УДК 519.682.3

О ПОДДЕРЖКЕ РАБОТЫ T++-ПРИЛОЖЕНИЙ НА ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРАХ НА БАЗЕ ПРОЦЕССОРОВ INTEL XEON И УСКОРИТЕЛЕЙ INTEL XEON PHI

В.А. Роганов, научный сотрудник; А.А. Кузнецов, научный сотрудник;
Г.А. Матвеев, ведущий инженер-исследователь (программист);
В.И. Осипов, к.ф.-м.н., научный сотрудник
(Федеральное государственное бюджетное учреждение науки Институт программных систем им. А.К. Айламазяна РАН, ул. Петра Первого, 4а, с. Веськово, Ярославская обл., 152021, Россия, var@pereslavl.ru, tonic@pereslavl.ru, gera@prime.botik.ru, val@pereslavl.ru)

Аннотация. В статье дано краткое описание программно-аппаратной архитектуры Intel MIC, а также методов реализации поддержки работы Т-приложений на гибридных кластерах, узлы которых построены на базе процессоров Intel Xeon и ускорителей Intel Xeon Phi, представлены способы оптимизации работы приложений T++ на данных ускорителях и результаты их тестовых прогонов на узлах гибридного кластера. В качестве тестового T++-приложения была выбрана задача «Вычисление числа π методом Монте-Карло».

Ключевые слова: *язык программирования T++, OpenTS, T-система, параллельное расширение Си++, гибридные кластеры, архитектура Intel MIC, процессоры Intel Xeon, ускорители Intel Xeon Phi.*

T-система с открытой архитектурой (OpenTS) была разработана в ИПС им. А.К. Айламазяна РАН в рамках суперкомпьютерной программы «СКИФ» Союзного государства России и Беларуси [1–9]. Она представляет собой современную реализацию идей Т-системы и обеспечивает лучшую, чем предыдущие версии системы, интеграцию базовых возможностей функционального подхода с возможностями языка программирования Си++.

Т-система обладает открытой и масштабируемой архитектурой, поддерживает входной язык программирования T++, являющийся синтаксически и семантически гладким расширением языка программирования Си++ за счет включения в него некоторого конечного набора ключевых слов.

Приложения, собранные с помощью Т-системы, имеют хорошую масштабируемость и позволяют добиваться хорошего ускорения на современных кластерных установках.

Архитектура системы OpenTS и принятые в ходе ее разработки технические решения дают возможность сравнительно легко адаптировать систему для эффективной работы на узлах кластера, содержащих ускорители Intel Xeon Phi. Выполнение данной работы дает надежду на хорошие результаты утилизации вычислительной мощности гибридных вычислительных кластеров при прогоне параллельных T++-приложений и позволит решить задачи сокращения временных трудозатрат при создании параллельных приложений для гибридных кластерных установок, а также повысить эффективность использования гибридных суперкомпьютерных установок для решения научных, производственных и социальных задач.

В статье рассматриваются ускоритель Intel Xeon Phi, построенный на архитектуре Intel MIC, и методы реализации поддержки работы Т-приложений на гибридных кластерах, узлы которых построены на базе процессоров Intel Xeon и ускорителей Intel Xeon Phi.

Intel MIC (Intel Many Integrated Core) – архитектура многоядерной процессорной системы [10], разработанная корпорацией Intel, в которой применяются очень широкие векторные АЛУ (512-разрядные SIMD) в микропроцессорах с классической архитектурой x86.

Методы реализации поддержки работы Т-приложений на вычислительных кластерах с гибридными узлами на базе ускорителей Intel Xeon Phi

Программно-аппаратная архитектура ускорителей Intel Many Integrated Core (Intel MIC)

Для программирования MIC-ускорителей предлагается использовать OpenMP, OpenCL, Intel Cilk Plus, специализированные компиляторы Intel Fortran, Intel C++. Также предоставляются математические библиотеки. На базе архитектуры MIC выпущены сопроцессоры под маркой Intel Xeon Phi, которые используют преимущества привычных языков программирования, параллельных моделей, инструментов и методик разработки, поддерживаемых архитектурами Intel. Это дает возможность более эффективно использовать параллельный код без необходимости изучения привязанных к определенному аппаратному обеспечению моделей программирования. Компания Intel предоставляет все необходимые

программные инструменты (включая Intel Parallel Studio XE и Intel Cluster Studio XE), позволяющие оптимизировать программы для работы на сопроцессорах Intel Xeon Phi.

Аппаратные продукты на базе архитектуры Intel MIC дают разработчикам ключевое преимущество: возможность использовать стандартные программные инструменты и методы. Программисты могут использовать унаследованный исходный код на языках C, C++ и FORTRAN. Можно также компилировать и выполнять на стандартных процессорах Intel Xeon исходный код, написанный для работы на Intel MIC-ускорителях.

Архитектура Intel MIC предназначена для областей и систем, в которых интенсивно используется параллельная обработка, включая высокопроизводительные вычислительные системы (HPC), рабочие станции и центры обработки данных. Высокая степень параллелизма в MIC-архитектуре достигается за счет использования процессорных ядер меньшего размера, потребляющих меньше электроэнергии. Результатом является высокая производительность в системах с интенсивной параллельной обработкой данных.

Сопроцессор Intel Xeon Phi

Сопроцессоры Intel Xeon Phi [11], созданные на базе архитектуры Intel MIC, работают на операционной системе Linux, поддерживают модель памяти x86 и арифметику с плавающей точкой IEEE 754, могут исполнять приложения, написанные на стандартных языках программирования, таких как C, C++, FORTRAN. Приложения для сопроцессора могут быть разработаны при помощи интегрированной среды разработки (Intel Composer XE 2013), которая включает в себя компиляторы и библиотеки, такие как библиотеки потоков (threading libraries), математические библиотеки высокой производительности, различные инструменты и отладчики.

Сопроцессор Intel Xeon Phi подключен к процессору Intel Xeon (host) через шину PCI Express (PCIe). Поскольку сопроцессор Intel Xeon Phi работает под операционной системой Linux, виртуальный TCP/IP-стэк может быть реализован по шине PCIe, что позволяет пользователю получить доступ к сопроцессору как к сетевому узлу. Таким образом, любой пользователь может подключиться к сопроцессору через Secure Shell и непосредственно запускать отдельные задачи или пакетные задания (batch jobs). Сопроцессор также поддерживает гетерогенные приложения, в которых часть приложения выполняется на головном узле, в то время как другая часть – на сопроцессоре.

