Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Программирование для Intel Xeon Phi

Архитектура Intel Xeon Phi

Линёв А.В.

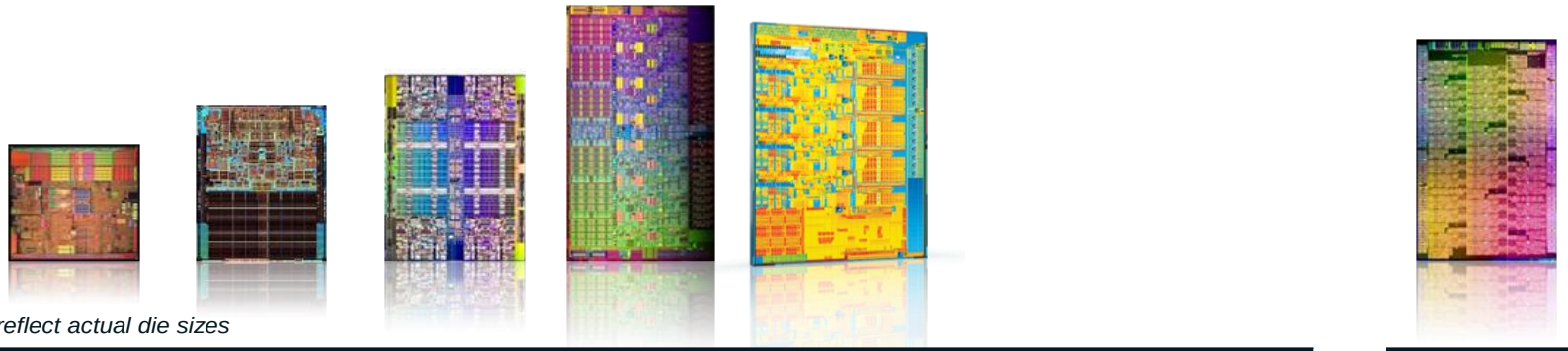
2014

Архангельск

Содержание

Введение

More cores. Wider vectors. Co-Processors.



Images do not reflect actual die sizes

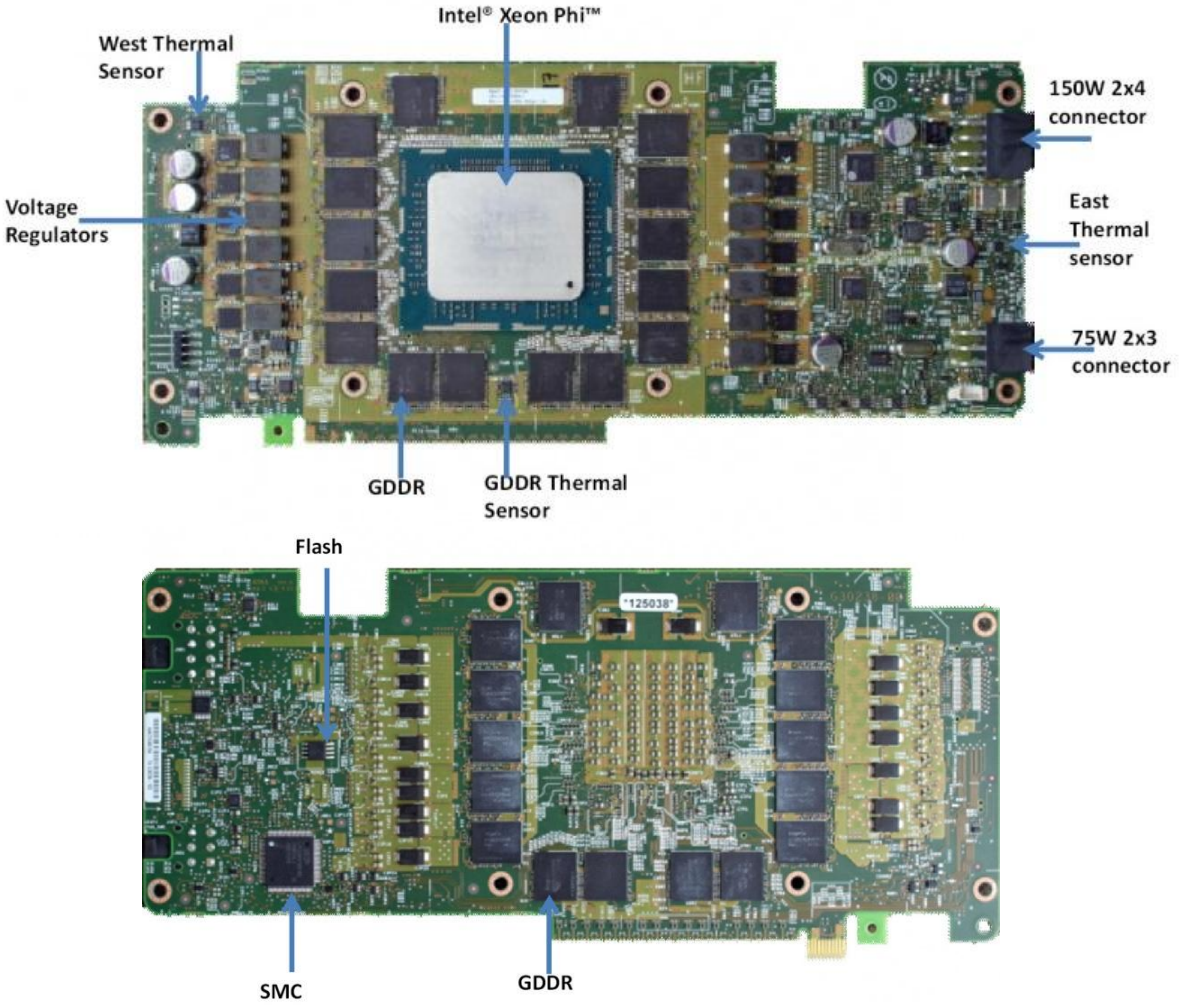
	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor code-named Sandy Bridge	Intel® Xeon® processor code-named Ivy Bridge	Intel® Xeon® processor code-named Haswell	Intel® Xeon Phi co-processor Knights Corner
Core(s)	1	2	4	6	8			60
Threads	2	2	8	12	16			240
SIMD Width	128	128	128	128	256	256	256	512
	SSE2	SSSE3	SSE4.2	SSE4.2	AVX	AVX	AVX2 FMA3 TSX	

Геннадий Федоров. Intel® Xeon Phi. Курс “молодого” бойца.

Сопроцессор Intel Xeon Phi

- В конце 2012 года Intel представила первый процессор с архитектурой Intel MIC (Intel® Many Integrated Core Architecture)
- Основой архитектуры MIC является использование большого количества вычислительных ядер архитектуры x86 в одном процессоре
- Для разработки параллельных программ могут быть использованы стандартные технологии: pthreads, OpenMP, Intel TBB, Intel Cilk Plus, MPI
- Intel® Xeon Phi™ 5110P подключается к разъему PCIe x16 на материнской плате, в двухпроцессорной системе можно установить до 8 карт Intel® Xeon Phi™ 5110P

Сопроцессор Intel Xeon Phi



Intel® Xeon Phi™ Coprocessor: Datasheet [http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-datasheet.html]

Что должен знать разработчик HPC-приложений…

- Программа, с которой обычно начинается изучение новой технологии параллельного программирования

```
double A[N], B[N], C;
int i;
C = 0.0;
for(i = 0; i < N; i ++){
  C += A[i] * B[i];
}
```

- Знание системных основ параллельных вычислений позволяет понимать, как должна/будет работать такая программа на аппаратном уровне

