УДК 004.4'422
DOI: 10.25559/SITITO.18.202202.256-262

Научная статья

Оптимизация условных переходов с учетом векторных возможностей потока управления Intel GPU

К. И. Владимиров*, Ю. В. Тарасов
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
При оптимизации программ для графических ускорителей и видеокарточек особую роль играют оптимизации потока управления. Кроме оптимизаций скалярного потока управления, широко известных и хорошо представленных в современных компиляторах, существует также актуальная проблема оптимизаций векторного потока управления. Векторный поток управления, с одной стороны, является естественным для языков высокого уровня, таких как ISPC и CM, где векторные управляющие конструкции являются частью семантики обычных программ. С другой стороны, векторные примитивы, в том числе для векторного потока управления представлены в современных графических ускорителях, например Intel XE. Поддержка в аппаратуре позволяет существенно улучшить производительность программ. Главной проблемой на этом пути является отсутствие векторных управляющих конструкций в стабильном промежуточном представлении. В этой работе предлагается промежуточное скалярное представление для векторных управляющих конструкций через явные предикаты и алгоритм восстановления векторного потока управления по этому представлению в графическом оптимизаторе.

Ключевые слова: компиляторы, оптимизации, графика, вектора, поток управления

Авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Владимиров К. И., Тарасов Ю. В. Оптимизация условных переходов с учетом векторных возможностей потока управления Intel GPU // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 2. С. 256-262. doi: https://doi.org/10.25559/SITITO.18.202202.256-262

© Владимиров К. И., Тарасов Ю. В., 2022

Контент доступен под лицензией Creative Commons Attribution 4.0 License.
The content is available under Creative Commons Attribution 4.0 License.

Original article

Conditional Jumps Optimization Taking into Account the Vector Capabilities of the Intel GPU Control Flow

K. I. Vladimirov*, Yu. V. Tarasov
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
Control flow optimizations are of particular importance when optimizing programs for graphics accelerators and video cards. In addition to scalar control flow optimizations, which are widely known and well represented in modern compilers, there is also a current issue of vector control flow optimizations. On the one hand, vector control flow is natural for high-level languages such as ISPC and CM, where vector control constructs are part of the semantics of ordinary programs. On the other hand, vector primitives, including those for vector control flow, are present in modern graphics accelerators, such as Intel XE. Support in the hardware can significantly improve the performance of programs. In this case, the main problem is the lack of vector control structures in a stable intermediate representation. This paper proposes an intermediate scalar representation for vector control structures through explicit predicates and an algorithm for restoring the vector control flow from this representation in a graphical optimizer.

Keywords: compilers, optimizations, graphics, vectors, control flow

The authors declare no conflict of interest.

For citation: Vladimirov K.I., Tarasov Yu.V. Conditional Jumps Optimization Taking into Account the Vector Capabilities of the Intel GPU Control Flow. *Sovremennye informacionnye tehnologii i IT-obrazovanie = Modern Information Technologies and IT-Education*. 2022; 18(2):256-262. doi: https://doi.org/10.25559/SITITO.18.202202.256-262

Введение

Графические компиляторы играют важную роль при оптимизации программ для видеокарт и графических ускорителей (будем коллективно называть их GPU) [1], [2] и при оптимизации гетерогенных вычислений [25], [26], [27]. При этом есть два больших класса входных контекстов. Это, во-первых, программы, написанные на языках программирования, оперирующих преимущественно скалярными конструкциями [3], [4]. Как пример таких языков можно указать GLSL, OpenCL C, HLSL, базовый DPC++, CUDA и список можно продолжать. В целом это золотой стандарт программирования для GPU. При оптимизации таких программ возникает много интересных задач, часть которых отлично изучены [5]-[8]. И, во-вторых, это программы, написанные на языках, оперирующих явными векторными конструкциями. Здесь примеров гораздо меньше, но часто они даже важнее для высокопроизводительных вычислений. В основном список таких языков сводится к ISPC [11], CM и DPC++ESIMD.

