DOI: 10.18721/JCSTCS.12403
УДК 004.4'416:004.382.2

# МЕТОД ГИБРИДНОГО НЕОДНОРОДНОГО ТАЙЛИНГА ДЛЯ АРХИТЕКТУР СУПЕРКОМПЬЮТЕРОВ С МНОГОУРОВНЕВОЙ ИЕРАРХИЕЙ ПАМЯТИ

*А.В. Левченко*

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

Предложен метод разбиения операций вычислительного алгоритма, основанный на выполнении преобразований в рамках полиэдральной модели компиляции в сочетании с классическим иерархическим параметризованным тайлингом для последующего преобразования кода, представленного в виде абстрактного синтаксического дерева. Разработана последовательность преобразований, позволяющих осуществить приоритизацию локальности на полиэдральной стадии, а на стадии многоуровневого тайлинга – сгенерировать тайлы на основе моделей для отображения на иерархическую архитектуру памяти. Построены производные алгоритмы акогерентного неоднородного тайлинга, расширяющие возможности предложенного метода посредством подстановки вариантов алгоритмов выбора формы и размера тайлов. Получены экспериментальные результаты, позволяющие оценить преимущества предложенного метода в контексте достижения переносимости производительности вычислительных алгоритмов на многомашинные макроузлы с глобально адресуемой когерентной памятью с неоднородным доступом.

**Ключевые слова:** иерархический тайлинг, компиляция, локальность, многоуровневая иерархия памяти, оптимизации цикла, переносимость производительности, полиэдральная модель.

**Ссылка при цитировании:** Левченко А.В. Метод гибридного неоднородного тайлинга для архитектур суперкомпьютеров с многоуровневой иерархией памяти // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2019. Т. 12. № 4. С. 29–44. DOI: 10.18721/JCSTCS.12403



# A HYBRID NON-UNIFORM LOOP TILING FOR SUPERCOMPUTERS WITH DEEP MEMORY HIERARCHIES

*A.V. Levchenko*

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

Non-uniform nature of multi-level memory architectures of modern supercomputers represents a notably underestimated issue in design of loop transformation algorithms. The imperfect compiler support for architectural features of

deep memory hierarchies results in insufficient data locality, which in turn is an obstacle to achieving performance portability for a wide range of important computational kernels like iterated stencils or sparse-matrices. The major contribution of this paper is an algorithmic skeleton for hybrid non-uniform loop tiling. The proposed approach combines locality-enhancing features of polyhedral compilation framework with capability of non-uniformity effects modeling via hierarchical parameterized tiling strategy performed in a canonical syntactic manner. The polyhedral stage focuses on spatial and temporal locality prioritization along with end-to-end optimization pipeline. At the syntactic stage the parameterized loop tiling strategy allows an automatic definition of tiled loop characteristics to map it according to the hierarchical memory architecture. The tiles with various parameters like size, shape and form can be generated through the novel permutational target-specific algorithms. As a result, the variants of acoherent non-uniform loop tiling algorithm were designed on the basis of the proposed approach to evaluate the permutational techniques of tile size and tile shape selection. The early-stage experimental results are presented to show the effects of hybrid non-uniform tiling on data locality to deliver near-optimal performance portability for representative benchmarks across deepening memory hierarchies of multi-machine macronodes.

**Keywords:** compiler optimizations, data locality, hierarchical tiling, loop optimizations, multi-level memory hierarchy, performance portability, polyhedral model.

**Citation:** Levchenko A.V. A hybrid non-uniform loop tiling for supercomputers with deep memory hierarchies. St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunications and Control Systems, 2019, Vol. 12, No. 4, Pp. 29–44. DOI: 10.18721/JCSTCS.12403



## Введение

Перспективным методом улучшения свойств вычислительных алгоритмов для эффективного использования многоуровневой иерархии памяти суперкомпьютеров является применение математической модели преобразований циклов, известной как полиэдральная модель компиляции. К замечательным свойствам полиэдрального подхода относится возможность реализации новых алгоритмов компиляции посредством композиции множества сложных автоматических преобразований-примитивов модели в виде единой алгебраической операции с возможностью последующего моделирования преобразований при помощи методов оптимизации [1].