Несколько сопроцессоров Intel Xeon Phi могут быть установлены в одной компьютерной системе (на одном host-узле). В рамках единой системы сопроцессоры могут обмениваться информацией друг с другом через PCIe-соединение (взаимодействие «точка–точка») без какого-либо вмешательства со стороны host-узла. Аналогичным образом сопроцессоры могут обмениваться данными через сетевую карту, такую как InfiniBand или Ethernet (см. рис. 1).

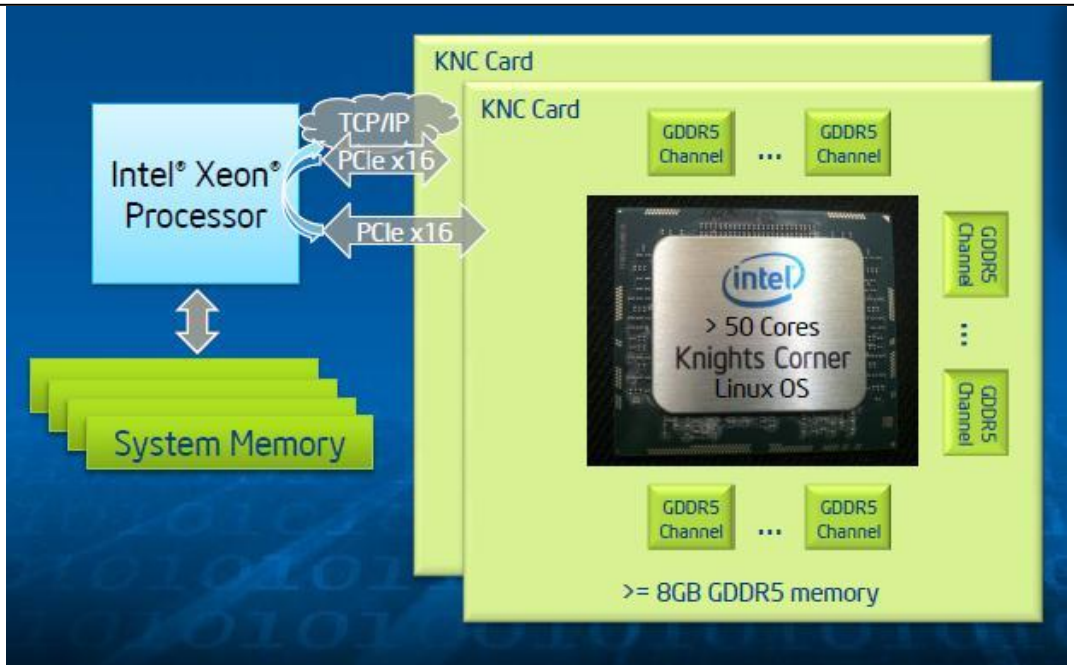

Рис. 1. Первое поколение процессоров Intel Xeon Phi под кодовым названием «Knights Corner»

Сопроцессор Intel Xeon Phi в основном состоит из процессорных ядер, кэша и контроллеров памяти, клиентской логики PCIe и двунаправленного интерконнекта с высокой пропускной способностью (см. рис. 2). Каждое ядро поставляется в комплекте с собственным кэшем второго уровня. Контроллеры памяти и логики PCIe-клиента обеспечивают прямой интерфейс с памятью GDDR5 на сопроцессоре и шиной PCIe. Все эти компоненты соединены друг с другом посредством кольцевого интерконнекта.

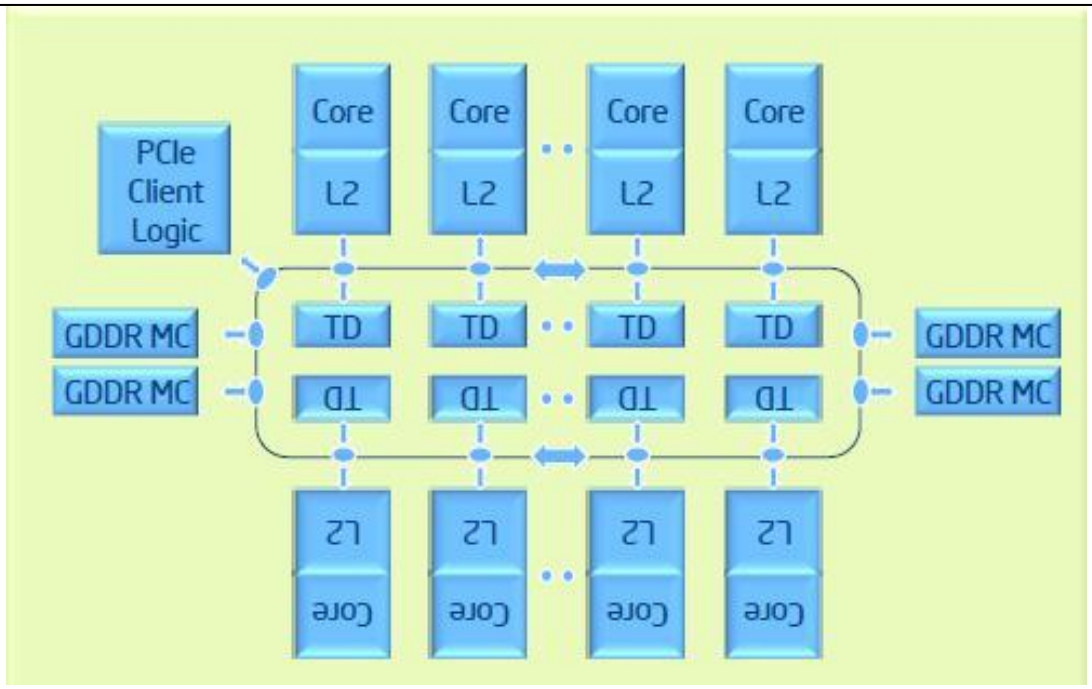

Рис. 2. Микроархитектура

Каждое ядро (см. рис. 3) в сопроцессоре Intel Xeon Phi разработано так, чтобы обеспечивать высокую пропускную способность во время параллельных вычислений. Каждое ядро способно поддерживать четыре потока на аппаратном уровне.