Одним из самых важных классов оптимизаций являются оптимизации потока управления [9], [10]. В скалярных архитектурах поток управления обычно реализован через инструкции условных переходов. При этом условие, от которого зависит переход, всегда также является скалярным. В отличие от этого в векторной модели может возникать векторное условие, верное для одних линий исполнения и неверное для других. При этом если для оптимизаций скалярного потока управления существует обширная литература [12]-[14] и поддержка в современных компиляторах, в том числе в LLVM [15], то векторный поток управления куда более сложен как концепция и гораздо хуже изучен. В основном в современных оптимизаторах его сводят к работе с масками, но этот подход очень ограничен и часто не использует всех возможностей аппаратуры. В архитектурах, не поддерживающих векторный поток управления, он действительно может быть сведен к скалярному путем маскированного исполнения всех инструкций дуги условного перехода или как минимум маскированного исполнения инструкций этой дуги, имеющих побочные эффекты (обычно это только записи в память).

В графических ускорителях Intel Xe и Intel ARC концепция векторных вычислений или SIMD-вычислений [2] (от Single Instruction Multiple Data) реализована на аппаратном уровне [16]. Широкая операция производится над всеми значениями в векторе за раз, и каждое значение в векторе вычисляется независимо [17]. Также на аппаратном уровне поддержан векторный поток управления, в частности включение и отключение векторных линий под определенными условиями.

Большой проблемой такого рода оптимизаций в компиляторах является тот факт, что в промежуточном представлении SPIRV, являющимся фактическим стандартом для графики, конструкций для векторного потока управления просто нет. Точно также таких конструкций нет в LLVM IR [18], [19].

В этой статье мы представим алгоритм, который унифицирует векторный поток управления для широкого класса векторных языков и передаёт его через скалярное промежуточное представление в таком виде, который позволяет его реконструкцию в графическом компиляторе в удобной для оптимизаций форме.

Аппаратные возможности для векторного потока управления

В графических ускорителях Intel XE есть возможность использовать векторный поток управления без сведения его к скалярному. Это реализовано через две ассемблерные инструкции: goto и join. Особенностью инструкции goto является возможность ее предикатирования (то есть исполнения в зависимости от предиката в специальном регистре). В этом случае исполнение программы произойдет так, что для предикатированных элементов вектора произойдет переход по указанной метке, а для всех остальных элементов произойдет переход на следующую инструкцию. Команда join синхронизирует исполнение инструкций. Можно думать о ней как о точке соединения потока управления и включения обратно части выключенных линий.

В аппаратуре подобная модель исполнения реализуется с помощью маски исполнения (*англ. execution mask*). Каждый бит этой маски соответствует элементу вектора. Если бит установлен в 0, то для соответствующего элемента вычисления не производятся, если же бит установлен в 1, то для соответствующего элемента исполнение происходит в обычном режиме. В самом начале программы маска состоит из единиц. Изменения в маске производят инструкции goto и join. Команда goto устанавливает в соответствии с предикатом биты маски в 0, выключая линии вектора, для которых условие не выполняется. Команда join восстанавливает маску до того состояния, в котором она находилась до изменения соответствующим goto.

Рассмотрим пример кода с векторным потоком управления на языке ассемблера для архитектуры графических ускорителей Intel. В данном примере реализована высокоуровневая конструкция if-else. Если элемент выбранного вектора r2.0<8;8,1>:d больше или равен нулю, то в вектор r3.0<1>:d он будет записан увеличенным на 1, в противном случае – уменьшенным на 1.

```
 (W) cmp (8|M0) (gt)f0.0 null<1>:d r2.0<8;8,1>:d -1:w
 (~f0. 0) goto (8|M0) BB_2 BB_2
BB_1:
 (W) add (8|M0) r3.0<1>:d r2.0<8;8,1>:d 1:w
 (f0.0) goto (8|M0) BB_2 BB_4
BB_2:
 join (8|M0) BB_2
BB_3:
 add  (8|M0) r3.0<1>:d r2.0<8;8,1>:d -1:w
BB_4:
 join (8|M0) BB_2
```

Листинг 1. Пример ассемблера Intel VISA
Listing 1. Intel VISA assembler example

	1	2	3	4	5	6	7	8
cmp								
goto								
add			X		X	X	X	
goto			X		X	X	X	
join	X	X	X	X	X	X	X	X
add	X	X		X				X
join	X	X		X				X

Р и с. 1. Схема исполнения примера (листинг 1)
F i g. 1. Diagram of an Example Implementation (Listing 1)

На (рис. 1) схематично изображено, как будет происходить исполнение кода из (листинга 1) операция за операцией сверху вниз. Цифрами сверху обозначены номера линий исполнения в SIMD-8 АЛУ, пустой клеткой обозначены включенные линии, а крестиком – выключенные.