Что должен знать разработчик HPC-приложений

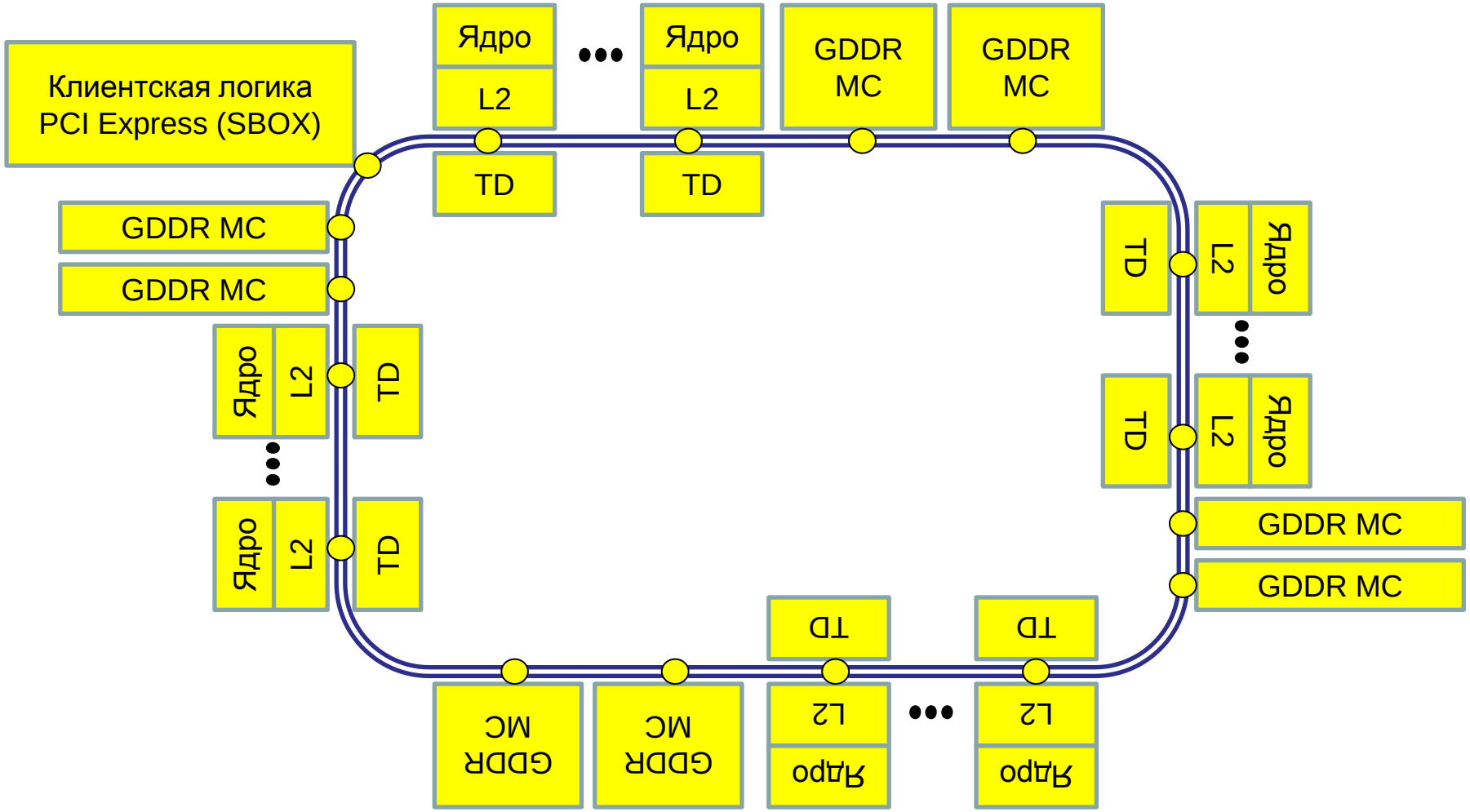
- Предварительные требования
  - Основы программирования
  - Алгоритмы и анализ сложности
  - Языки программирования
- Системные основы параллельных вычислений
  - **Архитектура вычислительных систем** (+ Ассемблер)
  - Компьютерные сети
  - Операционные системы
  - Компиляторы
- Использование параллельных вычислений
  - Параллельное программирование / Алгоритмы / Языки / Технологии / Инструменты / …

Архитектура вычислительных систем…

- Введение
- **Архитектура процессора, компоненты CPU**
- **Конвейеризация вычислений (статическое и динамическое планирование)**
- **Векторные вычисления**
- **Иерархия памяти**
- Классификация архитектур вычислительных систем
  - Симметричное мультипроцессирование
  - Массивно-параллельные системы, кластерные системы
  - Параллелизм в процессорах специального назначения
- Примеры вычислительных систем

