Xeon базовой системы, для вычислений может использоваться запуск функций на сопроцес-

core Xeon Phi. Данная модель поддерживается компиляторами C, C++, Fortran для архитектуры MIC, библиотекой Intel MKL, а также Intel MPI for Linux начиная с версии 4.0. Update 3.

- Процессы MPI выполняются на процессорах Xeon Phi с возможным запуском выполнения функций на процессорах Xeon базовой системы. Данная модель не поддерживается текущими версиями компиляторов и библиотек.

В режиме MPI базовая система и каждый сопроцессор Intel Xeon Phi рассматриваются как отдельные равноправные узлы, и процессы MPI могут выполняться на процессорах Xeon базовых систем и сопроцессорах Xeon Phi в произвольных сочетаниях. Поддерживаются следующие три основные модели:

- Модель симметричного выполнения (Symmetric model)

  MPI-процессы выполняются как на процессорах базовой системы, так и на сопроцессорах. Эта модель выглядит естественной для использования на гетерогенных кластерах, но требуется обеспечить адекватной распределение нагрузки между процессами, выполняющимися на процессорах базовой системы и сопроцессорах. Вторая и третья модели являются частными случаями первой.

- Модель использования только сопроцессоров (Coprocessor-only model)

  Все MPI-процессы выполняются на сопроцессорах.

- Модель использования только процессоров базовой системы (Host-only model)

  Все MPI-процессы выполняются на процессорах базовой системы, сопроцессоры не используются.

2.1. Режим выполнения Offload

В режиме Offload все MPI-процессы выполняются на процессорах базовой системы, и MPI-взаимодействие не затрагивает сопроцессоры. Для использования вычислительных возможностей Xeon Phi используется выгрузка и выполнение функций на сопроцессоре, при этом вызов функций MPI в выгруженном коде не поддерживаться.

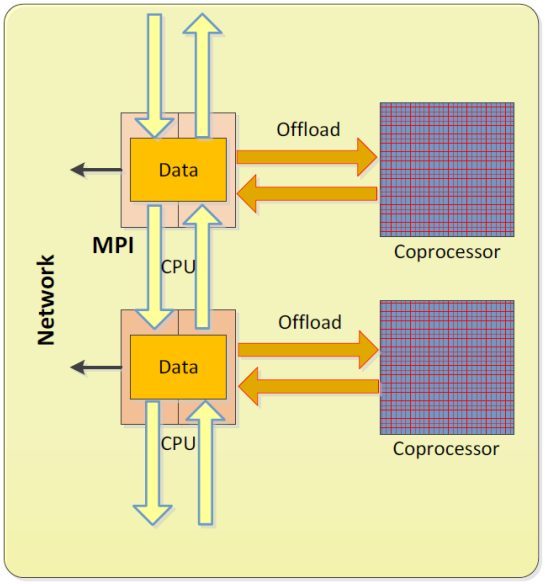

Рис. 5 MPI на хост-системах с вызовом offload-кода на сопроцессоре

Необходимо отметить, что суммарный размер выгружаемых на сопроцессор кода и данных не должен превышать 85% объема памяти сопроцессора.

2.2. Модель использования только сопроцессоров (Coprocessor-only model)

Данная модель предполагает выполнение MPI-процессов только на сопроцессорах, и также называется «родной» (native) моделью для архитектуры Intel MIC.

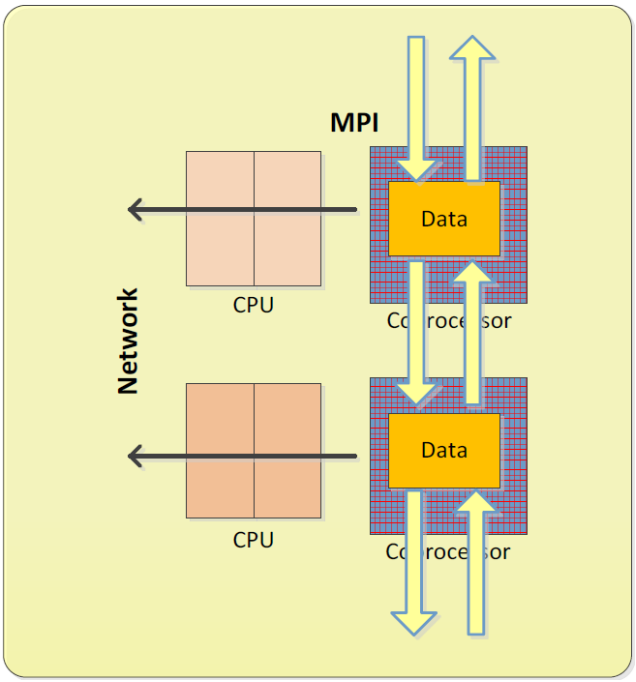

Рис. 6 MPI-процессы исполняются только на сопроцессорах Intel Xeon Phi

Приложение, библиотека MPI и другие необходимые библиотеки загружаются на сопроцессор для выполнения, после чего запуск приложения может быть произведен как из базовой системы, так и из системы сопроцессора.