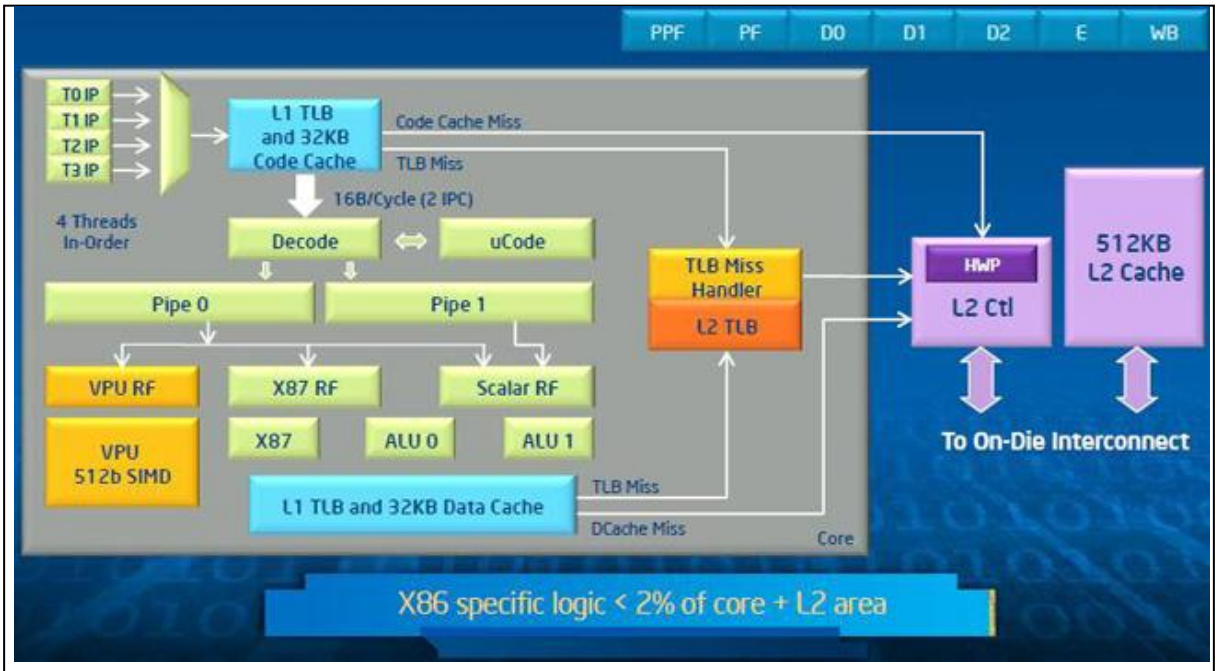

Рис. 3. Ядро сопроцессора Intel Xeon Phi

Благодаря сопроцессору Intel Xeon Phi реализация приложений максимальной производительности с высоким параллелизмом стала намного проще.

Основные характеристики сопроцессора Intel Xeon Phi SE10X:

1. микроархитектура Many Integrated Core (MIC), ядро Knights Corner;
2. четыре потока на ядро и до 61 ядра на сопроцессор (61 ядро/1,1 ГГц/244 потока);
3. дизайн x86 с поддержкой 64-разрядных инструкций;
4. 512-разрядные инструкции SIMD;
5. кэш-память 1-го уровня L1: 32 Кб данные и 32 Кб инструкции на ядро, латентность 1 такт;
6. кэш-память 2-го уровня L2: 512 Кб на ядро (до 30,5 Мб на сопроцессор), латентность 11 тактов;
7. поддержка Red Hat Enterprise Linux 6.x или SuSE Linux 12+, возможность IP-адресации;
8. использование трехмерных 22-нанометровых 3D-транзисторов Tri-Gate;
9. 8 Гб памяти GDDR5 на карту и полоса пропускания 320 Гб/с;
10. стандартный форм-фактор PCIe x16 (PCI Express x16).

Методы реализации поддержки работы T++-приложений на вычислительных кластерах с гибридными узлами на базе ускорителя Intel Xeon Phi

Для реализации поддержки работы T++-приложений на Intel MIC-ускорителях не требуется выполнять глубокую переработку ядра системы OpenTS. Достаточно адаптировать систему к сборке компилятором Intel C++ и при компиляции T++-приложений использовать специальные опции командной строки.

Схема сборки системы OpenTS для ускорителей Intel Xeon Phi отличается от сборки T-системы для host-узлов Intel Xeon: при сборке для ускорителей используется дополнительный набор ключей.

T++-приложения способны работать на Intel MIC-ускорителях при использовании библиотек Intel MPI и MPICH2. Для этого в настройках при сборке системы OpenTS указывается путь к каталогу, в который инсталлирована библиотека Intel MPI или MPICH2.

T++-приложения способны работать на MIC-ускорителях без использования MPI, за счет работы в режиме SMP. В этом режиме T++-приложение создает дополнительные системные потоки, которые задействуют вычислительные ресурсы всех процессоров SMP-системы (в данном случае всех процессоров MIC-ускорителя).

Следует учитывать следующую особенность: исполняемый код T++-приложения, полученный для host-узла, не может быть выполнен на ускорителе, и наоборот: код, полученный для ускорителя, не может быть запущен на host-узле.

Оптимизация работы T-приложений на ускорителе Intel Xeon Phi

Закрепление MPI-процесса за соответствующим логическим ядром процессора

В библиотеке Intel MPI [12] имеется возможность закрепить (прикрепить, pin) определенный MPI-процесс за соответствующим логическим процессором (logical CPU).

Номер логического процессора определяется как позиция соответствующего бита в битовой маске ядра (kernel affinity). Чтобы выяснить логические номера процессоров, можно воспользоваться командой 'cat /proc/cpuinfo' или утилитой 'cpuinfo', которая выдает информацию об архитектуре процессора.

Приведем описание соответствия логических ядер физическим ядрам на ускорителе Intel Xeon Phi.

Логическое ядро 0 привязано к (отображается на, соответствует) физическому ядру 60, контексту потока 0.