Интерфейс между языками высокого уровня и векторным компилятором

Одним из языков, используемых для разработки программ, исполняемых на видеокартах, является ISPC (Implicit SPMD Program Compiler) [11][20]. Это язык программирования, являющийся расширением C и реализующий концепцию SPMD (single program, multiple data). ISPC для GPU является языком с раздельным исходным кодом, как OpenCL, то есть код исполняемого ядра отделен от API исполнителя [21][22]. Он широко применяется при трассировке лучей и рендеринге [23] и ориентирован на оптимизации LLVM [15][24]. Изначально целевой архитектурой для данного языка была только архитектура x86 с векторными расширениями, но позже была добавлена поддержка для ARM процессоров с расширениями Neon, а также графических ускорителей Intel XE.
Программа на языке ISPC является SPMD программой и компилируется для векторов заранее известной зафиксированной длины. В такой программе могут обрабатываться как общие для всех потоков данные, так и свои данные для каждого запущенного потока. Общие данные обозначают uniform. Такие переменные не изменяются в зависимости от векторной линии. Остальные по умолчанию считаются varying (то есть изменяющимися). Таким образом любую varying переменную можно рассматривать как элемент вектора.
На листинге 2 приведен пример функции на языке ISPC.

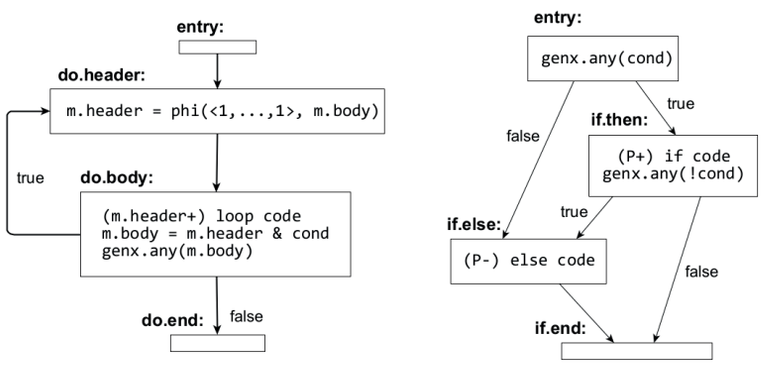
```
void simple(uniform float vin[],
        uniform float vout[],
        uniform int count) {
 foreach (index = 0 ... count) {
  float v = vin[index];
  v = (v < 3.)
    v = v * v;
  else
    v = sqrt(v);
  vout[index] = v;
 }
}
```

Листинг 2. Пример кода на ISPC с векторным потоком управления
Listing 2. ISPC Code Example with Vector Control Flow

В ISPC нет ограничений на векторный поток управления. В стандартных конструкциях языка он реализуется через использование в качестве условия varying переменной. При этом, так как ISPC является кроссплатформенным (и поддерживаемые им архитектуры различаются действительно кардинально) использование платформенно-специфичных интринсиков нежелательно. Вместо этого ISPC маскирует результаты операций после условных переходов для векторного потока управления. Для архитектуры графических ускорителей Intel такой подход не позволяет использовать всех векторных возможностей потока управления, аппаратуры.
В связи с этим нами предлагается интерфейс для языков высокого использующих векторный оптимизатор графического компилятора Intel. Так как интерфейс должен представлять собой конструкцию на промежуточном представлении LLVM IR, который не поддерживает векторный поток управления, неизбежно возникает необходимость сводить векторное условие к скалярному виду. Перед условным переходом необходимо проверить справедливо ли условие хоть для одного элемента и выполнять переход только в таком случае. Для наибольшей кроссплатформенности для подобных проверок следует использовать стандартные интринсики LLVM. Также для языков высокого уровня, поддерживающих только архитектуру графических ускорителей Intel как целевую (например, CM), допускается использовать для редукции условий платформенно зависимые интринсики. Для сохранения семантического смысла и поддержки платформ, на которых нет развитой поддержки векторного потока управления, необходимо маскировать условием побочные эффекты дуги перехода.
Данный подход также применим для скалярных архитектур, поддерживающих необходимые минимальные векторные расширения. При этом в процессе оптимизаций компилятор не сможет сделать трансформации, ломающие порядок побочных эффектов, поэтому такая конструкция будет валидна на всех этапах своего существования.

