УДК 004.4'422
DOI: 10.25559/SITITO.18.202202.249-255

Научная статья

Оптимизация разбиения структур для векторного оптимизатора в графическом компиляторе Intel

К. И. Владимиров*, И. В. Андреев
ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)», г. Долгопрудный, Российская Федерация
Адрес: 141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9
*konstantin.vladimirov@gmail.com

Аннотация
Вычисления на видеокарточках и специализированных ускорителях широко используются для решения большого количества важных практических задач. Разработчики на таких языках как OpenCL, SYCL, CM и ISPC существенно полагаются на качество оптимизаций в графическимх компиляторах. Для Intel GPU компилятор имеет две части: скалярную, которая работает в модели SIMT и векторную, нацеленную на SIMD языки. Именно векторная часть компилятора вносит наибольший вклад, когда речь заходит о критических задачах, таких как обучение нейросетей, решение систем уравнений, рендеринг изображений и так далее. К сожалению, до недавнего времени в архитектуре графического компилятора Intel отсутствовала возможность хорошо раскладывать структуры по векторным регистрам, что приводило к особенным проблемам с производительностью в программах, написанных на ISPC, таких как Embree и OSPRay. Чтобы решить эту проблему, предлагается алгоритм разбиения структур для векторного оптимизатора графического компилятора Intel. Приводится подробное описание алгоритма и замеры производительности, показывающие на некоторых задачах прирост до 80%.

Ключевые слова: компиляторы, оптимизации, графика, структуры, вектора

Авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Владимиров К. И., Андреев И. В. Оптимизация разбиения структур для векторного оптимизатора в графическом компиляторе Intel // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 2. С. 249-255. doi: https://doi.org/10.25559/SITITO.18.202202.249-255

Original article

Structures Partitioning Optimization for Vector Optimizer in Intel Graphics Compiler

K. I. Vladimirov*, I. V. Andreev
Moscow Institute of Physics and Technology (National Research University), Dolgoprudny, Russian Federation
Address: 9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation
*konstantin.vladimirov@gmail.com

Abstract
Computations on video cards and specialized accelerators are widely used to solve many important practical tasks. Developers working with OpenCL, SYCL, CM, and ISPC rely heavily on the quality of optimizations in graphics compilers. For the Intel GPU, the compiler has two parts: a scalar part that works in the SIMT model, and a vector part that targets SIMD languages. It is the vector part of the compiler that contributes the most when it comes to critical tasks such as training neural networks, solving systems of equations, rendering images, and so on. Unfortunately, until recently, the Intel graphics compiler architecture lacked the ability to properly decompose into vector registers, which led to particular performance problems in programs written in ISPC, such as Embree and OSPRay. To solve this problem, we propose a structure partitioning algorithm for the vector optimizer of the Intel graphics compiler. A detailed description of the algorithm and performance measurements are given, showing an increase of up to 80% on some tasks.

Keywords: compilers, optimizations, graphics, structures, vectors

The authors declare no conflict of interest.

For citation: Vladimirov K.I., Andreev I.V. Structures Partitioning Optimization for Vector Optimizer in Intel Graphics Compiler. *Sovremennye informacionnye tehnologii i IT-obrazovanie = Modern Information Technologies and IT-Education*. 2022; 18(2):249-255. doi: https://doi.org/10.25559/SITITO.18.202202.249-255

Введение

Вычисления на видеокартах и видеоускорителях (будем коллективно называть их GPU) являются эффективным средством решения широкого класса практически важных задач [1], [4], [16]. Существенный прирост производительности по сравнению с микропроцессором общего назначения (CPU) достигается за счёт наличия у GPU большого числа независимых ядер, способных выполнять вычисления параллельно. Например, свёрточная нейросеть (CNN) тренируется на GPU в шесть раз быстрее, чем на CPU. Также рендеринг изображений на GPU завершается значительно раньше. Тысячи вычислительных потоков GPU совместно достигают производительности в несколько десятков и даже сотен терафлопс. Но итоговая производительность программы для GPU (шейдера или кернела) часто также зависит от усилий по оптимизации. Часть обязанностей здесь ложится на программиста, а часть, особенно связанная с низкоуровневыми вещами – на компилятор [8], [9].