Архитектура Intel Xeon Phi

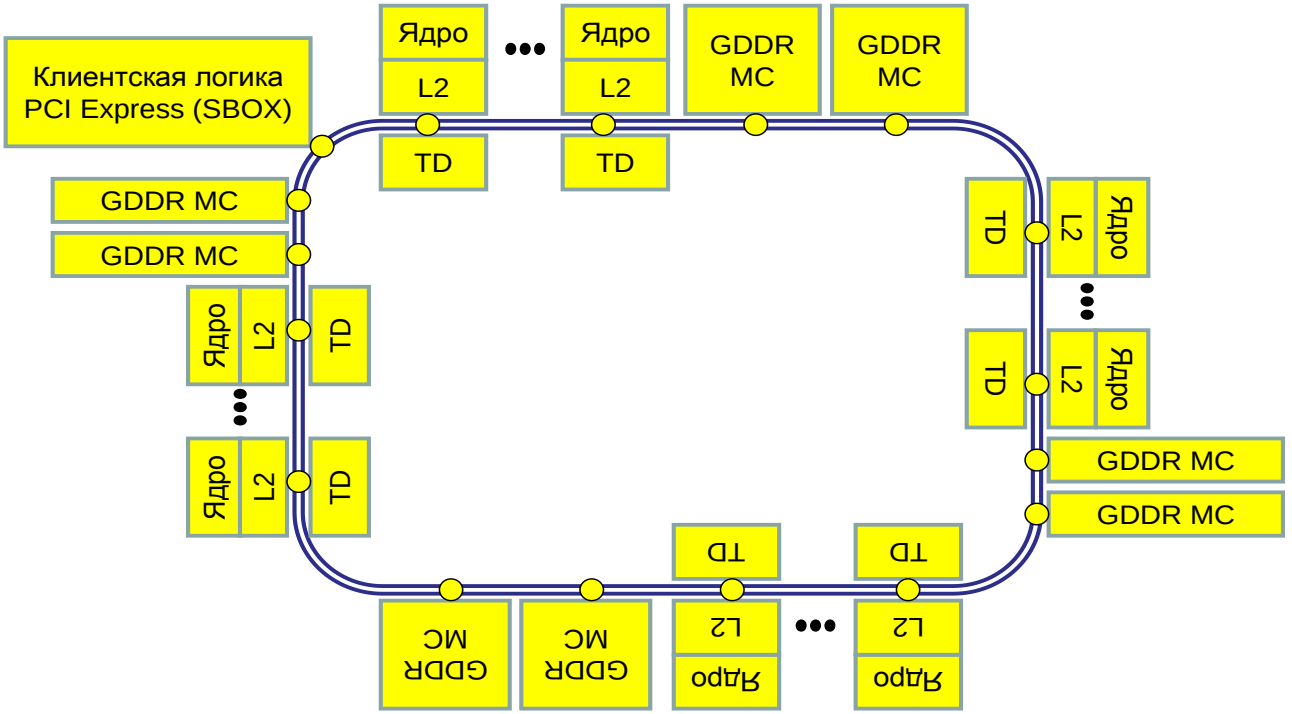
Архитектура Intel Xeon Phi



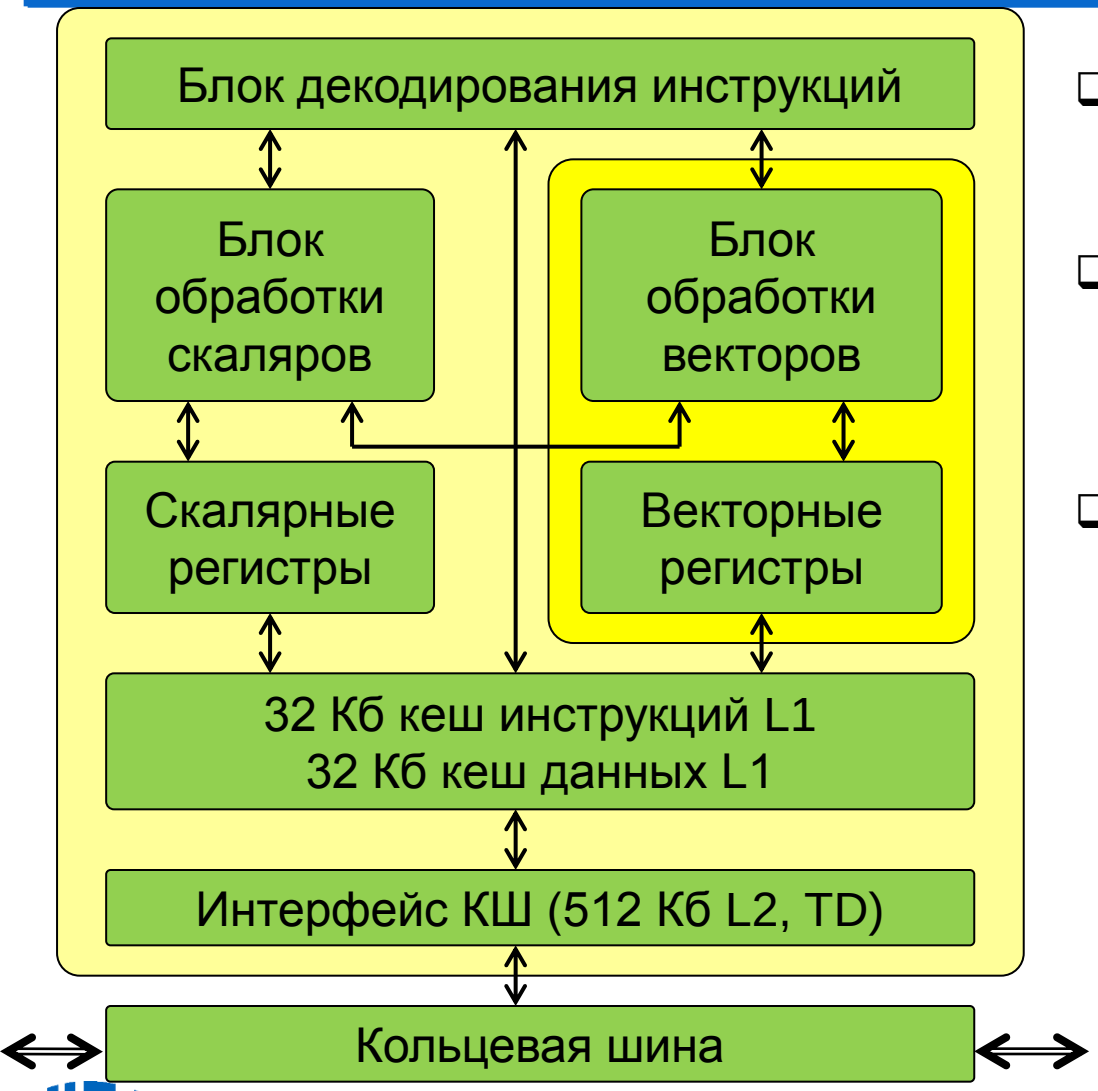
TD – Каталог тегов (Tag Directory), GDDR MC – Контроллер памяти

Архитектура Intel Xeon Phi

- До 61 процессорных ядер
- Высокопроизводительная кольцевая шина
- 8 контроллеров памяти обслуживают 16 каналов GDDR5
- Отдельный компонент реализует клиентскую логику PCI Express



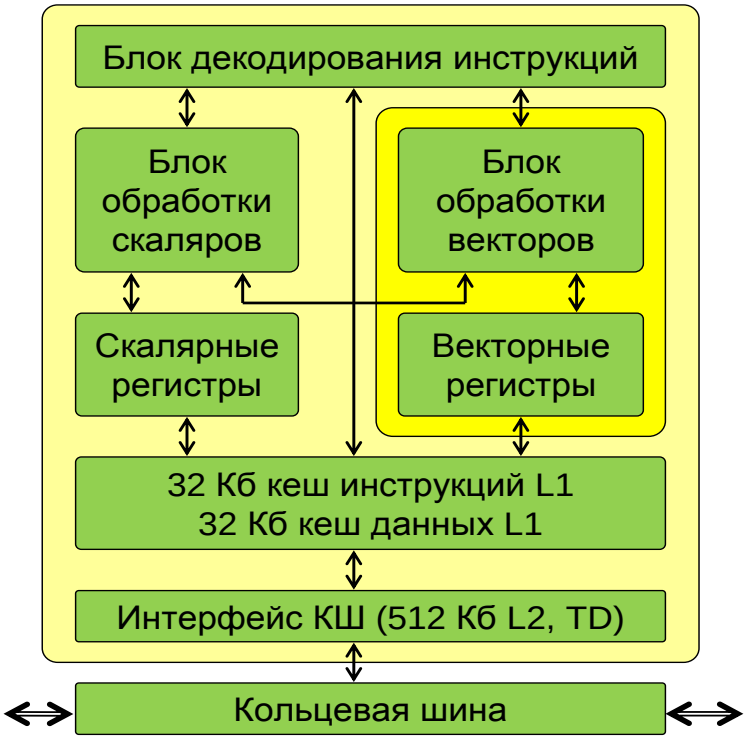
Исполнительное ядро Intel Xeon Phi…



- Выполняет выборку и декодирование инструкций 4 аппаратных потоков
- Поддерживает выполнение 32- и 64-битного кода, совместимого с архитектурой Intel64
- Ядро содержит 2 конвейера (U-конвейер и V-конвейер) и может выполнять 2 инструкции за такт
  - V-конвейер способен выполнять не все типы инструкций, возможность параллельного выполнения команд на U- и V-конвейерах задается набором правил

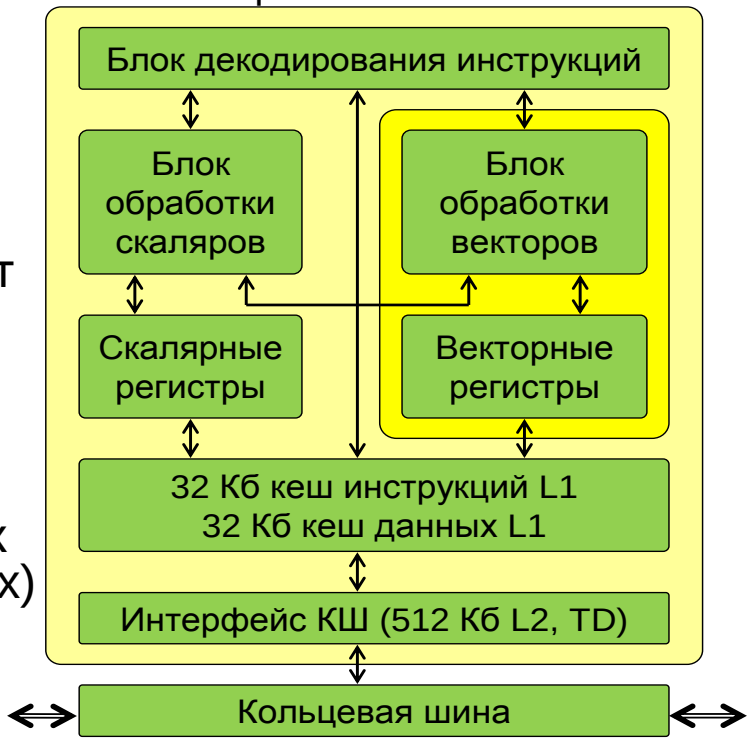
Исполнительное ядро Intel Xeon Phi

- Внеочередное выполнение инструкций не поддерживается
- Не реализованы команды Intel Streaming SIMD Extensions (SSE), MMX и Advanced Vector Extensions (AVX)
- Включает по 32 Кб 8-канальных множественно-ассоциативных кешей инструкций и данных (L1 I-Cache и L1 D-Cache)