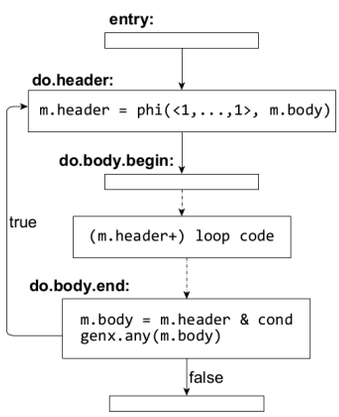
Оптимизация

Оптимизацией является фаза трансформации в векторном оптимизаторе графического компилятора Intel. Для упрощения проектирования, реализации и поддержки, оптимизация разделена на две части: анализ и трансформацию.
Перед рассмотрением анализа и трансформации, проведем обзор возможных конструкций, которые ожидаются на вход оптимизации.
Так как рассматривается только структурированный поток управления, то можно рассматривать только два атомарных случая: векторный условный переход и векторный цикл. На (рис. 2) представлены графы векторного потока управления для ветвления и цикла соответственно.



Р и с. 2. Схема потока управления для цикла и векторного условного оператора
F i g. 2. Control Flow Diagram for Loop and Vector Conditional Statement

В реальном коде может встречаться сложный поток управления, в котором базовые паттерны могут являться элементами других базовых паттернов, образовывая вложенные конструкции, например if-else вложенный в if-else или цикл с вложенным if-else.
Можно выделить регионы, содержащие в себе одну конструкцию векторного потока управления определенного уровня вложенности. Определим SIMD CF регион как регион, содержащий в себе одну конструкцию предложенного ранее интерфейса самого внешнего уровня вложенности, доступного для данного региона. Такой подход позволит упростить анализ, позволяя сначала обнаруживать самые внешние SIMD CF регионы, а после – вложенные. Тогда обобщенный SIMD CF регион для цикла будет выглядеть как на (рис. 3)



Р и с. 3. Схема обобщённого SIMDCF региона
F i g. 3. Diagram of a Generalized SIMDCF Region