В частности, одним из примитивов полиэдральной модели компиляции является метод разделения операций программной реализации вычислительного алгоритма на подмножества для реорганизации порядка вычислений: тайлинг. Тайлинг – фундаментальное преобразование для улучшения повторного использования данных при осуществлении доступа к регионам памяти, к которым доступ осуществлялся недавно, либо к соседствующим с ними регионам, что обуславливает временную и пространственную локальность вычислительных алгоритмов в неоднородной иерархической структуре памяти современных суперкомпьютеров [2].

Альтернативой тайлингу, выполняемому в рамках полиэдральной модели компиляции, является иерархический параметризованный тайлинг, основанный на применении корректных моделей параметров многоуровневой архитектуры памяти и использующий классический вариант преобразований вложенных циклов кода, представленного в виде абстрактного синтаксического дерева.

Несмотря на ряд полученных недавно результатов в изучении свойств полиэдрального и синтаксического подходов [3–8], на практике оказывается, что, будучи отдельно взятыми, эффекты этих подходов явля-

ются недостаточными [9]. Причина состоит в том, что существующие методы компиляции в должной степени не учитывают особые уровни неоднородности архитектуры и локальности доступа к памяти вычислительных алгоритмов при выполнении на реальных высокопроизводительных системах с глубокой иерархической памятью. В связи с этим для данных целевых архитектур должна быть решена задача построения усовершенствованных комбинированных методов тайлинга посредством использования преимуществ полиэдрального и синтаксического подходов.

Цель настоящего исследования – разработка метода гибридного неоднородного тайлинга, на основе которого будут построены алгоритмы преобразования циклов для достижения переносимости производительности вычислений на архитектуры суперкомпьютеров с многоуровневой иерархией памяти.

### Обзор литературы

Сформулируем особые требования к методам преобразований циклов, на основе которых будет выполнен анализ предметной области. Эти требования основаны на ряде недавних оценок положения дел в области высокопроизводительных вычислений [10–12] и, в частности, на прогнозе [13], согласно которому одним из принципов построения некоторых типов суперкомпьютеров экза- и зеттафлопсного уровня производительности станет наличие логически неделимой, глобально адресуемой многоуровневой архитектуры памяти, полученной путем объединения памяти вычислительных узлов, что неизбежно обусловит неудовлетворительную локальность широкого спектра научных вычислительных алгоритмов. Поэтому локальность становится главным фактором в модели для переносимости производительности [14] на системы данного класса. При этом системы с глобально адресуемой когерентной памятью с неоднородным доступом могут рассматриваться в качестве прототипа для гипотетических машин экзафлопсного уровня. Это следует из результатов работ [15–19], общей чертой которых является рассмотрение в таком качестве далеко не столь крупномасштабных систем с неоднородным доступом к памяти.

Таким образом, учитывая особенности целевой архитектуры, целесообразно сформулировать следующие требования к уже существующим методам автоматического преобразования циклов. Первое требование состоит в необходимости моделирования эффектов пространственной и временной локальности посредством применения алгоритмов планирования в полиэдральной модели компиляции [4]. Второе требование включает моделирование неоднородности посредством иерархического тайлинга. Этот метод предполагает параметризованное определение свойств атомарно выполняемых операций для отображения на архитектуру памяти в соответствии с конфигурацией кэш-памяти, узлов NUMA, буфера ассоциативной трансляции. Наконец, третье требование касается необходимости экспериментальной оценки производительности результирующего преобразованного кода с использованием реальной системы с многоуровневой иерархией памяти.