2.3. Модель симметричного выполнения (Symmetric model)

В данной модели MPI-процессы выполняются как на процессорах базовой системы, так и на сопроцессорах. Передача сообщений между процессорами базовой системы, в пределах сопроцессора и между сопроцессором и процессорами базовой системы может выполняться через механизмы разделяемой памяти или протокола tcp. По умолчанию используется tcp, для использования более производительного в данных случаях механизма разделяемой памяти необходимо для MPI-приложения установить значение переменной окружения: I_MPI_SSHM_SCIF={enable|yes|on|1}.

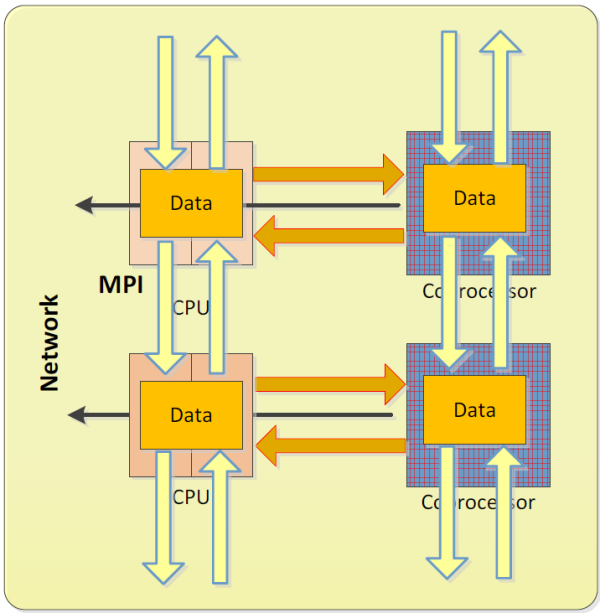

Рис. 7 MPI-процессы исполняются на процессорах базовой системы и на сопроцессорах Intel Xeon Phi

Приведем пример запуска параллельной программы, использующей модель симметричного выполнения. Посредством команды системы управления процессами Hydra выполняется запуск 4 процессов по 4 потока в каждом на основных процессорах и 2 процесса по 16 потоков в каждом на сопроцессоре mic0.

(host)$mpiexec.hydra –host $(hostname) -n 4 –env OMP_NUM_THREADS 4 ./test.exe.host -host mic0 –n 2 –env OMP_NUM_THREADS 16 –wdir /tmp /tmp/test.exe.mic

3. Создание приложений для Intel Xeon Phi

Разработка приложений для архитектуры Intel Xeon Phi требует наличия тех же знаний и навыков, что и разработка параллельных приложений для распределенных многоядерных систем. Можно использовать следующие программные инструменты:

- средства разработки «Intel Parallel Studio XE 2013», «Intel Cluster Studio XE 2013», «Intel(R) SDK for OpenCL Applications XE 2013 Beta», gcc (в настоящий момент не поддерживает векторные инструкции) и др.;

- библиотеки Intel Math Kernel Library (Intel MKL), Intel Threading Building Blocks (Intel TBB), Intel Integrated Performance Primitive (Intel IPP), входящие в состав средств разработки Intel, а также Intel MPI for Linux, MPICH2, Boost и др.
- отладчики (Intel Debugger, gdb, totalview), профилировщики (входят в состав средств разработки Intel), средства виртуализации (xen) и т.д.

Компиляция гетерогенных приложений производится в базовой системе. В случае использования режима Offload, для всех offload-блоков создается две версии кода – для базовой системы и для сопроцессора. Компилятор создает исполняемые файлы и/или библиотеки, содержащие весь код для процессора и сопроцессора. Во время исполнения программы при первом вызове offload-кода проверяется наличие сопроцессора Intel Xeon Phi. В случае его наличия и незанятости происходит загрузка на него двоичного исполняемого файла и инициализация необходимых библиотек, после чего выполняется вызов offload-кода. В случае отсутствия сопроцессора происходит вызов версии функции для базовой системы. Таким образом, приложение будет работать как при наличии сопроцессора, так и при его отсутствии.

Рассмотрим несколько версий программы сложения элементов массива для различных моделей выполнения.

3.1. Программы для режима выполнения Offload

При использовании режима Offload порядок использования сопроцессора определяется посредством директив компилятора – pragma в языках C/C++, директив в FORTRAN.

3.1.1. Последовательное синхронное выполнение

Блок программы, который следует выполнить на сопроцессоре, указывается посредством директивы #pragma offload target(mic). Операторы in, out, inout определяют необходимость и направление передачи данных между памятью хоста и сопроцессора. По умолчанию все переменные, объявленные вне offload-блока, перед началом его выполнения копируются на сопроцессор, а по окончании выполнения копируются назад в память базовой системы.

```
float Sum(float *Data, int Size){
   float Ret = 0.0f;
#pragma offload target(mic) in(Data:length(Size))
   for (int i = 0; i < Size; i ++){
      Ret += Data[i];
   }
   return Ret;
}
```

В данном примере переменная Ret перед началом выполнения offload-кода будет скопирована из памяти хост-системы в память сопроцессора, а по окончании – назад из памяти сопроцессора в основную память системы.