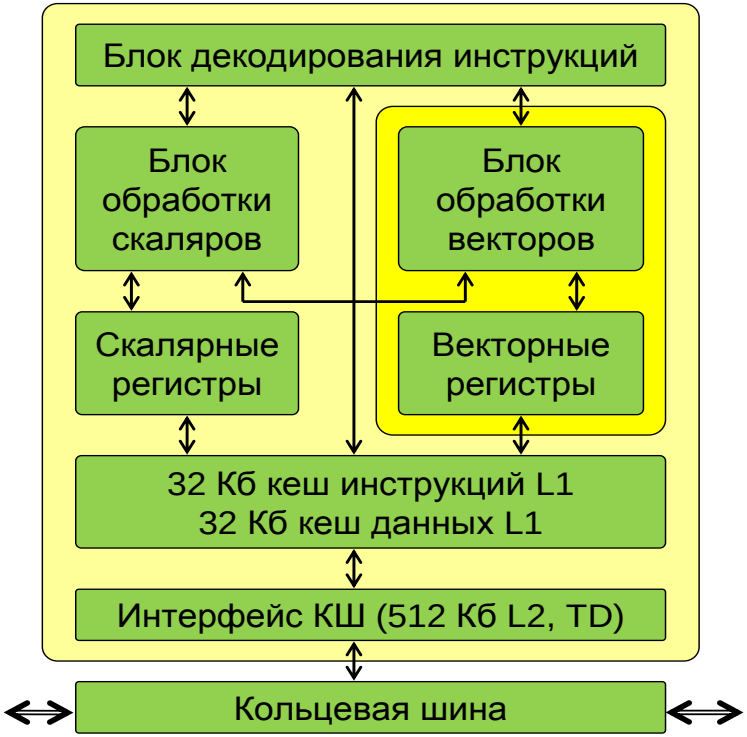
Исполнительное ядро Intel Xeon Phi

- 512-битный блок векторных вычислений (vector processor unit, VPU)
  - 32 512-битных регистра (zmm0-zmm31)
  - Включает расширенный блок математических вычислений (extended math unit, EMU)
  - Выполняет за 1 такт 16 операций над числами с плавающей точкой одинарной точности или 32-битными целыми числами или 8 операций над числами плавающей точкой двойной точности
  - Для операции «умножение и сложение» (multiply-add, FMA) - 32 операции над числами плавающей точкой одинарной точности за такт
  - Поддерживает операции заполнения и перестановки содержимого векторного регистра
  - 8 регистров масок для условного выполнения
  - Поддерживает вычисление для вещественных чисел одинарной точности 2^x, $\log_2 x$, 1/x, 1/sqrt(x)
  - Один из аргументов может считываться из оперативной памяти с выполнением при необходимости преобразования типа



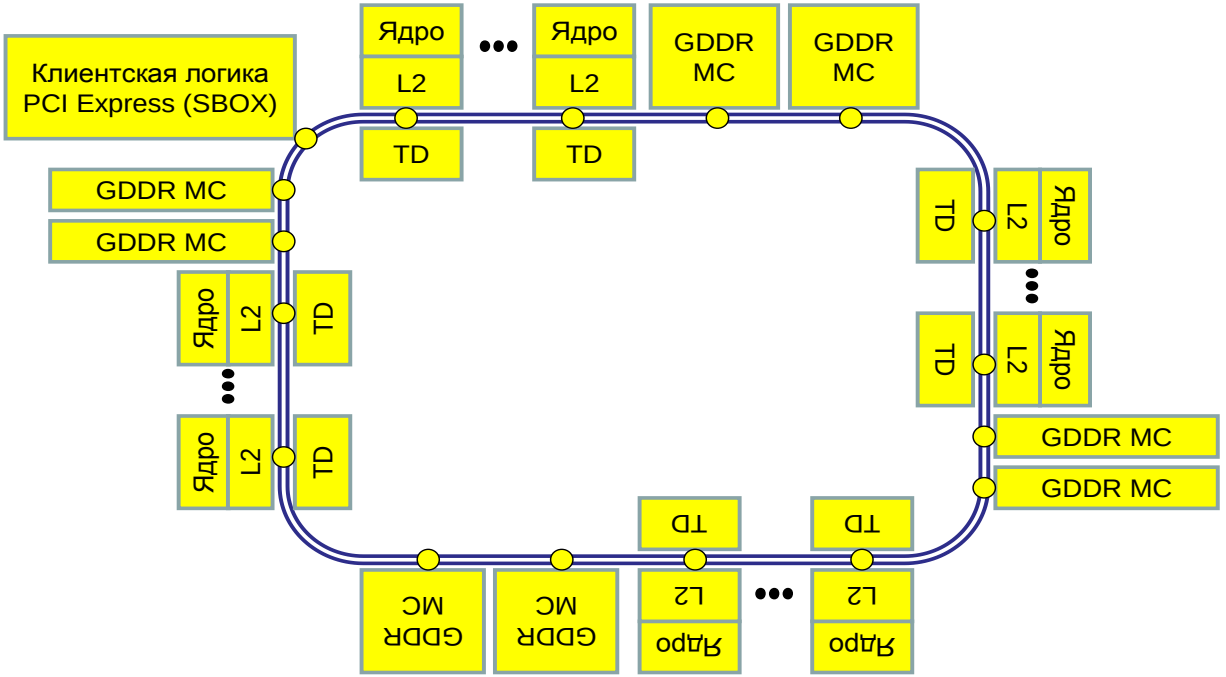
Исполнительное ядро Intel Xeon Phi

- Интерфейс кольцевой шины (Core-Ring Interface, CRI/L2)
  - Обеспечивает подключение ядра к высокопроизводительному встроенному интерконнекту сопроцессора
  - Включает 512 Кб 8-канального (8-way) множественно-ассоциативного кеша L2
- Каталог тегов (Tag Directory, TD) является частью распределенного каталога



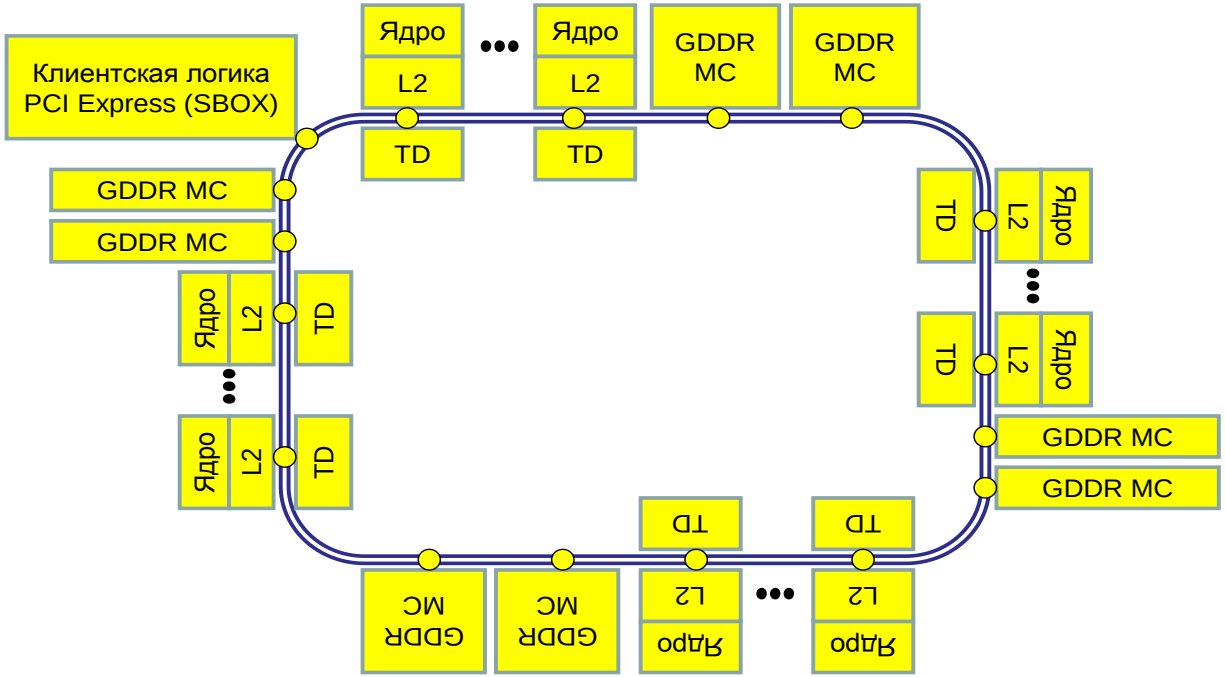
Компоненты Intel Xeon Phi

- Контроллер памяти (GDDR MC) включает: интерфейс кольцевой шины, планировщик запросов, интерфейс к устройствам GDDR
  - Каждый контроллер памяти включает два независимых канала доступа к памяти
  - Все контроллеры памяти сопроцессора действуют независимо друг от друга