Одной из самых долгих операций при вычислениях является работа с памятью. Иерархия памяти в видеускорителях отлична от CPU [10], [11]. Чтение изображений проходит через L1 и L2 кэши, через L3-кэш все операции на чтение/запись данных и запись изображений, которой не хватало в L1 и L2, а также чтение изображений. Дополнительно к L3 существует локальная память рабочих групп (shared local memory, SLM). Все операции производятся в каждом EU (execution unit), из которых состоит subslice, с данными из регистрового файла (general register file, GRF). Дальше от EU расположен кеш последнего уровня (last level cache, LLC), который уже может делиться между CPU и GPU. Далее, возможно, встроенная DRAM и затем системная DRAM. При этом, каждый из перечисленных уровней памяти накладывает свои ощутимые задержки, тем большие, чем дальше происходит доступ к данным.

В этой работе в основном будет рассматриваться контекст, порождённый из языка ISPC [6], [7] но подобный же контекст можно получить из SYCL с расширениями [12]-[15], [17] и других векторных языков.

Структуры – это набор из одной или более переменных, возможно различных типов, сгруппированных под одним именем для удобства обработки. Они полезны при организации сложных данных особенно в больших программах, так как они позволяют сгруппировать данные таким образом, что с ними можно обращаться как с полями одного объекта. Поскольку структуры широко распространены в программах, любые действия и оптимизации над ними будут влиять на большое количество приложений, что накладывает дополнительные требования по корректности и стабильности преобразований.

В видеоускорителях Intel Xe на низком уровне реализована концепция SIMD – Single Instruction Multiple Data. Широкая операция производится над всеми значениями в векторе за раз, и каждое значение в векторе вычисляется независимо. В векторном оптимизаторе Intel Graphics Compiler уже существует стадия, векторизующая различные типы данных [3].

В этой статье мы представим алгоритм разбиения структур, позволяющий свести сложные структуры к более простым, которые можно будет разложить на регистры видеоускорителя, алгоритм замены инструкций, приведём результаты замеров на тестах из открытого стека рендеринга Intel (OSPray, Embree), предложим дальнейшие возможные улучшения.

Архитектура

Семейство видеокарт Intel состоит из различных микроархитектур, включающих энергоэффективные модели -LP и высокопроизводительных -HPG и -HPC , ориентированных на игры и вычисления [18].

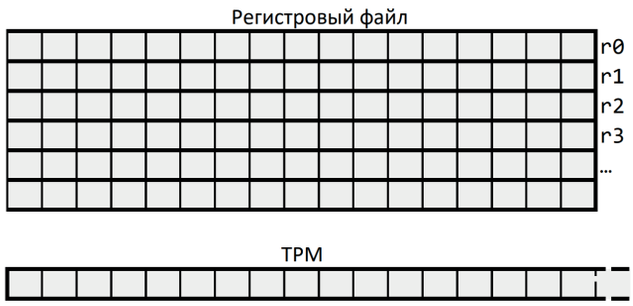
Для LP одновременно выполняемые потоки (SMT) объединены в Execution Unit (EU). Каждый EU состоит из семи потоков SMT. Главный вычислительный модуль состоит из SIMD8 ALU, поддерживающий SIMD8 FP/INT операции – целочисленные или с плавающей точкой, и SIMD2 ALU, поддерживающий расширенные математические операции. Каждый поток имеет 128 регистров (general-purpose registers) каждый размером 32 байта. Вся совокупность регистров для одного потока называется general register file (GRF). Важной концепцией является Shared Local Memory (SLM) – это общая память на рабочую группу потоков. 16 EUs объединены в Dual Subslice (DSS) с кэшем инструкций, памятью SLM и портом 128B/cycle. Два EU могут быть объединены в пару для выполнения SIMD16 инструкций. Каждые 6 DSS объединены в Slice вместе с 16MB L3 кэшем.