Полиэдральная модель компиляции рассматривается в ряде недавних работ [20–24] как алгоритмический аппарат, в рамках которого тайлинг может быть применен для улучшения локальности. Важным примером представляется работа [4], в которой предложен перспективный метод оптимизации пространственной и временной локальности систем с глубокой иерархией памяти посредством решения задачи целочисленного линейного программирования с использованием лексикографически минимального вектора пространства поиска. Результаты расширенного исследовательского отчета [25], предшествующего указанной работе, свидетельствуют о превосходстве предложенного алгоритма над существующими полиэдральными, синтаксическими и гибридными аналогами, воплощенными в современных оптимизато-

рах. Однако существуют и некоторые общие для полиэдральных техник ограничения, касающиеся преобразования статических областей управления, подробно описанные в [26], так же как и ограничения, связанные с возможным наличием скалярных зависимостей, рассмотренные в [27], и ряд других вопросов [28]. Кроме того, в [29] отмечена неудовлетворительная степень масштабируемости полиэдральной модели. На практике следствием перечисленных особенностей является недостаточность суперкомпьютерных полиэдральных тестов производительности: экспериментальные исследования обычно ограничиваются использованием набора программ PolyBench/C, примером чего являются многие из перечисленных работ.

Применение исключительно полиэдрального подхода представляется недостаточным для высокопроизводительных систем с глубокой иерархической памятью. В связи с этим в работе [30] была успешно проверена концепция комбинированного применения преобразований полиэдрального представления для локальности с последующими синтаксическими преобразованиями для повторного использования данных, межтайловой параллелизации и внутритайловой векторизации. Некоторые современные варианты синтаксического тайлинга рассмотрены в обзоре [2]. Среди перспективных возможностей можно допустить подстановку в разрабатываемый метод алгоритмов выбора размера тайла, предложенных в [31, 32]. В зависимости от специфических особенностей целевой архитектуры могут также применяться аналогичные процедуры для трафаретных вычислений, описанные в [33–35]. Наконец, решению проблемы неоднородности адресовано исследование [36], в котором предложен метод синтаксического неоднородного тайлинга, основанный на моделировании абстрактных уровней иерархии памяти. При этом данный метод не фокусируется на аспектах локальности доступа.

Обзор литературы привел к выводу о том, что рассмотренные подходы в полной мере не удовлетворяют всем сформулированным требованиям. В настоящем исследовании при разработке метода гибридного неоднородного тайлинга предлагается выполнить все требования за счет использования преимуществ существующих подходов.

### Метод гибридного неоднородного тайлинга

**Предварительные сведения.** Метод гибридного неоднородного тайлинга основан на выполнении преобразований в рамках полиэдральной модели компиляции в сочетании с синтаксическим иерархическим тайлингом. Главная цель разработки метода состоит в достижении переносимости производительности посредством улучшения локальности, важного свойства научных вычислительных алгоритмов. На основе концепции [30] разработана последовательность необходимых преобразований, позволяющих осуществить приоритизацию пространственно-временной локальности на полиэдральной стадии. Далее многоуровневый тайлинг осуществляется для последующего преобразования кода, представленного в виде абстрактного синтаксического дерева с генерацией тайлов необходимой размерности. Тайлы генерируются на основе уточненных моделей для отображения на неоднородную иерархическую архитектуру памяти.

Метод предназначен для улучшения локальности научных вычислительных алгоритмов, в частности, трафаретных вычислений (stencil computations). К данному классу относятся многосеточные методы и их компоненты: симметричный метод Гаусса-Зейделя, метод Гаусса-Якоби, метод последовательной верхней релаксации (SOR) и ряд других методов. Производительность трафаретных вычислений имеет значение для научных вычислений, поскольку данный класс задач лежит в основе решателей уравнений с частными производными. Так, репрезентативный тест производительности суперкомпьютеров HPCG (High Performance Conjugate Gradients) основан на решении системы линейных алгебраических уравнений с

разреженной симметричной матрицей методом сопряженных градиентов с предобуславливателем Гаусса-Зейделя.