Компоненты Intel Xeon Phi

- SBOX реализует клиентскую логику PCI Express, включая механизм прямого доступа к памяти (Direct Memory Access, DMA) и ограниченные возможности по управлению питанием
- Двунаправленная кольцевая шина обеспечивает передачу данных между компонентами сопроцессора



Теоретическая производительность

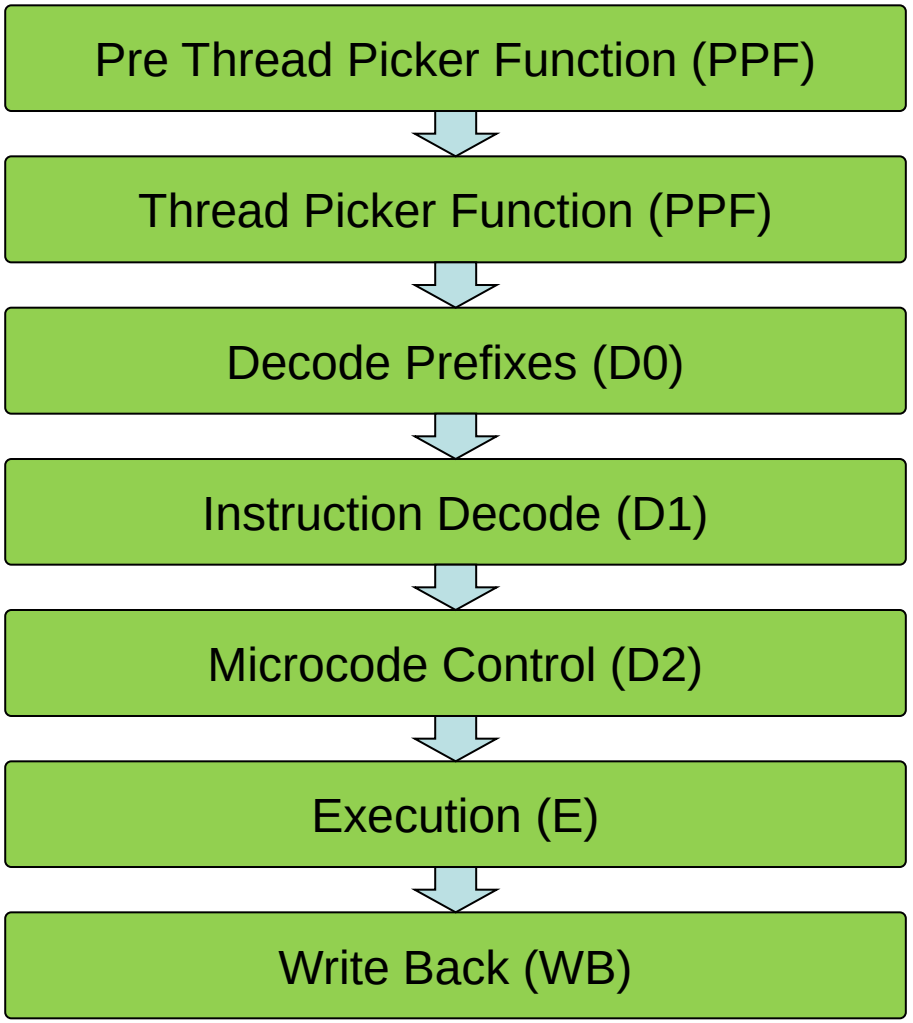
- Intel Xeon Phi содержит 61 ядро, но он исполняет собственную операционную систему, и одно ядро выделено для исполнения кода ОС
- Для вещественных чисел одинарной точности

  16 (длина вектора) * 2 flops(FMA) * 1.1 (GHZ) * 60 (число ядер) = **2112 GFLOPS**
- Для вещественных чисел двойной точности

  8 (длина вектора) * 2 flops (FMA) * 1.1 (GHZ) * 60 (число ядер) = **1056 GFLOPS**

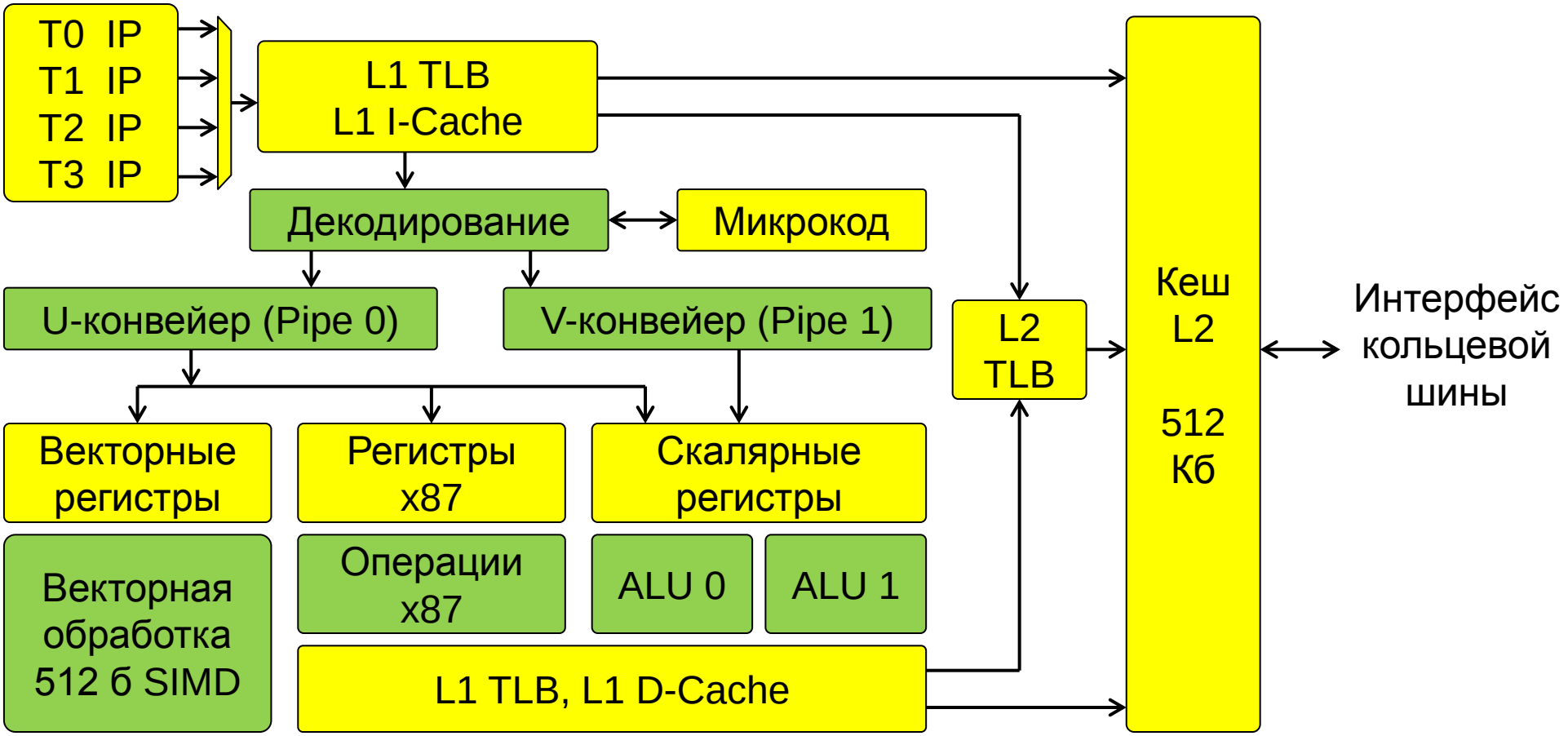
Конвейер ядра Intel Xeon Phi

Конвейер ядра Intel Xeon Phi…



- Содержит 7 этапов
- Блок векторных вычислений также имеет конвейерную структуру и состоит из 6 этапов
- Все этапы основного конвейера кроме последнего (WB), поддерживают спекулятивное выполнение
- Каждое ядро может выполнять инструкции 4 потоков

Конвейер ядра Intel Xeon Phi…



Конвейер ядра Intel Xeon Phi…

- Реализация выборки команд (стадии PPF и PF) накладывает ограничение на выполнение потоков – на двух последовательных тактах не могут выбираться инструкции одного и того же потока
  - для полной загрузки ядра необходимо выполнять на нем по крайней мере два потока одновременно, а при работе только одного потока выборка инструкций будет выполняться через такт
  - с учетом того, что заполнение буфера предварительной выборки требует 4-5 тактов при попадании в кеш инструкций и значительно больше при промахе, для обеспечения полной загрузки может потребоваться 3-4 потока