В отличие от LP, где в качестве вычислительной ячейки используется EU, в HPC и HPG используются -core, похожие на LP DSS. -core содержит 8 векторных и 8 матричных движков, которые производят высокопроизводительные вычисления. Core объединены в Slice, Slice в Stack. В итоге получается объединение большого числа вычислительных потоков [19]-[21].

Все вычисления выполняются над данными, расположенными в регистрах – сверхбыстрая память, располагающаяся непосредственно рядом с вычислительными элементами. Преимущество в скорости накладывает сильное ограничение на размер регистровой памяти. Чтобы эффективно расположить данные на регистрах используется алгоритм раскрашивания RIG (Register interference graph) [2].

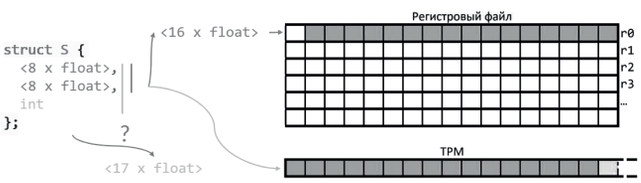
Наибольший вред производительности наносит выход за пределы регистрового файла. Тогда перед использованием данных они загружаются из памяти (fill), а после использования – выгружаются обратно (spill). Поскольку работа с памятью на порядок медленнее работы с регистрами, в важных частях программы возникают задержки, и общая производительность снижается катастрофически.

Скорость обращения к данным для видеоускорителей так же сильно отличается в зависимости от используемой памяти. К данным на регистрах обращение идёт напрямую, в то время как обращение к данным в TPM (Thread Private Memory – индивидуальное место в памяти для потока) идёт опосредованно, с предварительной загрузкой/отправкой этих данных сообщением send. Схематичное изображение регистрового файла и TPM на (рис. 1). Для видеоускорителей критически необходим быстрый доступ к данным, иначе потоки будут большую часть времени проводить в ожидании. Поэтому нужно данные с наиболее частым обращением раскладывать на регистры. Регистров в GPU больше чем в CPU, и они представляют собой регистровую матрицу.



Р и с. 1. Репрезентация регистрового файла и TPM
F i g. 1. Register File Representation and TPM

Векторный компилятор Intel имеет возможность векторизовать и разложить на регистры последовательные типы данных. В них входят вектора, массивы, структуры, состоящие из элементов одного примитивного типа и так далее. Так, например, компилятор может обработать структуру type {float, <7 x float>} и преобразовать её в вектор <8 x float>.



Р и с. 2. Возможное расположение структуры в памяти
F i g. 2. Possible structure location in memory

Однако, векторизация не справляется с более сложными структурами. На рис. 2 видно, что структура S без поля int может быть оптимизирована, векторизована и положена на регистры. Но как только появляется дополнительное поле иного типа, в данном случае это int, встаёт вопрос как векторизовать такой объединённый тип. Это приводит к использованию структур из памяти, появлению spill'ов и, таким образом, к сильному падению производительности. Обсуждаемое ниже решение проблемы векторизации структур, также решает проблему размещения структур на регистры и сокращает количество обращений в память.

Разбиение структур

Перед описанием алгоритма разбиения определим правила, по которым мы будем выделять части агрегатных типов для векторизации.

В стандарте C++ существует концепция скалярного типа. Скалярными называются арифметические типы, типы перечислений, указатели и cv-квалифицированные версии перечисленных типов. Назовём базовым скалярный тип, поддерживаемый данной оптимизацией. В это множество входят также векторные типы, которых нет в C++, но которые есть в LLVM IR [5], [28]. В это множество не входят, например, указатели. Определение сознательно является несколько нечетким, чтобы заложить возможность будущего расширения. Алгоритм ниже не разбивает базовые типы.