Важной особенностью трафаретных вычислений является ограничение пропускной способности памяти ввиду низкого соотношения числа операций с плавающей точкой на байт информации, считываемой из памяти. Поэтому повышение производительности приложений этого класса неизбежно требует снижения перемещения данных из неоднородной памяти посредством увеличения их повторного использования. Таким образом, улучшение локальности трафаретных вычислений будет означать улучшение производительности для широкого спектра вычислительных алгоритмов при переносе на системы с глубокой иерархической памятью.

**Описание метода.** Метод гибридного неоднородного тайлинга предусматривает выполнение необходимых преобразований на протяжении двух стадий. На первой стадии решается задача улучшения локальности посредством полиэдральных преобразований. Вообще говоря, для иерархии памяти вычислительного узла цепочка преобразований могла быть ограничена только первой стадией, и тайлинг был бы осуществлен в рамках полиэдральной модели. Несмотря на это, именно вторая стадия является ядром предлагаемого метода.

Основная задача здесь состоит в обеспечении более эффективного использования глобально адресуемой когерентной памяти с неоднородным доступом. Для этого реализуется тайлинг, основанный на моделировании неоднородного доступа на разных уровнях иерархии памяти. В полной версии метода на первой стадии тайлинг только подготавливается посредством перестановки циклов и, в перспективе, посредством слияния циклов, но осуществляется уже на второй стадии. При этом применяется тайлинг в рамках синтаксического подхода как более простой и эффективный способ моделирования ряда аспектов архитектуры памяти, что неоднократно было отмечено в работах [30, 36]. Следует отметить, что в этом отношении потенциал метода раскрыт далеко не в полной мере. Рассмотрим возможности применения тайлинга в рамках предлагаемого метода.

Первая стадия метода предполагает наличие возможности применения тайлинга в рамках математического подхода полиэдральной компиляции. Тайлинг применяется к статическим областям управления (Static Control Parts – SCoPs), которые представляют собой максимальное множество последовательных выражений, где границы цикла и условные операторы являются аффинными функциями окружающих итераторов циклов и параметров, константные значения которых неизвестны на этапе компиляции [1].

После того, как статические области управления будут обнаружены в промежуточном представлении программы, происходит их преобразование в полиэдральное представление. Далее тайлинг осуществляется наряду с прочими преобразованиями за один проход компилятора как единая алгебраическая операция. Это дает возможность приоритизировать улучшение пространственной и временной локальности, моделируя тайлинг посредством численных методов оптимизации.

Как один из примитивов полиэдральной модели компиляции тайлинг применяется в полиэдральном представлении, основанном на геометрическом представлении итераций цикла, описанных в пространстве итераций в виде целочисленного полиэдроида. Полиэдроид охватывает множество точек в векторном пространстве $\mathbb{Z}^n$, удовлетворяющих неравенству

$$\mathcal{D} = \left\{ x \mid x \in \mathbb{Z}^n, Ax + a \geq 0 \right\}, \qquad (1)$$

где $x$ – вектор значений счетчика цикла; $A$ – константная матрица; $a$ – константный параметрический вектор.

Область итераций выражения является множеством его итерационных векторов, состоящих из значений итераторов цикла, содержащих выражение, и подмножеством

полного возможного пространства итераций $\mathcal{D} \subseteq \mathbb{Z}^n$ [1]. Область итераций $\mathcal{D}$ выражения $S$ представлена в качестве полиэдроида:

$$\mathcal{D}_S(\vec{p}) = \left\{ \vec{i}_S \mid D_S \begin{pmatrix} \vec{i}_S \\ \vec{p} \\ 1 \end{pmatrix} \geq \vec{0} \right\}, \qquad (2)$$

где $\vec{p}$ – вектор параметров; $\vec{i}_S \in \mathbb{Z}^{dim(\vec{i}_S)}$ – вектор итераций выражения $S$; $D_S \in \mathbb{Z}^{m_{D_S} \times (dim(\vec{i}_S)+dim(\vec{p})+1)}$ – целочисленная матрица с константами, где $m_{D_S}$ – число условий.