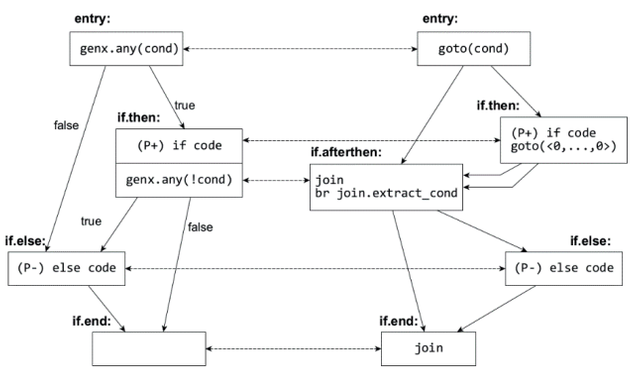
После определения SIMD CF региона можно свести задачу поиска заранее оговоренных конструкций к поиску SIMD CF регионов самого внешнего уровня вложенности, после чего искать вложенные SIMD CF регионы.
Шаг 1. Поиск условного перехода, похожего на векторный поток управления.
Для каждого базового блока проверяется его терминатор. Если это инструкция условного перехода, то проверяется его условие, в противном случае конструкция не является SIMD CF регионом. Если условием является результат вызова одного из обозначенных ранее интринсиков, то идет переход к шагу 2.
Шаг 2. Сопоставление с образцом.
Проверяется структура потока управления и происходит попытка сопоставить его либо с SIMD CF if/else, либо с SIMD CF циклом. Если сопоставление с одним из заданных паттернов невозможно, то конструкция не является SIMD CF регионом. В противном случае происходит переход к шагу 3.
Шаг 3. Проверка маскирования побочных эффектов.
Для каждой инструкции, у которой есть побочные эффекты проводится проверка, является ли такая инструкция маскирована и если она маскирована, то проверяется, совпадает ли маска для данной инструкции с условием перехода в эту дугу. Для вложенных регионов проверяется, является ли маска данного региона подмножеством маски внешнего региона. Если проверка неудачная - данный регион не является SIMD CF регионом.
Дополнение к шагу 3 для цикла. Проверяются фи-узлы для индуктивностей и пересчет маски для каждого цикла. Если проверка неудачная - данный регион не является SIMD CF регионом. В противном случае идет переход к шагу 4.
Дополнение к шагу 3 для if-else. Если кроме if также имеется else, то происходит проверка, являются ли маски if и else строго противоположны друг другу. Если проверка неудачная - данный регион не является SIMD CF регионом. В противном случае идет переход к шагу 4.
Шаг 4. Данный регион является SIMD CF регионом.
Рекурсивно осуществляется поиск вложенных SIMD CF регионов для данного региона.
В псевдокоде шаг данного алгоритма, который сопоставляет обобщённый цикл, будет выглядеть как на листинге 3.

```
procedure MatchLoop(BranchingBB) return SIMD_CF_Loop_Region
  BranchingBB: Basic_Block
begin
  loop: Loop
  loop := Get_Loop(BranchingBB)
  if !loop then
    return nil
  fi

  || рекурсивный вызов для определения регионов внутри цикла
    subregions append Find_SIMD_CF_Regions(loop.entering, loop.
exiting)
  return SIMD_CF_Loop_Region(loop, subregions)
end
```

Листинг. 3. Шаг алгоритма для выделения SIMDCF-цикла
Listing. 3. Algorithm step for highlighting a SIMDCF cycle

После сбора всей информации о SIMD CF регионах начинается их оптимизирующая трансформация. Для трансформирования разработанного интерфейса в уже имеющуюся модель goto-join. Для этого надо заменить интринсик llvm.vector.reduce.and или llvm.vector.reduce.and на интринсик, соответствующий инструкции goto для вычисления условия условного перехода, вставить соответствующий ему join, после чего убрать маскирование побочных эффектов.



Р и с. 4. Трансформация if-else
F i g. 4. Transformation if-else

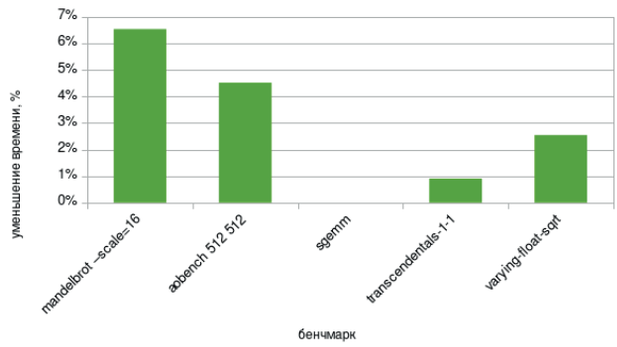
Результаты

В рамках этой работы был предложен алгоритм восстановления векторного потока управления из скалярного промежуточного представления для использования в графическом оптимизаторе Intel. Были проанализированы основные структуры потока управления и их представление в аппаратуре Intel Iris XE и Intel ARC.
Тестирование эффективности производилось на видеокарте Intel Iris XE на примерах и тестах из официального репозитория ISPC. Были выбраны следующие тесты: mandel, aobench,

sgemm из пакета xpu, а также transcendentials-1-1 varying-float-sqrt из базового набора. Тесты mandel и aobench запускались со следующими параметрами: mandelbrot –scale=16, и aobench 512 512. Остальные тесты запускались без дополнительных параметров.

На (рис. 5) представлено ускорение исполнения выбранных тестовых образцов в процентах.

Разработанная оптимизация открывает целый спектр дальнейших возможных улучшений, включая как более глубокие оптимизации векторного потока управления в компиляторе, так и алгоритмические оптимизации в пользовательском коде, нацеленном на использование векторных управляющих конструкций.