Логическое ядро 1 соответствует физическому ядру 0, контексту потока 0.
Логическое ядро 2 соответствует физическому ядру 0, контексту потока 1.
Логическое ядро 3 соответствует физическому ядру 0, контексту потока 2.
Логическое ядро 4 соответствует физическому ядру 0, контексту потока 3.
Логическое ядро 5 соответствует физическому ядру 1, контексту потока 0.
Логическое ядро 6 соответствует физическому ядру 1, контексту потока 1.
Логическое ядро 7 соответствует физическому ядру 1, контексту потока 2.
Логическое ядро 8 соответствует физическому ядру 1, контексту потока 3.
[...]
Логическое ядро 237 соответствует физическому ядру 59, контексту потока 0.
Логическое ядро 238 соответствует физическому ядру 59, контексту потока 1.
Логическое ядро 239 соответствует физическому ядру 59, контексту потока 2.
Логическое ядро 240 соответствует физическому ядру 59, контексту потока 3.
Логическое ядро 241 соответствует физическому ядру 60, контексту потока 1.

Логическое ядро 242 соответствует физическому ядру 60, контексту потока 2.

Логическое ядро 243 соответствует физическому ядру 60, контексту потока 3.

Следует отметить, что на ускорителе логическое ядро с номером 0 зарезервировано за операционной системой, поэтому не рекомендуется использовать его в расчетах.

Ускоритель Intel Xeon Phi имеет 61 физическое ядро [0..60] и содержит 244 логических ядра [0..243]. Используется технология Intel Hyper-Threading, которая позволяет выполнять одновременно до 4 потоков на каждом физическом ядре, что существенно повышает пропускную способность процессоров. Операционная система распознает каждое физическое ядро как 4 логических ядра, поэтому утилита 'cpuinfo' выдает информацию о наличии 244 ядер. При параллельных расчетах следует учитывать, что при использовании для MPI-процессов всех логических ядер производительность вычислений будет низкой, поскольку четыре потока Hyper-Threading будут делить между собой ресурсы одного физического ядра. Это будет происходить для всех физических ядер ускорителя.

Необходимо при помощи переменных окружения организовать вычисления так, чтобы один MPI-процесс приложения приходился на одно физическое ядро.

Трехуровневая иерархическая идентификация использует триплеты, которые предоставляют информацию о расположении процессора и его порядке. Триплеты иерархически упорядочены (пакет/сокет, ядро, потоки).

Переменные окружения

I_MPI_PIN

Включение/выключение закрепления процесса (process pinning). Эта переменная окружения устанавливается для выключения процесса закрепления для библиотеки Intel MPI Library.

Синтаксис:

I_MPI_PIN=<arg>

Аргумент <arg>	Описание
enable | yes | on | 1	Включить процесс закрепления. Это значение используется по умолчанию
disable | no | off | 0	Выключить процесс закрепления

I_MPI_PIN_PROCESSOR_LIST

Определяет подмножество процессоров и правила отображения для MPI-процессов (правила привязки MPI-процессов) в этом подмножестве.

Синтаксис:

I_MPI_PIN_PROCESSOR_LIST=<value>

Переменная окружения имеет следующие синтаксические формы:

1. <proclist>

[<procset>][:[grain=<grain>][,shift=<shift>]\[,preoffset=<preoffset>][,postoffset=<postoffset>]

[<procset>][:map=<map>]

Аргумент <value>	Описание
<proclist>	Список номеров логических процессоров или рангов процессоров. Процесс с i-м рангом закрепляется за i-м процессором в списке. Номер процессора не должен быть больше количества процессоров на узле
<l>	Процессор с логическим номером <l>
<l>-<m>	Диапазон процессоров с логическими номерами от <l> до <m>
<k>,<l>-<m>	Процессоры <k>, а также <l> по <m>
<procset>	Задает подмножество процессоров на основе топологической нумерации. Значение по умолчанию allcores

Аргумент <value>	Описание
all	Все логические процессоры. Это подмножество определяется как количество всех процессоров (CPUs) на узле
allcores	Все ядра (физические CPUs). Это подмножество определяется как число всех ядер на узле. Это значение по умолчанию
allsocks	Все пакеты/сокеты. Это подмножество определяет все сокеты на узле
<map>	Шаблон отображения (соответствия), используется для размещения процесса
bunch	Процессы отображаются (размещаются) как можно ближе на сокетах
scatter	Процессы отображаются (размещаются) удаленно насколько это возможно, чтобы не разделять общие ресурсы: FSB, кэш, ядро
spread	Процессы отображаются (размещаются) последовательно с возможностью не разделять общие ресурсы
<grain>	Задает гранулу ячейки для подмножества процессоров <procset>. Минимальным значением гранулы <grain> является единственный элемент в подмножестве <procset>. Максимальной гранулой является число <procset> элементов в сокете. Значение <grain> должно быть кратным значению <procset>. В противном случае предполагается минимальная гранула. Значением по умолчанию является минимальная гранула <grain>
<shift>	Определяет гранулярность кругового сдвига (размер сдвига) ячеек для подмножества <procset>. Значение <shift> измеряется в <grain>-блоках. Значение <shift> должно быть положительным целым числом. В противном случае сдвиг не выполняется. Значением по умолчанию является значение «сдвига нет»
<preoffset>	Задает циклический сдвиг подмножества процессоров на величину <preoffset> перед выполнением кругового сдвига. Значение <preoffset> измеряется в <grain>-блоках. Значение <preoffset> должно быть неотрицательным целым числом. В противном случае сдвиг не выполняется. Значением по умолчанию является значение «сдвига нет»
<postoffset>	Задает циклический сдвиг подмножества процессоров на величину <postoffset> после выполнения кругового сдвига. Значение <postoffset> измеряется в <grain>-блоках. Значение <postoffset> должно быть неотрицательным целым числом. В противном случае сдвиг не выполняется. Значением по умолчанию является значение «сдвига нет»
<n>	Задает явное значение соответствующих параметров, ранее упоминавшихся. <n> является неотрицательным целым числом
fine	Задает минимальное значение соответствующего параметра

Аргумент <value>	Описание
core	Задает значение, равное количеству единиц соответствующего параметра, содержащихся в одном ядре
cache1	Задает значение, равное количеству единиц соответствующего параметра для кэша L1
cache2	Задает значение, равное количеству единиц соответствующего параметра для кэша L2
cache3	Задает значение, равное количеству единиц соответствующего параметра для кэша L3
cache	Наибольшее значение среди cache1, cache2 и cache3
socket | sock	Задает значение, равное количеству единиц соответствующего параметра, содержащегося в одном физическом пакете/сокете
half | mid	Задает значение параметра, равное значению socket/2
third	Задает значение параметра, равное значению socket/3
quarter	Задает значение параметра, равное значению socket/4
octavo	Задает значение параметра, равное значению socket/8

Переменная окружения I_MPI_PIN_PROCESSOR_LIST действительна только в случае, если определена переменная окружения I_MPI_PIN.