Назовём примитивным либо базовый тип, либо агрегатный тип, у которого типы всех элементов одинаковы и примитивны. Такой тип нет необходимости делить, так как объекты таких типов могут быть легко переделаны в вектора фазой aggregate lowering, которая представлена в векторном оптимизаторе IGC. Для алгоритма нет разницы между примитивным типом и базовым, из которого этот примитивный тип состоит, поэтому примитивный всегда будет сводиться к базовому. Например, <3 x [5 x int]> будет эквивалентен типу int.

Целью алгоритма разбиения структур является приведение всех структур в модуле (например в модуле LLVM IR) к структурам примитивного типа с обязательным сохранением поведения исходной программы в рамках as-if rule.

Фаза разбиения структур в векторном компиляторе разрабатывалась специально для ISPC – компилятора для SPMD (single program, multiple data). При этом алгоритм сам по себе может быть использован в более широких классах оптимизаторов, даже не основанных на LLVM [26], [27]. Оптимизация работает над LLVM модулем и должна применятся до векторизации.

Разбиение невозможно, если:

1. элемент структуры является указателем на другую нетривиальную структуру, в том числе на саму себя.
2. на структуру взят указатель.

Первое ограничение появляется из-за использования версии LLVM, в которой ещё не были введены opaque указатели. Собственноручная поддержка таких указателей приводит к сильному разрастанию получаемого модуля и невозможности применения других оптимизаций. По этой же причине была невозможна поддержка передачи структур в пользовательские функции. Второе ограничение связано с тем, что замена кода работы с указателем на аналогичный код с поделёнными структурами может привести к значительному увеличению количества инструкций.

Описание алгоритма

Шаг 1. Сбор информации о структурах в модуле

Информация о структуре хранится в виде хэш-таблицы, где ключ – это примитивный тип элемента, а значения – элементы, соответствующие этому типу. Структура в дальнейшем будет делиться как раз по элементам этой хэш-таблицы, так как ключам будет соответствовать примитивный тип, а значениям – элементы будущей структуры. Количество ключей минус один – столько структур необходимо будет сгенерировать. Соответственно, если у структуры существует только один примитивный тип, то данную структуру делить нет необходимости, так как она сама является примитивной.

Шаг 2. Построение графа вложенности структур

Так как элементами структуры могут быть другие структуры, то такие ситуации необходимо разрешать. Граф строится следующим образом: вершине соответствует структура %A. Ребро от вершины в вершину проводится в том случае, если структура, соответствующая вершине , вложена в структуру, соответствующую вершине . Назовём головой графа() вершины, которые соответствуют структурам, не вложенным ни в какие другие структуры. Голов графа может быть как одна, так и несколько.

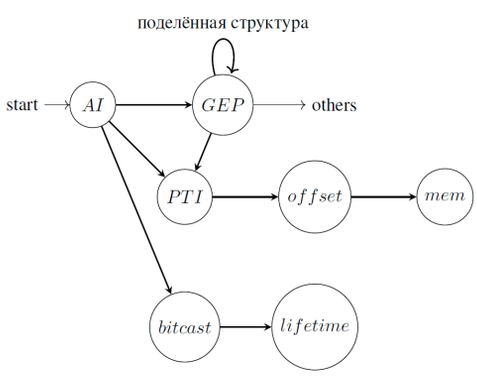
Шаг 3. Обработка графа вложенности структур

Обработка графа начинается с головы, однако деление – с самой нижней вершины. Все нижние вершины всегда характеризуются тем, что структуры, связанные с ними, содержат элементы только примитивных типов, а значит данные структуры могут быть поделены. При обработке вершины графа возможны два варианта. Первое – если вершина отвечает за примитивную структуру, то в делении нет необходимости и данная вершина удаляется из графа. Второе – структура, отвечающая за вершину, была поделена. Тогда нужно заменить во всех структурах, в которые была вложена данная, все элементы, отвечающие за данную структуру, на новые элементы с поделёнными структурами. Во время замены элементов появляются промежуточные представления структур, которые затем необходимо удалить. После обработки вершины она удаляется, поэтому все структуры будут обработаны тогда, когда весь граф удалится.