Данный подход позволяет применять методы линейной алгебры и линейного программирования для анализа и преобразования итерационных пространств в полиэдральном представлении. Таким образом, программа представлена в виде набора полиэдроидов, соответствующих множеству выражений $S_i \in S$, заключенных в одном или нескольких циклах. Порядок экземпляров выражений меняется посредством планирования и определяется отношениями планирования, когда каждому выражению присваивается логическая дата $\vec{t}_S$:

$$\Theta_S(\vec{p}) = \left\{ \vec{i}_S \to \vec{t}_S \mid T_S \begin{pmatrix} \vec{i}_S \\ \vec{t}_S \\ \vec{p} \\ 1 \end{pmatrix} \geq \vec{0} \right\}, \qquad (3)$$

где $T_s$ – целевое выражение [37].

Тайлинг осуществляется при условии, что функции планирования не имеют зависимостей, и порядок этих функций можно менять без искажения первоначальной семантики программы. Таким образом, при обнаружении групп взаимозаменяемых (допускающих перестановку) функций планирования осуществляется полное планирование и выполняется тайлинг. Преобразование посредством вставки копии взаимозаменяемой группы соответствующей размерности осуществляется непосредственно перед группой с установкой новой размерности. При этом существует возможность варьирования формы и размеров тайла. Для приоритизации локальности применяется метод лексикографической минимизации компонентов вектора пространства поиска [25], где для построения целевой функции используются отношения близости (пространственные и временные), т. к. расстояние между связанными экземплярами минимизируется для улучшения локальности обращения к памяти. Поскольку ожидается, что улучшение временной локальности даст больший прирост производительности, чем улучшение пространственной локальности, то в первую очередь минимизируются временные отношения близости, которые размещаются первыми.

Преобразования перестановки циклов и, в перспективе, слияние циклов используются для подготовки тайлинга на основе аналитической модели доступа к памяти для каждой итерации гнезда цикла для определенного уровня иерархии памяти:

$$MC(t_1, \ldots, t_n) = \\ = LC \times \frac{N(t_1, \ldots, t_n) \times M(t_1, \ldots, t_n)}{t_1 \times \ldots \times t_n}, \qquad (4)$$

где $LC$ – издержки промаха на строку в кэше и буфере ассоциативной трансляции; $N(t_1, \ldots, t_n)$ – функция размеров тайлов; $M(t_1, \ldots, t_n)$ – функция формы тайлов; $LC \times N(t_1, \ldots, t_n) \times M(t_1, \ldots, t_n)$ – издержки переноса всех данных цикла в кэш.

Выбор применяемых преобразований определяется данной моделью и самой возможностью таких преобразований [30]. Многоуровневый тайлинг для локальности может быть далее применен в рамках полиэдральной модели вместо перехода в синтаксическую стадию. Вследствие предполагаемой сложности реализации многоуровневого тайлинга в полиэдральном представлении, решение данной задачи оставлено для дальнейших исследований за рамками настоящей работы.

Вторая стадия метода основана на применении подхода иерархического параметризованного тайлинга для преобразования

кода, представленного в виде абстрактного синтаксического дерева. Иерархический тайлинг увеличивает сложность действий для преобразования тайлов, поскольку каждый уровень разбиения на тайлы имеет свои параметры выбора размера и формы тайлов, а особенности схемы разбиения на тайлы на разных уровнях вступают в конфликт. Это означает, что для минимизации времени выполнения простой выбор формы тайла для каждого уровня в отдельности не приведет к наилучшему выбору. Выбор формы тайла может быть осуществлен для множества уровней с помощью глобального иерархического тайлинга. Таким образом, генерируются размеры тайлов, которые являются символическими параметрами для различных конфигураций иерархической архитектуры памяти.