Переменная окружения I_MPI_PIN_PROCESSOR_LIST устанавливается для определения размещения процессоров. Чтобы избежать конфликтов с различными версиями shell-оболочки, значение переменной среды необходимо заключать в кавычки.

Переменная окружения I_MPI_PIN_PROCESSOR_LIST имеет следующие различные варианты синтаксиса.

- Явный список процессоров. Это разделенный запятыми список определяется в терминах логических номеров процессоров. Относительный ранг процесса на узле является индексом в списке процессоров так, что *i*-й процесс закрепляется за *i*-м элементом (номером) списка. Это позволяет определить размещение любого процесса на *логическом процессоре* (CPU). Например, процесс отображения для I_MPI_PIN_PROCESSOR_LIST = *p*0, *p*1, *p*2, ..., *pn* выглядит следующим образом:

Ранг на узле (Rank on a node)	0	1	2	...	*n*-1	*N*
Логическое ядро (Logical CPU)	*p*0	*p*1	*p*2	...	*pn*-1	*Pn*

- grain/shift/offset-отображение. Этот метод обеспечивает циклический сдвиг определенной гранулы grain по списку процессоров с шагами, равными shift*grain, и один сдвиг на offset*grain в конце. Действие по смещению (сдвигу) повторяетя shift раз.

Например: grain = 2 логических процессора (logical processors), shift = 3 grains, offset = 0.

Чтобы прикрепить MPI-процессы к логическим ядрам CPU0 и CPU3 на каждом узле в глобальном масштабе, используется следующая команда:

$ mpirun -genv I_MPI_PIN_PROCESSOR_LIST='0,3' -n <# of processes> <executable>

Чтобы закрепить MPI-процессы за различными логическими ядрами CPUs на каждом узле индивидуально (CPU0 и CPU3 на host1 и CPU0, CPU1 и CPU3 на host2), используется следующая команда:

$ mpirun -host host1 -env I_MPI_PIN_PROCESSOR_LIST='0,3' -n <# of processes> <executable> : -host host2 -env I_MPI_PIN_PROCESSOR_LIST='1,2,3' -n <# of processes> <executable>

Чтобы напечатать дополнительную отладочную информацию о процессе закрепления, используется переменная окружения I_MPI_DEBUG с параметром 4. Например:

$ mpirun -genv I_MPI_DEBUG=4 -m -host host1 -env I_MPI_PIN_PROCESSOR_LIST='0,3' -n <# of processes> <executable> : -host host2 -env I_MPI_PIN_PROCESSOR_LIST='1,2,3' -n <# of processes> <executable>

I_MPI_PIN_DOMAIN

Библиотека Intel MPI Library предоставляет дополнительную переменную окружения для управления процессом закрепления для гибридных приложений. Эта переменная окружения используется для определения целого ряда неперекрывающихся (непересекающихся) подмножеств (доменов), состоящих из логических процессоров на узле, а также набор правил о том, как MPI-процессы связаны с этими доменами, по следующей формуле: один MPI-процесс на один домен. Например: имеем два домена – Domain0={1,2,3,4} и Domain2={5,6,7,8}. В фигурных скобках указаны логические процессоры (ядра). Тогда к домену Domain0 можно прикрепить MPI-процесс с рангом 0 (rank0), а к домену Domain2 прикрепить другой MPI-процесс с рангом 1 (rank1). Но домены {1,2,3,4}, {4,5,6,7} не могут быть созданы в такой конфигурации, так как они должны быть непересекающимися (здесь логический процессор 4 входит в оба домена). Максимальное количество доменов, которое может быть создано, равно количеству логических процессоров на ускорителе.

Каждый MPI-процесс может создать ряд дочерних потоков, которые будут выполняться в пределах соответствующего домена. Потоки (нити) могут свободно мигрировать из одного логического процессора в другой в пределах конкретного домена. Если переменная окружения I_MPI_PIN_DOMAIN определена, то переменная I_MPI_PIN_PROCESSOR_LIST игнорируется. Если переменная I_MPI_PIN_DOMAIN не определена, то MPI-процессы закрепляются за логическими процессорами в соответствии с текущим значением переменной I_MPI_PIN_PROCESSOR_LIST.

Переменная I_MPI_PIN_DOMAIN имеет следующие синтаксические формы:

- описание домена в терминах мультиядра (multi-core);
- описание домена через указание размера и расположения домена;
- явное описание домена посредством битовой маски.

Следующие таблицы описывают эти синтаксические формы.

Многоядерные форматы

I_MPI_PIN_DOMAIN=<mc-shape>

<mc-shape>	Определяет домены в терминах multi-core
core	Каждый домен состоит из логических процессоров, которые разделяют конкретное ядро. Число доменов на узле равно количеству ядер на нем
socket | sock	Каждый домен состоит из логических процессоров, которые разделяют конкретный сокет. Число доменов на узле равно количеству сокетов на узле. Это рекомендуемое значение
node	Все логические процессоры на узле располагаются в одном домене
cache1	Логические процессоры, которые разделяют конкретный кэш 1-го уровня, располагаются в одном домене
cache2	Логические процессоры, которые разделяют конкретный кэш 2-го уровня, располагаются в одном домене
cache3	Логические процессоры, которые разделяют конкретный кэш 3-го уровня, располагаются в одном домене
cache	Выбирается самый большой среди доменов cache1, cache2 и cache3

Явные форматы
I_MPI_PIN_DOMAIN=<size>[:<layout>]

<size>	Определяет число логических процессоров в каждом домене (domain size)
omp	Размер домена равен значению переменной окружения OMP_NUM_THREADS. Если переменная OMP_NUM_THREADS не установлена, каждый узел обрабатывается как отдельный домен
auto	Размер домена определяется по формуле size=#cpu / #proc, где #cpu – число логических процессоров на узле, #proc – число MPI-процессов, стартовавших на узле
<n>	Размер домена определяется как десятичное целое число <n>
<layout>	Упорядочение элементов домена. Значение по умолчанию – compact
platform	Элементы домена упорядочиваются в соответствии с их нумерацией в BIOS. Это нумерация, зависящая от платформы (platform-dependent numbering)
compact	Элементы домена располагаются как можно ближе друг к другу, насколько это возможно в терминах общих ресурсов (ядра, кэш, сокеты и т.д.). Это значение по умолчанию
scatter	Элементы домена располагаются как можно дальше друг от друга, насколько это возможно в терминах общих ресурсов (ядра, кэш, сокеты и т.д.)