Шаг 4. Обработка и замена инструкций

Обработка инструкций всегда начинается с инструкции выделения памяти на стековом фрейме под структуру (AI – alloca в LLVM IR). Далее единственными допустимыми пользователями инструкций выделения памяти являются только инструкции обращения к элементу агрегата (GEP – getelementptr). Однако в нашей реализации были допущены инструкции приведения (bitcast) и ptrtoint (PTI) только с определёнными шаблонами использования. В случае bitcast единственными пользователями могут быть только интринсики lifetime.start/end. В случае ptrtoint фронтенд ISPC вместо доступа к нулевому элементу структуры через getelementptr использует указатель на структуру. Данный подход необходимо было поддерживать, поэтому в анализе ptrtoint проверяется соответствие шаблону вычисления указателя из следующих инструкций: insertelement, shufflevector, add и read/write. В случае если результатом обращения к элементу является примитивный тип, то эта инструкция заменяется на эквивалентную с другим операндом и новой цепочкой индексов. Если результатом является уже разбитая структура, то будут сгенерированы столько новых обращений, на сколько структур была разбита исходная.

На (рис. 3) показан автомат обработки инструкций в модуле.



Р и с. 3. Автомат обработки инструкций

F i g. 3. Instruction Processing Machine

Результаты

В рамках этой работы была реализована стадия компиляции, направленная на деление структур по типам, с целью их последующей векторизации. Было показано, что это помогает распределению структур на векторные регистры и уменьшает количество обращений в память. Разработан алгоритм с делением структур неограниченной вложенности. Реализация внедрена в графический компилятор Intel. Предложен способ замены инструкций с сохранением корректности работы программы.

Для тестирования алгоритма использовались приложения из открытого стека рендеринга Intel (OSPray, Embree) [22]-[25]. Была оценена корректность алгоритма (прошли все тесты) и его положительное влияние на производительность скомпилированной программы.

Test name	№
stable_v1 pathtracer gravity_spheres_volume	1
stable_v1 raymarch gravity_spheres_volume	2
stable_v1 raymarchiterator gravity_spheres_volume	3
stable_v2 pathtracer gravity_spheres_volume	4
stable_v2 raymarch gravity_spheres_volume	5
stable_v2 raymarchiterator gravity_spheres_volume	6
stable_v2 ng boxes	7
stable_v3 scivis boxes	8
stable_v3 scivis boxes_lit	9
stable_v3 scivis gravity_spheres_volume	10
stable_v3 scivis gravity_spheres_isosurface	11
stable_v3 scivis gravity_spheres_volume_with_isosurface	12

Р и с. 4. Названия тестов

F i g. 4. Test Names

№	DG2		PVC	TGLLP		GEN9	
	simd8	simd16	simd16	simd8	simd16	simd8	simd16
1	0.04%	0.01%	0.15%	-0.40%	0.07%	0.02%	-0.74%
2	0.10%	0.16%	0.41%	1.67%	0.00%	0.00%	0.00%
3	-0.06%	0.02%	-0.82%	1.60%	-0.01%	0.02%	-0.05%
4	5.43%	7.10%	6.84%	16.13%	12.75%	3.30%	4.86%
5	1.69%	4.10%	-3.29%	нет данных	4.24%	2.58%	45.29%
6	1.61%	62.16%	1.03%	0.30%	79.83%	13.87%	9.88%
7	42.58%	74.27%	47.28%	57.59%	83.68%	39.05%	59.36%
8	76.38%	53.25%	81.41%	88.43%	21.68%	38.65%	76.82%
9	76.34%	53.08%	80.83%	87.93%	21.24%	38.47%	76.70%
10	59.45%	39.27%	53.45%	72.44%	30.73%	4.32%	11.00%
11	75.04%	18.70%	68.35%	нет данных	0.00%	16.05%	9.86%
12	70.71%	24.29%	65.55%	нет данных	32.51%	12.43%	15.82%