Конвейер ядра Intel Xeon Phi

- Этап исполнения (E), реализован в виде двух конвейеров – U и V
- Первая инструкция всегда отправляется на U-конвейер, для второй инструкции проверяется возможность одновременного выполнения с первой согласно набору правил парного выполнения команд
- Скалярные целочисленные операции выполняются арифметико-логическими устройствами (ALU), для скалярных и векторных операций с вещественными числами используется дополнительный 6-стадийный конвейер
- Векторные инструкции выполняются в основном на U-конвейере
- Большинство инструкций с целыми числами и масками имеют латентность 1, большинство векторных инструкций – 4 или более при использовании операций чтения/записи с заполнением или перестановкой

Векторные операции

Векторные операции…

- Длина вектора – 512 бит
- Вектор может обрабатываться как
  - 8 вещественных чисел двойной точности
  - 16 вещественных чисел одинарной точности
  - 8 64-битных целых чисел
  - 16 32-битных целых чисел
- Планируется поддержка
  - 32 16-битных целых чисел
  - 64 8-битных целых чисел

Векторные операции…

- Позволяют за одну инструкцию выполнить арифметическую операцию над несколькими парами операндов
  - Имеются унарные, бинарные и тернарные операции
- Использования векторизации
  - Специальные директивы или intrinsic
  - Автоматическая векторизация компилятором

```
double A[N], B[N], C = 0.0;
for(int i = 0; i < N; i ++){
  C += A[i] * B[i];
}
```

Векторные вычисления

- Аппаратно векторизованные функции для одинарной точности:
  2^x, $\log_2 x$, 1/x, 1/sqrt(x)
- Компилятор Intel поддерживает набор intrinsic, представляющих собой векторные версии скалярных математических операций (Short Vector Math Library Intrinsics, библиотека SVML)
  - Входной параметр – вектор, выходной параметр – вектор, над каждым элементом которого была выполнена скалярная операция
  - Поддерживаются операции для одинарной и двойной точности
  - Используется для векторизации циклов с математическими функциями

Операции, поддерживаемые SVML

acos	ceil	fabs	round
acosh	cos	floor	sin
asin	cosh	fmax	sinh
asinh	erf	fmin	sqrt
atan	erfc	log	tan
atan2	erfinv	log10	tanh
atanh	exp	log2	trunc
cbrt	exp2	pow	

Автовекторизация

- Компилятор Intel поддерживает возможность автоматической векторизации циклов
  - Для обеспечения возможности автоматический векторизации требуется выполнение ряда условий
  - Можно помочь компилятору выполнить векторизацию, используя специальные директивы pragma
  - Является рекомендуемым вариантом использования векторизации (обеспечивает переносимость кода между различными архитектурами)

Иерархия памяти Intel Xeon Phi

Иерархия памяти Intel Xeon Phi…

- Все ядра совместно используют оперативную память сопроцессора
- Каждое ядро сопроцессора Intel Xeon Phi имеет собственные кеши L1 и L2
- Кеши L1 и L2 являются инклюзивными
  - все данные, хранящиеся в кэше L1, хранятся также в кэше L2
- Кеши L1 и L2 используют псевдо-LRU алгоритм

Иерархия памяти Intel Xeon Phi…

- Кеши первого уровня - кеш инструкций L1 I-Cache и кеш данных L1 D-Cache
  - размер - по 32 Кб
  - размер строки - 64 байта
  - степень ассоциативности – 8
  - «чистая» латентность - 1 такт
  - средняя латентность доступа - 3 такта
  - load-to-use латентность - 1 такт (целочисленное значение, загруженное на текущем такте из кеша, может быть использовано на следующем такте целочисленной инструкцией, для векторных инструкций может быть больше)
  - обслуживает до ~38 одновременных запросов на ядро

Иерархия памяти Intel Xeon Phi…

- Кеш второго уровня L2
  - размер - 512 Кб
  - размер строки - 64 байта
  - степень ассоциативности – 8
  - 32 Гб кешируемых адресов (размер адреса 35 бит)
  - «чистая» латентность доступа - 11 тактов
  - средняя латентность доступа - 14-15 тактов
  - имеет аппаратное потоковое устройство предвыборки, способное выполнять избирательную предвыборку инструкций для исполнения и данных для операций чтения и записи
    - может инициировать до 4 составных запросов предвыборки, позволяет инициировать параллельную предвыборку до 4 Кб данных
  - обслуживает до ~38 одновременных запросов на ядро

Иерархия памяти Intel Xeon Phi…

- Для контроля когерентности кешей используется комбинация протоколов: MESI (Modified, Exclusive, Shared, Invalid) на каждом ядре и GOLS3 (Globally Owned Locally Shared) для распределенного каталога тегов
- Распределенный между ядрами каталог тегов сопроцессора (TD) разделен на 64 части, каждая из которых отвечает за контроль глобального состояния когерентности части строк кеша
- Особенности реализации кешей L1 и L2:
  - обращения к кешу L1 можно выполнять в последовательные такты, между обращениями к кешу L2 должен выдерживаться интервал в 1 такт (то есть к кешу L2 можно обращаться в лучшем случае через такт)
  - на каждом конкретном такте для L1 и L2 допускается выполнение либо чтения из кеша, либо записи в кеш, но не чтения и записи одновременно
  - реализованы только две схемы взаимодействия между кешем и основной памятью – отсутствие кеширования (uncacheable, UC) и отложенная запись (write-back, WB)
    - streaming stores

Иерархия памяти Intel Xeon Phi

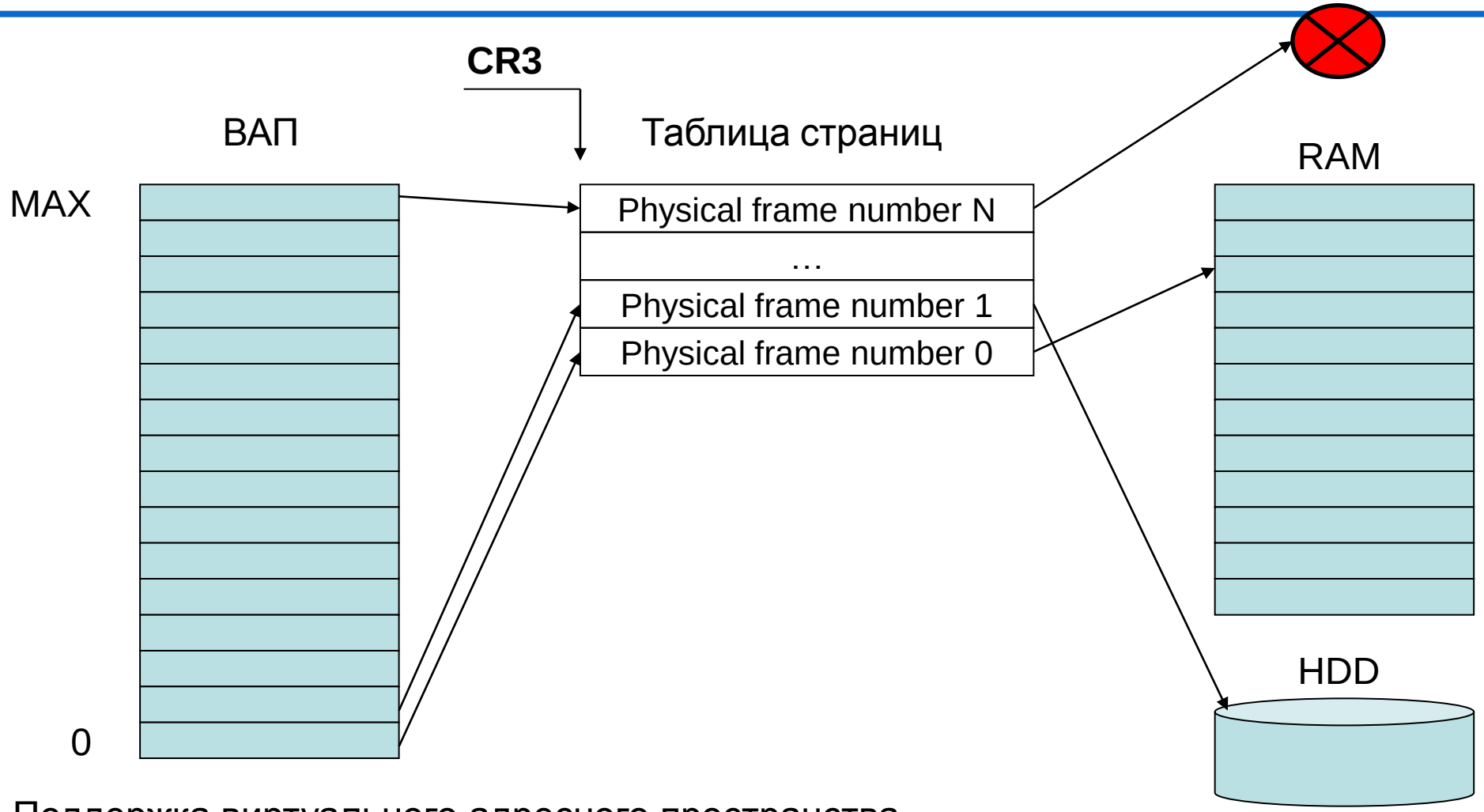
- Оперативная память
  - 8 встроенных контроллеров памяти, каждый обслуживает по два 32-битных канала GDDR5
  - суммарная производительность - 5,5 GT/s (миллиардов пересылок в секунду)
  - суммарная пропускная способность - 352 GB/s
  - латентность доступа - более 300 тактов
- Компонент сопроцессора, реализующий клиентскую логику PCI Express, также обеспечивает работу механизма прямого доступа к памяти (DMA). 8 независимых каналов DMA, работающих на той же частоте, что и ядра сопроцессора, могут выполнять следующие типы передачи данных:
  - из GDDR5-памяти сопроцессора в оперативную память хоста
  - из оперативной памяти хоста в GDDR5-память сопроцессора
  - из GDDR5-памяти в GDDR5-память в пределах сопроцессора