Эффективный набор тайлов с различными вариантами размеров и формы определяется в соответствии с иерархией кэш-памяти, конфигурацией узлов NUMA и буфера ассоциативной трансляции. Для существующих уровней эвристически определяется степень тайлинга, подходящая для снижения времени прямого доступа к памяти. Модель прямого доступа к памяти определена как

$$T_{i,j} = \alpha \times HL_{i,j} + \beta \times TS_{i,j} + \gamma \times TSH_{i,j}, \qquad (5)$$

где $HL_{I,j}$ – уровень иерархии с соответствующими издержками прямого доступа; $TS_{i,j}$ – размер тайла; $TSH_{i,j}$ – форма тайла в том случае, если она рассматривается; $\alpha$, $\beta$, $\gamma$ – положительные константы. Таким образом, необходимо определить размер и форму тайла для различных уровней иерархии в целях снижения издержек прямого доступа к памяти.

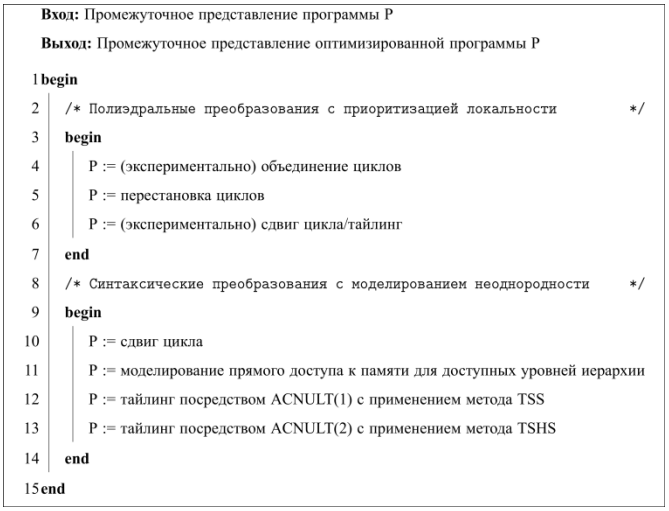
Из формулы, предложенной в работе [38], получаем издержки доступа для каждого уровня иерархической памяти:

$$HL_{i,j} = \sum_{i=1}^{N}\sum_{j=1}^{M} \sigma(i,j)\, j \left( T_{DMA} + \frac{block_i}{BW} \right). \qquad (6)$$

Здесь $\sigma(i, j)$ показывает, происходят ли промахи в кэш $j$ раз в $i$-м блоке данных; $block_i$ – размер $i$-го блока данных; $T_{DMA}$ – задержка времени доступа к удаленным данным; $BW$ – пропускная способность памяти при удаленном доступе. С учетом издержек доступа метод предполагает корректировку параметров тайла для различных уровней иерархии $HL_{i,j}$ посредством применения алгоритмов выбора размера тайла (Tile Size Selection – TSS) и формы тайла (Tile Shape Selection – TSHS).

**Построение производных алгоритмов.** Предложенный метод является своеобразным шаблоном, допускающим варианты корректировки таких параметров тайла, как размер и форма в зависимости от характеристик доступной иерархической памяти посредством применения подстановочных алгоритмов. Данная возможность предусмотрена в связи с постоянным совершенствованием алгоритмов класса TSS/TSHS. На основе описанного метода были построены варианты 1 и 2 алгоритма акогерентного неоднородного тайлинга (Acoherent Non-Uniform Loop Tiling – ACNULT).

Акогерентность в данном случае означает, что алгоритм полагается на аппаратную схему обеспечения когерентности кэшей и работает с глобально адресуемой общей памятью терабайтного объёма как с памятью обычного вычислительного узла. Варианты ACNULT(1,2) реализуют техники TSS и TSHS соответственно. Подстановочные алгоритмы поиска размеров и формы тайла в рамках алгоритмов ACNULT(1,2) представляют собой поиск по пространству, определяемому эксцентриситетом и обратным рабочим множеством, размер которого зависит от размера кэша. Порядок поиска определяется эвристикой, которая также используется для обрезки пространства поиска. Вопросы построения эффективных алгоритмов типа TSS/TSHS выходят за пределы настоящей статьи и в дальнейшем не рассматриваются. Выводы о степени сложности данных методов можно сделать из работ [31–35]. Общая схема предложенного метода демонстрируется на рис. 1.