Р и с. 5. Результаты работы

F i g. 5. Work Results

На рис. 4 и рис. 5 показан прирост скорости работы приложений при различной ширине SIMD и на различных платформах. В тех тестах, где структуры не были использованы, нет прироста производительности. Однако с ростом сложности программ и с увеличением случаев обращения к структурам данных наблюдается прирост производительности вплоть до 88%.

Разработанная оптимизация открывает целый спектр дальнейших возможных улучшений, включая разбиение массивов и векторов структур, введение пользовательских эвристик, поддержку указателей и многое другое.

References

[1] Lueh G.-Y., et al. C-for-Metal: High Performance Simd Programming on Intel GPUs. *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Computer Society, Seoul, Korea (South); 2021. p. 289-300. (In Eng.) doi: https://doi.org/10.1109/CGO51591.2021.9370324

[2] Chen W.-Y., Lueh G.-Y., Ashar P., Chen K., Cheng B. Register allocation for Intel processor graphics. *Proceedings of the 2018 International Symposium on Code Generation and Optimization (CGO'2018).* Association for Computing Machinery, New York, NY, USA; 2018. p. 352-364. (In Eng.) doi: https://doi.org/10.1145/3168806

[3] Chandrasekhar A., et al. IGC: The Open Source Intel Graphics Compiler. *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Computer Society; 2019. p. 254-265. (In Eng.) doi: https://doi.org/10.1109/CGO.2019.8661189

[4] Castaño G., et al. Evaluation of Intel's DPC++ Compatibility Tool in heterogeneous computing. *Journal of Parallel and Distributed Computing.* 2022; 165:120-129. (In Eng.) doi: https://doi.org/10.1016/j.jpdc.2022.03.017

[5] Lattner C., Adve V. The LLVM Compiler Framework and Infrastructure Tutorial. In: Eigenmann R., Li Z., Midkiff S.P. (eds.) *Languages and Compilers for High Performance Computing. LCPC 2004. Lecture Notes in Computer Science.* Vol. 3602. Springer, Berlin, Heidelberg; 2005. p. 15-16. (In Eng.) doi: https://doi.org/10.1007/11532378_2

[6] Pharr M., Mark W.R. *ISPC: A SPMD compiler for high-performance CPU programming. 2012 Innovative Parallel Computing (InPar).* IEEE Computer Society; 2012. p. 1-13. (In Eng.) doi: https://doi.org/10.1109/InPar.2012.6339601

[7] Brodman J., Babokin D., Filippov I., Tu P. Writing scalable SIMD programs with ISPC. *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing (WPMVP'14).* Association for Computing Machinery, New York, NY, USA; 2014. p. 25-32. (In Eng.) doi: https://doi.org/10.1145/2568058.2568065

[8] Tian X., Saito H., Su E., Lin J., Guggilla S., Caballero D., Masten M., Savonichev A., Rice M., Demikhovsky E., Zaks A., Rapaport G., Gaba A., Porpodas V., Garcia E. LLVM Compiler Implementation for Explicit Parallelization and SIMD Vectorization. *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC'17).* Association for Computing Machinery, New York, NY, USA; 2017. Article number: 4. p. 1-11. (In Eng.) doi: https://doi.org/10.1145/3148173.3148191

[9] Tian X., Saito H., Su E., Gaba A., Masten M., Garcia E., Zaks A. LLVM framework and IR extensions for parallelization, SIMD vectorization and offloading. *Proceedings of the Third Workshop on LLVM Compiler Infrastructure in HPC (LLVM-HPC'16).* IEEE Computer Society; 2016. p. 21-31. (In Eng.) doi: https://dl.acm.org/doi/10.5555/3018869.3018872