Явная маска домена
I_MPI_PIN_DOMAIN=<masklist>

<masklist>	Определение доменов через список шестнадцатеричных чисел, разделенных запятыми (маски домена)
[m1,...,mn]	Каждое число mi определяет один отдельный домен. Используется следующее правило: *i*-й логический процессор включается в домен, если соответствующее значение mi установлено в 1. Все остальные процессоры помещаются в другой отдельный домен. Используется BIOS -нумерация

Если не нужно, чтобы процесс мигрировал между сокетами на multi-socket-платформе, следует задать размер домена как I_MPI_PIN_DOMAIN=socket или меньше.

Некоторые эксперименты с T++-приложением на гибридном кластере

В качестве T++-приложения была выбрана задача «Вычисление числа π методом Монте-Карло». Существует много способов вычисления числа π. Самым простым и понятным является численный метод Монте-Карло. Об алгоритме и MPI-версии приложения можно узнать из [13].

Алгоритм вычисления числа π на псевдо коде для T++-приложения выглядит так:

```
Главная T-функция генерирует n случайных начальных чисел (random seed)
For rank i=0 to n-1
 Для каждой вспомогательной T-функции на ранге i выполнить
```

```
    получить соответствующее начальное случайное число (corresponding random seed)
    установить num_inside = 0
    For j=0 to Nc / n
    сгенерировать точку с координатами x, y, z:
    x в пределах [i/n, (i+1)/n]
    y в пределах [0, 1]
    z в пределах [0, 1]
    вычислить расстояние d = x^2 + y^2 + z^2
    if расстояние d <= 1, увеличить num_inside
    End For
  Конец работы вспомогательной T-функции на ранке i
  End For
  Главная T-функция ожидает готовности всех значений num_inside от вспомогательных
T-функций
  Главная T-функция вычисляет Ns как сумму всех значений num_inside
  Главная T-функция вычисляет Pi = 6 * Ns / Nc
```

Эксперименты проводились на кластере «Торнадо ЮУрГУ», построенном на базе процессоров Intel Xeon X5680 и ускорителей (сопроцессоров) Intel Xeon Phi SE10X.

На кластере используется планировщик задач SLURM [14]. В пакетном режиме пользователь создает сценарий для запуска задачи и использует утилиту «`sbatch`». Задача отправляется в очередь и будет выполнена, как только станут доступными запрошенные ресурсы. Вывод результата выполнения задачи будет записан в файл `slurm-<номер_задачи_в_очереди>.out` в директории, откуда осуществлялся запуск задачи. В интерактивном режиме пользователь либо использует для запуска задачи утилиту «`srun`», либо самостоятельно выделяет необходимое количество *вычислительных узлов* (ВУ) и запускает задачу с помощью утилиты `mpiexec.hydra` или `srun`.

Выполним сборку T++-приложения для ускорителя Intel Xeon Phi с опцией «`-mmic`»:

```
$ t++ -opt -mmic pi_monte_carlo.tcc -o pi_monte_carlo.mic
```

Сборка приложения для выполнения на host-узле выглядит так:

```
$ t++ -opt pi_monte_carlo.tcc -o pi_monte_carlo.host
```

Приведем примеры запусков T++-приложения в интерактивном режиме.

1) Используем 8 MPI-процессов на ускорителе.

```
$ salloc -N 1 --gres=mic:1
$ echo $SLURM_NODELIST
node031
$ ssh node031-mic0
```

Далее запускаем наше T++-приложение командой `mpiexec.hydra`:

```
$ mpiexec.hydra -perhost 8 ./pi_monte_carlo.mic
```

Результат выполнения данной команды показан на рисунке 4.

2) Запустим теперь T++-приложение на ускорителе в режиме SMP. Для этого используем команду

```
$ DMPI='smp 8' ./pi_monte_carlo.mic
```

Результат выполнения данной команды показан на рисунке 5.

3) Запустим T++-приложение на host-узлах:

```
$ salloc -N 2
$ mpiexec.hydra  -n 16 -host node195,node204 ./pi_monte_carlo.host
```

Результат выполнения данной команды показан на рисунке 6.

4) Запустим T++-приложение на ускорителе:

```
$  mpiexec.hydra  -genv  I_MPI_DEBUG=4  -genv  I_MPI_PIN_DOMAIN=core  -n  60
~/pi_monte_carlo.mic
```

Используем в командной строке переменные окружения I_MPI_DEBUG и I_MPI_PIN_DOMAIN. Переменная I_MPI_DEBUG выводит отладочную информацию о привязке каждого ранга MPI-процесса к соответствующему домену логических ядер. Переменная окружения I_MPI_PIN_DOMAIN задает способ создания домена – «core».

В этом случае привязка MPI-процессов к доменам происходит по следующей схеме:

rank0->{1,2,3,4}, rank1->{5,6,7,8}, rank2->{9,10,11,12}, …, rank59->{237,238,239,240}.

Результат выполнения данной команды показан на рисунке 7. Из рисунка была исключена некоторая отладочная печать.

5) Еще один вариант запуска T++-приложения на ускорителе. Используем переменные окружения I_MPI_DEBUG и I_MPI_PIN_PROCESSOR_LIST:

```
$  mpiexec.hydra  -genv  I_MPI_DEBUG=4  -genv
I_MPI_PIN_PROCESSOR_LIST='grain=cache2,shift=sock' -n 60 ./pi_monte_carlo.mic
```

Результат выполнения данной команды показан на рисунке 8. Из рисунка была исключена некоторая отладочная печать.

Обратите внимание на время счета T++-приложения в данной конфигурации: оно почти в два с половиной раза больше, чем в примере предыдущего запуска.