  - Выполнение операции передачи данных может быть запрошено как со стороны центрального процессора хоста, так и со стороны сопроцессора

Предвыборка данных (Prefetching)

- Использование intrinsics
  - _mm_prefetch((char *) &a[i], hint);
  - См. xmmintrin.h для возможных значений hint (for L1, L2, non-temporal, …)
- pragma / directive
  - #pragma prefetch a [:hint[:distance]]
- По умолчанию работает аппаратная предвыборка

Страничная модель адресации…

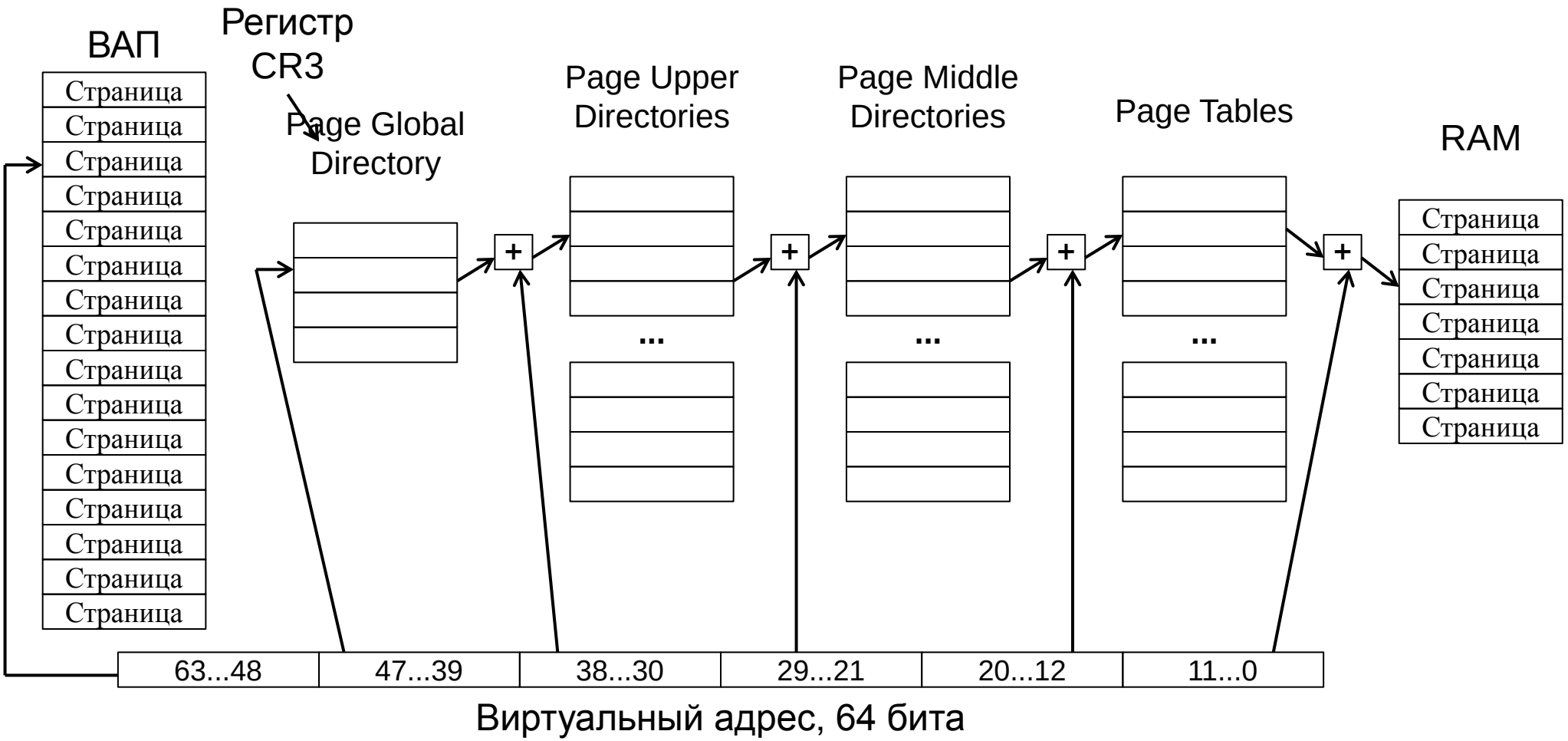


Поддержка виртуального адресного пространства процесса на основе страничного преобразования

Страничная модель адресации…

- Поддержка виртуальных адресных пространств процессов
  - 32-битные физические адреса при работе в 32-битном режиме
  - 36-битные адреса при использовании технологии PAE (Physical Address Extension) в 32-битном режиме
  - 40-битные физические адреса при работе в 64-битном режиме
- Операционная система сопроцессора поддерживает работу только в 64-битном режиме
- Процессам предоставляется линейное виртуальное адресное пространство (ВАП) и возможность использовать 64-битные адреса
  - Для поддержки ВАП используется стандартная схема архитектуры x86_64 – страничная адресация с 4 уровнями таблиц страниц

Страничная модель адресации…



Страничная модель адресации

- Поддерживаемые размеры страниц – 4 Кб и 2 Мб
- Кеш дескрипторов страниц (translation look-aside buffer, TLB) имеет двухуровневую архитектуру; каждое ядро имеет следующие кеши

Кеш	Размер страницы	Число записей	Степень ассоциативности	Соответствующий размер памяти
L1 TLB данных	4 Кб	64	4	256 Кб
	2 Мб	8	4	16 Мб
L1 TLB инструкций	4 Кб	32	4	128 Кб
L2 TLB	4 Кб, 2 Мб	64	4	128 Мб

Итоги

Требования к приложению, портируемому на MIC

- Параметры архитектуры Intel Xeon Phi позволяют выделить базовые характеристики приложения, определяющие возможность его эффективной адаптации
  - Эффективно распараллеленное на центральном процессоре
    - Перед портированием приложения на специализированную архитектуру необходимо убедиться в возможности построения эффективной параллельной реализации
  - Массивный параллелизм
  - Векторизация кода
  - Высокая вычислительная сложность и переиспользование данных
  - Использование “фиксированных” интерфейсов из MKL
  - MPI-приложения