[10] Ashbaugh B., Brodman J.C., Kinsner M., Lueck G., Pennycook J., Schulz R. Toward a Better Defined SYCL Memory Consistency Model. *International Workshop on OpenCL (IWOCL'21).* Association for Computing Machinery, New York, NY, USA; 2021. Article number: 20. p. 1-3. (In Eng.) doi: https://doi.org/10.1145/3456669.3456696

[11] Mrozek M., Ashbaugh B., Brodman J. Taking Memory Management to the Next Level: Unified Shared Memory in Action. *Proceedings of the International Workshop on OpenCL (IWOCL'20).* Association for Computing Machinery, New York, NY, USA; 2020. Article number: 1. p. 1-3. (In Eng.) doi: https://doi.org/10.1145/3388333.3388644

[12] Ashbaugh B., Bader A., Brodman J., Hammond J., Kinsner M., Pennycook J., Schulz R., Sewall J. Data Parallel C++: Enhancing SYCL Through Extensions for Productivity and Performance. *Proceedings of the International Workshop on OpenCL (IWOCL'20).* Association for Computing Machinery, New York, NY, USA; 2020. Article number: 7. p. 1-2. (In Eng.) doi: https://doi.org/10.1145/3388333.3388653

[13] Reinders J.R. SYCL, DPC++, XPUs, oneAPI. *International Workshop on OpenCL (IWOCL'21).* Association for Computing Machinery, New York, NY, USA; 2021. Article number: 19. p. 1 (In Eng.) doi: https://doi.org/10.1145/3456669.3456719

[14] Alpay A., Soproni B., Wünsche H., Heuveline V. Exploring the possibility of a hipSYCL-based implementation of oneAPI. *International Workshop on OpenCL (IWOCL'22).* Association for Computing Machinery, New York, NY, USA; 2022. Article number: 10. p. 1-12. (In Eng.) doi: https://doi.org/10.1145/3529538.3530005

[15] Hardy D.J., Choi J., Jiang W., Tajkhorshid E. Experiences Porting NAMD to the Data Parallel C++ Programming Model. *International Workshop on OpenCL (IWOCL'22).* Association for Computing Machinery, New York, NY, USA; 2022. Article number: 15. p. 1-5. (In Eng.) doi: https://doi.org/10.1145/3529538.3529560

[16] Fang J., Huang C., Tang T., et al. Parallel programming models for heterogeneous many-cores: a comprehensive survey. *CCF Transactions on High Performance Computing*. 2020; 2(4):382-400. (In Eng.) doi: https://doi.org/10.1007/s42514-020-00039-4

[17] Alpay A., Heuveline V. How much SYCL does a compiler need? Experiences from the implementation of SYCL as a library for nvc++. *International Workshop on OpenCL (IWOCL'22).* Association for Computing Machinery, New York, NY, USA; 2022. Article number: 11. p. 1. (In Eng.) doi: https://doi.org/10.1145/3529538.3529556

[18] Aktemur B., Metzger M., Saiapova N., Strasuns M. Debugging SYCL Programs on Heterogeneous Intel® Architectures. *Proceedings of the International Workshop on OpenCL (IWOCL'20).* Association for Computing Machinery, New York, NY, USA; 2020. Article number: 13. p. 1-10. (In Eng.) doi: https://doi.org/10.1145/3388333.3388646

[19] Reyes R., Brown G., Burns R., Wong M. SYCL 2020: More than meets the eye. *Proceedings of the International Workshop on OpenCL (IWOCL'20).* Association for Computing Machinery, New York, NY, USA; 2020. Article number: 4. p. 1. (In Eng.) doi: https://doi.org/10.1145/3388333.3388649