Это связано с распределением MPI-процессов по логическим ядрам ускорителя. Данная конфигурация неоптимальна, так как привязывает каждый MPI-процесс [0..59] к логическим ядрам [1..60] в следующем виде:

rank0->CPU1, rank1->CPU2, rank2->CPU3, rank3->CPU4, ..., rank59->CPU60

Вывод: MPI-процессы делят между собой ресурсы физического ядра. Отсюда имеем существенное замедление процесса счета.

```
Open T-System runtime v3.2, 2003-2012, PSI RAS, Russia.
Running under Intel MPI MPI on 8-rank cluster:
  ([146.0Gf,2189BM,0.00GiB]*8) ~= [1168.4Gf,17512BM,0.00GiB]
Starting tfun main, good luck!

Rank 6/8 at host `node031-mic0' counts 142733300 points inside the sphere
Rank 5/8 at host `node031-mic0' counts 221809952 points inside the sphere
Rank 0/8 at host `node031-mic0' counts 419475718 points inside the sphere
Rank 1/8 at host `node031-mic0' counts 406286873 points inside the sphere
Rank 4/8 at host `node031-mic0' counts 287686277 points inside the sphere
Rank 2/8 at host `node031-mic0' counts 379929343 points inside the sphere
Rank 3/8 at host `node031-mic0' counts 340382912 points inside the sphere
Out of 4294967295 points, there are 2248816144 points inside the sphere =>
pi=  3.141559839249
Tasks activated:      [1/1/2]
Tasks exported:       [0/0/7]
Msgs sent:            [25/30/53]
Async Msgs:           [0/0/0]
Msgs size:            [2800/3758/9152]
Taskboard visits:     [3481249/3535269/3742829]
Scheduler time:       [6.667/13.168/15.184]
MPI time:             [0.000/0.000/0.001]
Idle time:            [268.138/269.100/271.636]
Tasks time:           [268.604/271.139/272.102]
Total time:           [540.240/540.240/540.241]
```

Рис. 4. Результат выполнения T-приложения командой mpiexec.hydra на ускорителе Intel Xeon Phi

```
Portal 7 is online
Rank 6/8 at host `node004-mic0' counts 142754458 points inside the sphere
Rank 5/8 at host `node004-mic0' counts 221816238 points inside the sphere
Rank 0/8 at host `node004-mic0' counts 419468677 points inside the sphere
Task 7 has been removed from portal
Task 8 has been removed from portal
Task 0x2ee5b28(taskId=9) is inserted to portal
Task 0x2ee5b28(taskId=9) is inserted to portal
Rank 1/8 at host `node004-mic0' counts 406282514 points inside the sphere
Task 3 has been removed from portal
Rank 4/8 at host `node004-mic0' counts 287682135 points inside the sphere
Task 6 has been removed from portal
Rank 2/8 at host `node004-mic0' counts 379947794 points inside the sphere
Task 4 has been removed from portal
Rank 3/8 at host `node004-mic0' counts 340408316 points inside the sphere
Task 5 has been removed from portal
Rank 7/8 at host `node004-mic0' counts 50514687 points inside the sphere
Task 9 has been removed from portal
Out of 4294967295 points, there are 2248874819 points inside the sphere => pi=
3.141641616821
Total time: 543 sec.
Sorry, no statistics in fault-tolerant mode.
~ $ Destroying fd=4 (rank=1)
Configuration has changed (generation=9).
Portal 1 is offline
Destroying fd=5 (rank=2)
```

Рис. 5. Результат выполнения T-приложения в режиме SMP на ускорителе Intel Xeon Phi

```
Open T-System runtime v3.2, 2003-2012, PSI RAS, Russia.
Running under INTELMPI MPI on 16-rank cluster:
  ([662.8Gf,6649BM,0.00GiB]*16) ~= [10604.7Gf,106384BM,0.00GiB]
Starting tfun main, good luck!

Rank 14/16 at host `node204' counts 37615391 points inside the sphere
Rank 12/16 at host `node204' counts 82090882 points inside the sphere
Rank 13/16 at host `node204' counts 60672765 points inside the sphere
Rank 9/16 at host `node195' counts 136439831 points inside the sphere
...
Rank 10/16 at host `node195' counts 119950329 points inside the sphere
Rank 6/16 at host `node195' counts 175960741 points inside the sphere
Rank 15/16 at host `node204' counts 12904081 points inside the sphere
Out of 4294967295 points, there are 2248899272 points inside the sphere =>
pi=  3.141675949097
Tasks activated:      [1/1/2]
Tasks exported:       [0/0/15]
Msgs sent:            [34/42/96]
Async Msgs:           [0/0/0]
Msgs size:            [3816/5162/17656]
Taskboard visits:     [1388660/1597015/2195657]
Scheduler time:       [0.251/0.609/1.039]
MPI time:             [0.000/0.000/0.000]
Idle time:            [11.959/16.272/25.981]
Tasks time:           [12.405/16.415/25.853]
Total time:           [24.393/32.688/50.937]
```

Рис. 6. Результат выполнения T-приложения на двух host-узлах

```
Open T-System runtime v3.2, 2003-2012, PSI RAS, Russia.
Running under Intel MPI MPI on 60-rank cluster:
  ([146.1Gf,2189BM,0.00GiB]*60) ~= [8763.7Gf,131340BM,0.00GiB]
Starting tfun main, good luck!

Rank 1/60 at host `node048-mic0' counts 56185061 points inside the sphere
...
Rank 27/60 at host `node048-mic0' counts 44411231 points inside the sphere
Rank 46/60 at host `node048-mic0' counts 22455052 points inside the sphere
Rank 59/60 at host `node048-mic0' counts 932476 points inside the sphere
Out of 4294967295 points, there are 2248825101 points inside the sphere =>
pi=  3.141572475433
Tasks activated:      [1/1/2]
Tasks exported:       [0/0/59]
Msgs sent:            [128/133/316]
Async Msgs:           [0/0/0]
Msgs size:            [14328/15361/62712]
Taskboard visits:     [57661/69453/87371]
Scheduler time:       [0.131/0.294/0.367]
MPI time:             [0.001/0.002/0.003]
Idle time:            [35.404/35.895/36.295]
Tasks time:           [35.759/36.158/36.650]
Total time:           [72.058/72.059/72.060]
```

Рис. 7. Результат выполнения T-приложения на ускорителе (I_MPI_PIN_DOMAIN=core)

Таким образом, изменяя конфигурацию и манипулируя опциями командной строки, можно оптимизировать работу T++-приложений.