Offload-код данной версии будет выполняться последовательно на одном ядре сопроцессора.

3.2. Последовательное синхронное выполнение с векторизацией

Компилятор Intel по умолчанию выполняет векторизацию кода. Можно облегчить ему задачу, используя операции с массивами в формате Intel Cilk Plus Extended Array Notation.

```
float Sum(float *Data, int Size){
   float Ret = 0.0f;
#pragma offload target(mic) in(Data:length(Size))
   //Intel Cilk Plus Extended Array Notation
   Ret = __sec_reduce_add(Data[0:Size]);
   return Ret;
}
```

Offload-код данной версии будет выполняться последовательно на одном ядре сопроцессора с использованием векторных вычислений, выполняя по 16 операций сложения за одну операцию.

Последовательное асинхронное выполнение

При использовании режима Offload можно использовать технику двойной буферизации, обеспечивающую одновременное выполнение offload-функции и передачу на сопроцессор входных данных для следующего вызова и/или передачу выходных данных в память основной системы для предыдущего вызова (подробнее см. [2]). Пример программы, использующей двойную буферизацию, имеется в составе средств разработки Intel (…/C++/mic_samples/intro_sampleC/sampleC13.c)

3.3. Явное копирование памяти

Для передачи между хостом и сопроцессором сложных структур данных, например, использующих указатели, в языках C/C++ реализована модель «разделяемой памяти». Она обеспечивает размещение специальным образом маркированных переменных (квалификатор типа _Cilk_shared) по одним и тем же виртуальным адресам на хост-системе и сопроцессоре, а также включает специальные функции для динамического выделения памяти по одним и тем же адресам на хост-системе и сопроцессоре.

«Разделяемая память» не может быть реализована непосредственным отображением адресов памяти сопроцессора на адреса памяти хост-системы, эти подсистемы памяти являются полностью независимыми. Этот механизм является вариацией обычного вызова offload-кода – при выполнении вызова функции с использованием квалификатора _Cilk_offload определяется, какие изменения произошли в копии, хранящейся в памяти хост-системы, и изменения передаются в память сопроцессора (аналогично при возврате из функции).

```
float * _Cilk_shared Data;

_Cilk_shared float MIC_Sum(int Size) {
   float Result;

   for (int i = 0; i < Size; i++){
      Result += Data[i];
   }
   return Result;
}

int main(){
   size_t Size = 1000000;
   int MemSize;

   MemSize = Size * sizeof(float);
   Data = (_Cilk_shared float *) _Offload_shared_malloc
(MemSize);
   for (int i = 0; i < Size; i++){
      Data[i] = i;
   }
   _Cilk_offload MIC_Sum(Size);
   _Offload_shared_free(Data);
   return 0;
}
```

Offload-код данной версии будет выполняться последовательно на одном ядре сопроцессора.

3.4. Параллельное программирование для сопроцессора Intel Xeon Phi

Большинство возможностей, доступных для хост-системы, реализованы и для Intel Xeon Phi. Вы можете использовать:

- Intel Threading Building Blocks (Intel® TBB)
- OpenMP*
- Intel® Cilk Plus

- pthreads*
- MPI

Необходимо помнить, что потоки программ, выполняющихся на хост-системе и на сопроцессоре, являются совершенно независимыми, и можно использовать, например, OpenMP только в части программы, выполняющейся на хосте, или только в части, выполняющейся на сопроцессоре, или в обоих частях сразу.

Использование MPI не отличается от разработки программ для кластеров, состоящих из многоядерных узлов, за исключением необходимости распределения нагрузки при использовании симметричной модели выполнения.

4. Литература

1. Intel and Third Party Tools and Libraries available with support for Intel® Xeon Phi™ Coprocessor [http://software.intel.com/en-us/articles/intel-and-third-party-tools-and-libraries-available-with-support-for-intelr-xeon-phitm].
2. User and Reference Guide for the Intel® C++ Compiler [http://software.intel.com/en-us/compiler_14.0_ug_c]Reinders J. An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors. [http://software.intel.com/en-us/blogs/2012/11/14/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi].
3. Loc Q Nguyen et al. Intel Xeon Phi Coprocessor Developer's Quick Start Guide. [http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide].
4. Pseudo-LRU. [http://en.wikipedia.org/wiki/Pseudo-LRU].
5. Intel Xeon Phi Coprocessor System Software Developers Guide. [http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide].
6. Rahman R. Intel Xeon Phi Core Micro-architecture [http://software.intel.com/en-us/articles/intel-xeon-phi-core-micro-architecture].