**Вход:** Промежуточное представление программы P

**Выход:** Промежуточное представление оптимизированной программы P

```
1  begin
2    /* Полиэдральные преобразования с приоритизацией локальности        */
3    begin
4      P := (экспериментально) объединение циклов
5      P := перестановка циклов
6      P := (экспериментально) сдвиг цикла/тайлинг
7    end
8    /* Синтаксические преобразования с моделированием неоднородности    */
9    begin
10     P := сдвиг цикла
11     P := моделирование прямого доступа к памяти для доступных уровней иерархии
12     P := тайлинг посредством ACNULT(1) с применением метода TSS
13     P := тайлинг посредством ACNULT(2) с применением метода TSHS
14   end
15 end
```

Рис. 1. Метод гибридного неоднородного тайлинга с использованием вариантов ACNULT

Fig. 1. Overall algorithmic skeleton pipeline extended with ACNULT variants

Для реализации предложенного метода используется Polly, инструмент для полиэдральной оптимизации промежуточного представления инфраструктуры компиляторов LLVM [26]. Инструмент Polly применяется для обнаружения в канонизированном коде статических областей управления, которые можно преобразовать в полиэдральное представление и подвергнуть оптимизации. Преобразования в рамках полиэдрального подхода описаны в формате JSCoP, который используется для импорта и экспорта полиэдрального представления с измененными последовательностями выражений.

### Экспериментальные результаты

Цель эксперимента – проверка возможностей алгоритмов акогерентного неоднородного тайлинга ACNULT в отношении повышения эффективности тестовых полиэдральных программ на машинах с глубокой иерархией памяти.

Экспериментальная система включает набор многомашинных макроузлов с глобально адресуемой когерентной памятью с неоднородным доступом (Cache-Coherent Non-Uniform Memory Access – ccNUMA). Аппаратные конфигурации базовых узлов M0 и варианты построенных на их основе макроузлов M1, M2, M3 представлены в табл. 1. Каждый макроузел представляет собой кластер с общей памятью, при этом максимальные конфигурации макроузлов имеют ресурс процессоров и оперативной памяти, недоступный на узле стандартного вычислительного кластера общего назначения. Так, максимальный узел типа M3 способен предоставить более 12 ТБ RAM одновременно с 3072 ядер процессоров под управлением единственного экземпляра стандартной операционной системы на основе ядра Linux версии 4.14. Поскольку общий объём памяти ограничен только 48-битным диапазоном физических адресов, то общая возможность адресации составляет до 256 ТБ RAM. Глубокая иерархическая структура памяти строится посредством объединения базовых узлов типа M0 в макроузел посредством вычислительной сети с использованием протокола когерентности, основанного на стандарте Scalable Coherent Interface (SCI).

Таким образом, основным архитектурным принципом системы является наличие глобально адресуемого адресного пространства, что и позволяет процессорам осуществлять неоднородный доступ к удаленной памяти непосредственно. Акогерентный алгоритм будет рассматривать весь доступный объём памяти как память одного узла. Полный список уровней иерархии памяти экспериментального макроузла представлен в табл. 2.