Таким образом, в результате исследований выработаны и реализованы на практике методы адаптации системы OpenTS поддержки исполнения параллельных T++-приложений к эффективной работе на гибридных суперЭВМ.

```
Open T-System runtime v3.2, 2003-2012, PSI RAS, Russia.
Running under Intel MPI MPI on 60-rank cluster:
  ([61.0Gf,2189BM,0.00GiB]*60) ~= [3660.5Gf,131340BM,0.00GiB]
Starting tfun main, good luck!

Rank 35/60 at host `node048-mic0' counts 36535649 points inside the sphere
Rank 1/60 at host `node048-mic0' counts 56186057 points inside the sphere
...
Rank 34/60 at host `node048-mic0' counts 37634936 points inside the sphere
Rank 59/60 at host `node048-mic0' counts 931698 points inside the sphere
Out of 4294967295 points, there are 2248817946 points inside the sphere =>
pi=  3.141562461853
Tasks activated:      [1/1/2]
Tasks exported:       [0/0/59]
Msgs sent:            [128/133/316]
Async Msgs:           [0/0/0]
Msgs size:            [14328/15361/62712]
Taskboard visits:     [17313/31666/89017]
Scheduler time:       [0.243/0.476/1.394]
MPI time:             [0.004/0.005/0.011]
Idle time:            [52.074/75.095/129.238]
Tasks time:           [56.115/110.258/133.277]
Total time:           [185.370/185.372/185.374]
```

Рис. 8. Результат выполнения T-приложения на ускорителе (`I_MPI_PIN_PROCESSOR_LIST='grain=cache2,shift=sock'`)

Результаты проведенных исследований и выполненных экспериментальных разработок позволяют рассчитывать на хорошие показатели эффективности при счете научно-прикладных T++-приложений на гибридных вычислительных кластерах с узлами, содержащими ускорители Intel Xeon Phi.

Работы, положенные в основу данной статьи, были выполнены в рамках Программы фундаментальных научных исследований ОНИТ РАН «Архитектурно-программные решения и обеспечение безопасности суперкомпьютерных информационно-вычислительных комплексов новых поколений», а также в рамках НИР «Методы и программные средства разработки параллельных приложений и обеспечения функционирования вычислительных комплексов и сетей нового поколения».

Литература

1. Абрамов С.М., Васенин В.А., Мамчиц Е.Е., Роганов В.А., Слепухин А.Ф. Динамическое распараллеливание программ на базе параллельной редукции графов. Архитектура программного обеспечения новой версии T-системы / Научная сессия МИФИ-2001: сб. науч. тр. М., 2001. Т. 2.
2. Абрамов С.М., Кузнецов А.А., Роганов В.А. Кроссплатформенная версия T-системы с открытой архитектурой / Параллельные вычислительные технологии (ПаВТ'2007): тр. Междунар. науч. конф. (29 января–2 февраля 2007 г., Челябинск). Челябинск: Изд-во ЮУрГУ, 2007. Т. 1. С. 115–121.
3. Абрамов С.М., Кузнецов А.А., Роганов В.А. Кроссплатформенная версия T-системы с открытой архитектурой // Вычислительные методы и программирование. 2007. Т. 8. № 1. С. 175–180; URL: http://num-meth.srcc.msu.su/ (дата обращения: 05.06.14).
4. Кузнецов А.А., Роганов В.А. Экспериментальная реализация отказоустойчивой версии системы OpenTS для платформы Windows CCS // Суперкомпьютерные системы и их применение: тр. II Междунар. науч. конф. (SSA'2008) (27–29 октября 2008, г. Минск). Минск: Изд-во ОИПИ НАН Беларуси, 2008. С. 65–70.
5. Степанов Е.А. Планирование в OpenTS-системе автоматического динамического распараллеливания // Информационные технологии и программирование: сб. статей. М.: Изд-во МГИУ, 2005. Вып. 2.
6. Абрамов С.М., Есин Г.И., Загоровский И.М., Матвеев Г.А., Роганов В.А. Принципы организации отказоустойчивых параллельных вычислений для решения вычислительных задач и задач управления в T-системе с открытой архитектурой (OpenTS) // Программные системы: теория и приложения (PSTA-2006): тр. Междунар. конф. (23–28 октября 2006 г., г. Переславль-Залесский). Переславль-Залесский: Изд-во ИПС РАН. С. 257–264.

7. Roganov V. and Slepuhin A. Distributed Extension of the Parallel Graph Reduction. GRACE: Compact and Efficient Dynamic Parallelization Technology for the Heterogeneous Computing Systems // International Conference on Parallel and Distributed Processing Techniques and Applications, June 25–28, 2001, Las Vegas, Nevada, USA.

8. Moskovsky A., Roganov V., and Abramov S. Parallelism Granules Aggregation with the T-System. Parallel Cmputing Technologies: 9th International Conference, PaCT 2007 Pereslavl-Zalessky, Russia, September 2007. Proceedings. Victor Malyshkin (Ed.). Berlin etc., Springer, 2007. Lecture Notes in Computer Science: Vol. 4671, pp. 293–302.

9. Moskovsky A., Roganov V., Abramov S., and Kuznetsov A. Variable Reassignment in the T++ Parallel Programming Language. Parallel Computing Technologies: 9th International Conference, PaCT 2007 Pereslavl-Zalessky, Russia, September 2007. Proceedings. Victor Malyshkin (Ed.). Berlin etc. Springer, 2007. Lecture Notes in Computer Science: Vol. 4671, pp. 579–588.

10. Архитектура Intel Many Integrated Core (Intel MIC) – расширенные возможности. URL: http://www.intel.ru/content/www/ru/ru/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html (дата обращения: 05.06.2014).

11. Семейство продукции Intel Xeon Phi. URL: http://www.intel.ru/content/www/ru/ru/processors/xeon/xeon-phi-detail.html (дата обращения: 05.06.2014).

12. Intel MPI Library Reference Manual for Linux. URL: http://download-software.intel.com/sites/products/documentation/hpc/ics/impi/41/lin/Reference_Manual/Reference_Manual.pdf (дата обращения: 05.06.2014).

13. Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems. URL: https://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems (дата обращения: 05.06.2014).

14. Планировщик задач SLURM. URL: https://computing.llnl.gov/linux/slurm/documentation.html (дата обращения: 05.06.2014).