Противопоказания

- Недостаточная степень параллелизма
- Множественные синхронизации
- Структуры данных подразумевают активное использование gather/scatter
- Использование 64-разрядных индексов или конверсия int64 <-> fp
- Интенсивная коммуникация между host-частью и Xeon Phi
- Ограничение памяти 8Gb при непосредственном исполнении на Xeon Phi (native mode)

Замечания для повышения производительности

- Размер задачи
  - Большие задачи имеют больший параллелизм
  - Но не слишком большие (доступно только 8GB RAM)
- Выравнивание данных
  - 64 байта для эффективного использования векторизации
- Количество потоков OpenMP и их привязка
  - Избегайте миграции потоков для сохранения локальности данных
- Используйте большие страницы
  - Уменьшает количество промахов TLB и накладные расходы на поддержку ВАП

Более подробно о векторных расширениях

Типы данных и регистровый пул

- В каждом ядре Xeon Phi сконструирован специальный модуль векторной обработки (VPU, vector processor unit)
  - 32 512-разрядных zmm-регистра; двукратное увеличение размера по сравнению с AVX
  - векторный FMA, fused multiply–add: ***a = a + b * c*** с однократным округлением.
- Одновременные действия над 16 32-битными целыми числами или 8 64-битными целыми числами или 16 числами с плавающей запятой одинарной точности или 8 числами с плавающей запятой двойной точности. Поддерживаются операции с комплексными данными.
- Большинство операций – тернарные (2 аргумента и один результат). По некоторым данным это приводит к 20% приросту производительности.

Обзор основных типов операций

- ***Арифметические операции***: сложение, вычитание, умножение, деление, FMA (для вычислений с плавающей запятой).
- ***Операции преобразования типов***, позволяющие выполнять повышающие и понижающие преобразования согласно определенным правилам (см. [1, 3]).
- ***Логические операции***, позволяющие выполнять векторные сравнения, находить минимум и максимум и т.д.
- ***Операции доступа к данным*** (загрузка/выгрузка память/регистр; scatter/gather, предвыборка, streaming stores). Могут применяться *маскирование*, *swizzle*, *shuffle*.

Расширенная поддержка математических функций

- В рамках Intel Xeon Phi была реализована расширенная поддержка некоторых математических функций:
  Команды для вычисления в одинарной точности:

Функция	Латентность	Пропускная способность
1/x	4	1
1/sqrt(x)	4	1
$\log_2(x)$	4	1
2^x	8	2

- **sqrt(x)**, **a^x** и **деление** могут быть вычислены при помощи указанных функций.

Способы векторизации

1. Использовать высокопроизводительные специализированные библиотеки, эффективно использующие векторные инструкции.
2. Написать программу на C/C++ или Fortran и откомпилировать ее тем транслятором, который «знает» про соответствующие наборы команд.
3. Использовать специальные ключи и директивы компилятора (подсказки).
4. Использовать возможности Array Notation и Elemental Function в рамках технологии Intel Cilk Plus.
5. Использовать классы интринсиков для SIMD.
6. Использовать векторные функции-интринсики.
7. Написать реализацию на ассемблере.

Векторизация. Используем векторизованные библиотеки

- **MKL** и не только.

- **Проблемы**:
  - Не все, что нам нужно, есть в библиотеке.
  - Реализация, присутствующая в библиотеке, не всегда оптимальна для нашей конкретной задачи.
  - Сложности интеграции, поддержки, миграции на другие платформы.

Векторизация. Используем C/C++ или Fortran в сочетании с оптимизирующим компилятором

Intel Compliers (предпочтительно), **gcc**

- Крайне желательно минимизировать всевозможные зависимости по данным.
- Нежелательно вызывать функции в вычислительно трудоемких циклах. Однако вызов функции тоже не приговор. Компилятор может встроить код функции и успешно векторизовать цикл.
- Необходимо следить за выравниванием данных в памяти, то есть за их размещением с «правильных» адресов, кратным определенному числу байт (размеру кеш-линии, размеру векторного регистра).

  SSE: по 16, AVX: по 32, Xeon Phi: по 64

  __declspec(aligned), __mm_malloc()…

Векторизация. Используем C/C++ или Fortran в сочетании с оптимизирующим компилятором

- Необходимо стараться обеспечить однородный доступ к памяти, когда мы загружаем/выгружаем данные, лежащие последовательно (крайне желательно) либо с одинаковым шагом (допустимо).
- Необходимо сводить к минимуму смешивание объектов разных типов данных в выражениях.
- Необходимо по возможности избавляться от условных операторов в теле внутреннего цикла. Раньше компилятор в принципе отказывался векторизовывать такие циклы. Сейчас в ряде случаев ему удается это сделать.

Примеры:

Векторизация. Используем C/C++ или Fortran в сочетании с оптимизирующим компилятором

Пример 1:

```
#pragma simd
#pragma vector aligned
for (int i = 0; i < n; i++) {  s = s + max(a[i],0); }
```

Пример 2:

```
#pragma simd
#pragma vector aligned
for (int i = 0; i < n; i++)
  if (a[i] == key) {
    Index = i; break;
  }
```

Векторизация. Используем C/C++ или Fortran в сочетании с оптимизирующим компилятором

Пример кода, автоматически векторизуемого компилятором:

```
void test(float * restrict a, float * restrict b, float * restrict c, int n)
{
  for (int i = 0; i < n; i++)
    c[i] = a[i] * b[i] + a[i];
}
```

- **restrict**: мы говорим компилятору, что доступ в указанную память будет осуществляться только через указатель, использованный при объявлении. Не забываем ключ **–restrict**.
- Это необходимо, что компилятор зафиксировал факт отсутствия зависимостей по данным между массивами a, b и c и успешно векторизовал цикл.

Векторизация. Используем ключи и директивы компилятора

- -O2, -O3
- #pragma ivdep
- #pragma vector
  - always
  - aligned
  - nontemporal
- #pragma simd (с параметрами)
  - самый мощный на сегодняшний день инструмент;
  - необходимо пользоваться с осторожностью.

Векторизация
и математические функции

Введение

- Отдельно необходимо обсудить важный вопрос о сочетании векторизации и математических функций, вызываемых в циклах, так как именно на вычисление этих функций приходится основное время работы значительного числа прикладных программ.

- Ранее мы установили факт наличия в наборе команд Intel Xeon Phi специальных инструкций для вычисления четырех математических функций.

Как быть с остальными функциями?

Пример

```
void test(float * a, float * b,
          float * c, int n)
{
  #pragma simd
  #pragma vector aligned
  for (int i = 0; i < n; i++)
    c[i] = a[i] * b[i] + sinf(a[i]);
}
```

Отчет: цикл векторизован.

Вопрос: как векторизуется синус?

Реализации математических функций

- **LibM** – модуль компилятора.
  ICC содержит быстрый LibM, оптимизированный под современные архитектуры.
- **SVML** (short vector math library) – модуль компилятора ICC.

  Используется, если цикл векторизован.

  Мат. функции реализованы с использованием SIMD, вычисляются для короткого вектора аргументов. Длина вектора соответствует длине xmm, ymm, zmm регистра.
- **VML** (vector math library) – часть библиотеки MKL.

  Используется при явном вызове функций (vsSin, vdSin…).

  Вычисляет значение функции в N точках.

Реализации математических функций

- **LibM** (см. math.h). Перекомпиляция ICC в программах, активно использующих мат. функции, часто приводит к ускорению расчетов.
- **SVML vs. VML**

  VML может выигрывать у SVML на больших длинах, но не всегда (эффект существенно зависит от архитектуры).

```
for (int i = 0; i < n; i++)
    c[i] = a[i] * b[i] + sinf(a[i]);
```

Не векторизован: **LibM**

Векторизован: **SVML**

VML: vsSin(n, a, c);

Реализации математических функций. Точность

Настройки, влияющие на точность (ICC):

- -fp-model
- -fimf-domain-exclusion
- -fimf-precision

Настройки, влияющие на точность (MKL/VML):

- Режимы High Accuracy (HA), Low Accuracy (LA), Enhanced Performance (EP).
- Могут быть настроены для конкретного вызова.

Литература

- 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]
- 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]
- Intel Xeon Phi Coprocessor System Software Developers Guide
  - http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide]
- Rahman R. Intel Xeon Phi Core Microarchitecture
  - [http://software.intel.com/en-us/articles/intel-xeon-phi-core-micro-architecture]