[20] Nozal R., Bosque J.L. Exploiting Co-execution with OneAPI: Heterogeneity from a Modern Perspective. In: Sousa L., Roma N., Tomás P. (eds.) *Euro-Par 2021: Parallel Processing. Euro-Par 2021. Lecture Notes in Computer Science.* Vol. 12820. Springer, Cham; 2021. p. 501-516. (In Eng.) doi: https://doi.org/10.1007/978-3-030-85665-6_31

[21] Constantinescu D.A., Navarro A., Corbera F., et al. Efficiency and productivity for decision making on low-power heterogeneous CPU+GPU SoCs. *The Journal of Supercomputing*. 2021; 77(1):44-65. (In Eng.) doi: https://doi.org/10.1007/s11227-020-03257-3

[22] Pharr M. The ray-tracing engine that could: technical perspective. *Communications of the ACM*. 2013; 56(5):92. (In Eng.) doi: https://doi.org/10.1145/2447976.2447996

[23] Pharr M. Guest Editor's Introduction: Special Issue on Production Rendering. *ACM Transactions on Graphics*. 2018; 37(3):1-4. (In Eng.) doi: https://doi.org/10.1145/3212511

[24] Moreau P., Pharr M., Clarberg P. Dynamic many-light sampling for real-time ray tracing. *Proceedings of the Conference on High-Performance Graphics (HPG'19).* Eurographics Association, Goslar, DEU; 2019. p. 21-26. (In Eng.) doi: https://doi.org/10.2312/hpg.20191191

[25] Favre J.M., Blass A. A comparative evaluation of three volume rendering libraries for the visualization of sheared thermal convection. *Parallel Computing*. 2019; 88:102543. (In Eng.) doi: https://doi.org/10.1016/j.parco.2019.07.003

[26] Zhou K., et. al. Measurement and analysis of GPU-accelerated applications with HPCToolkit. *Parallel Computing.* 2021; 108:102837. (In Eng.) doi: https://doi.org/10.1016/j.parco.2021.102837

[27] Rodríguez A., et. al. Lightweight asynchronous scheduling in heterogeneous reconfigurable systems. *Journal of Systems Architecture.* 2022; 124:102398. (In Eng.) doi: https://doi.org/10.1016/j.sysarc.2022.102398

[28] Purkayastha A.A., Rogers S., Shiddibhavi S.A., Tabkhi H. LLVM-based automation of memory decoupling for OpenCL applications on FPGAs. *Microprocessors and Microsystems*. 2020; 72:102909. (In Eng.) doi: https://doi.org/10.1016/j.micpro.2019.102909

Поступила 10.04.2022; одобрена после рецензирования 27.05.2022; принята к публикации 15.06.2022.
Submitted 10.04.2022; approved after reviewing 27.05.2022; accepted for publication 15.06.2022.

Об авторах:

Владимиров Константин Игоревич, старший преподаватель кафедры микропроцессорных технологий в интеллектуальных системах, факультет радиотехники и кибернетики, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9), **ORCID: https://orcid.org/0000-0003-0925-1300**, konstantin.vladimirov@gmail.com

Андреев Илья Витальевич, магистрант кафедры микропроцессорных технологий в интеллектуальных системах, факультет радиотехники и кибернетики, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский переулок, д. 9), **ORCID: https://orcid.org/0000-0001-6450-7917,** andreev.iv@phystech.edu

Все авторы прочитали и одобрили окончательный вариант рукописи.

About the authors:

Konstantin I. Vladimirov, Senior Lecturer of the Chair of Microprocessor Technologies in Intelligent Systems, Department of Radio Engineering and Cybernetics, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation), **ORCID: https://orcid.org/0000-0003-0925-1300**, konstantin.vladimirov@gmail.com

Ilya V. Andreev, Master degree student of the Chair of Microprocessor Technologies in Intelligent Systems, Department of Radio Engineering and Cybernetics, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Dolgoprudny 141701, Moscow Region, Russian Federation), **ORCID: https://orcid.org/0000-0001-6450-7917,** andreev.iv@phystech.edu

All authors have read and approved the final manuscript.