Р и с. 5. Результаты замера ускорения исполнения тестовых образцов, в процентах

F i g. 5. The Results of Measuring the Acceleration of the Execution of Test Samples, in Percentage Terms

References

[1] 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

[2] 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

[3] 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

[4] 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

[5] Vasudevan S. Inner loops in flowgraphs and code optimization. *Acta Informatica*. 1982; 17(2):143-155. (In Eng.) doi: https://doi.org/10.1007/BF00288967

[6] Sarkar V. Optimized Unrolling of Nested Loops. *International Journal of Parallel Programming*. 2001; 29(5):545-581. (In Eng.) doi: https://doi.org/10.1023/A:1012246031671

[7] Weiss S., Smith J.E. A Study of Scalar Compilation Techniques for Pipelined Supercomputers. *ACM Transactions on Mathematical Software*. 1990; 16(3):223-245. (In Eng.) doi: https://doi.org/10.1145/79505.79508

[8] Matoussi O., Pétrot F. Loop aware CFG matching strategy for accurate performance estimation in IR-level native simulation. *Integration*. 2019; 65:444-454. (In Eng.) doi: https://doi.org/10.1016/j.vlsi.2018.02.001

[9] Mansky W., Gunter E.L., Griffith D., Adams M.D. Specifying and executing optimizations for generalized control flow graphs. *Science of Computer Programming*. 2016; 130:2-23. (In Eng.) doi: https://doi.org/10.1016/j.scico.2016.06.003

[10] Carminati A., Starke R.A., de Oliveira R.S. Combining loop unrolling strategies and code predication to reduce the worst-case execution time of real-time software. *Applied Computing and Informatics*. 2017; 13(2):184-193. (In Eng.) doi: https://doi.org/10.1016/j.aci.2017.03.002

[11] 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

[12] Muntean P., Neumayer M., Lin Z., Tan G., Grossklags J., Eckert C. Analyzing control flow integrity with LLVM-CFI. *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC'19)*. Association for Computing Machinery, New York, NY, USA; 2019. p. 584-597. (In Eng.) doi: https://doi.org/10.1145/3359789.3359806

[13] Moll S., Hack S. Partial control-flow linearization. *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018)*. Association for Computing Machinery, New York, NY, USA; 2018. p. 543–556. (In Eng.) doi: https://doi.org/10.1145/3192366.3192413

[14] Wolf M.E., Maydan D.E., Chen D.K. Combining Loop Transformations Considering Caches and Scheduling. *International Journal of Parallel Programming*. 1998; 26(4):479-503. (In Eng.) doi: https://doi.org/10.1023/A:1018754616274

[15] 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

[16] 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

[17] 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

[18] 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

[19] Racordon D. From ASTs to Machine Code with LLVM. *Companion Proceedings of the 5th International Conference on the Art, Science, and Engineering of Programming (Programming '21)*. Association for Computing Machinery, New York, NY, USA; 2021. p. 68-76. (In Eng.) doi: https://doi.org/10.1145/3464432.3464777

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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

[25] Lee J., Hur C.-K., Jung R., Liu Z., Regehr J., Lopes N.P. Reconciling high-level optimizations and low-level code in LLVM. *Proceedings of the ACM on Programming Languages*. Vol. 2, No. OOPSLA. Association for Computing Machinery, New York, NY, USA; 2018. Article number: 125. p. 1-28. (In Eng.) doi: https://doi.org/10.1145/3276495

[26] 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

[27] Nozal R., Bosque J.L. Exploiting Co-execution with OneAPI: Heterogeneity from a Modern Perspective. In: Sousa L., Roma N., Tomás P. (eds.) E*uro-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

Поступила 13.04.2022; одобрена после рецензирования 27.05.2022; принята к публикации 15.06.2022.
Submitted 13.04.2022; 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

Тарасов Юлий Валерьевич, 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/0000-0003-0416-9703**, tarasov.iuv@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

Yuly V. Tarasov, 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/0000-0003-0416-9703**, tarasov.iuv@phystech.edu

All authors have read and approved the final manuscript.