Таблица 1

**Варианты конфигураций экспериментальных макроузлов**

Table 1

**Target macronode configurations**

| Особенности архитектуры | Базовый узел (M0) | Варианты конфигурации макроузлов | | |
|---|---|---|---|---|
| | | Минимальный (M1) | Средний (M2) | Максимальный (M3) |
| RAM | 188 Гбайт | 752 Гбайт | 3 ТБ | 12 ТБ |
| Узлы NUMA | 6 | 24 | 96 | 384 |
| Плата/Сокет/Ядра | 1/3/48 | 4/12/192 | 16/48/768 | 64/192/3072 |

Таблица 2

**Доступные уровни иерархии для моделирования локальности**

Table 2

**Available levels of deep memories to directly target locality**

| Уровни | Размер | Определение |
|---|---|---|
| L1d cache | 16 K | Кэш данных |
| L1i cache | 64 K | Кэш инструкций |
| L2 cache | 2018 K | Кэш второго уровня |
| L3 cache | 6144 K | Общий кэш для ядер, принадлежащих узлу NUMA |
| NUMA 0 | 32 Гбайт | Собственный узел NUMA |
| NUMA 1 | 32 Гбайт | Соседний узел NUMA по CPU |
| NUMA 2-5 | До 188 Гбайт | NUMA-узлы процессоров внутри одного базового узла |
| L4 cache | 1-16 Гбайт | Конфигурируемый кэш для удаленных данных |
| NUMA 6-384 | До 12 ТБ | NUMA-узлы соседних базовых узлов |

Первая стадия экспериментальной проверки возможностей алгоритмов ACNULT содержит оценку эффективности преобразованного кода для задач класса трафаретных вычислений из набора тестовых программ PolyBench/C, который обычно используется для оценки полиэдральных оптимизаторов [4, 26, 27]. Избранные задачи данного класса включают неявный метод переменного направления для двумерной диффузии тепла adi, метод конечных разностей во временной области для двумерных данных fdtd-2d, решение уравнения теплопроводности на трёхмерном пространстве heat-3d. Кроме того, используются трафаретные вычисления методом Якоби на одномерных или двумерных данных с использованием трёхточечного или пятиточечного шаблона jacobi-1d и jacobi-2d, а также вычисления методом Гаусса-Зейделя по двумерным данным с использованием шаблона с девятью точками seidel-2d. Данные программы использовались с максимальными настройками размеров данных LARGE и EXTRALARGE для создания репрезентативных настроек на узле типа M0. Иерархия памяти включает уровни кэшей L1d/L1i/L2/L3 наряду с трёхуровневой иерархией узлов NUMA и с конфигурируемой памятью L4 для обеспечения работы протокола когерентности кэшей.

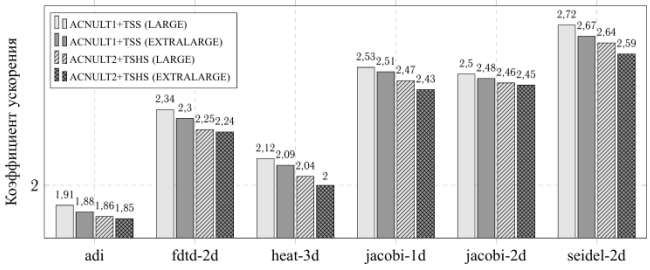



Рис. 2. Ускорение трафаретных вычислений PolyBench/C при использовании вариантов ACNULT

Fig. 2. Speedup with ACNULT on PolyBench/C stencils

На рис. 2 результаты алгоритмов ACNULT(1,2) сопоставляются с результатами кода, полученного только с параметром компилятора clang -O3 без использования Polly. Улучшения при использовании Polly здесь обусловлены применением тайлинга и выбором алгоритмов TSS/TSHS для подстановки в алгоритм ACNULT. Для случая более крупных узлов ускорение будет определяться в основном посредством выбора алгоритмов TSS. Результаты EXTRALARGE близки к результатам LARGE, что объясняется использованием тестовыми задачами недостаточных размеров данных в случае максимальной настройки EXTRALARGE, в результате чего демонстрируется некоторое ускорение.

Вторая стадия экспериментальной проверки метода включает исследование локальности как фактора переносимости производительности. Проблема переносимости производительности состоит в недостаточной эффективности вычислительного алгоритма при компиляции и выполнении на архитектурах, созданных с использованием принципов, отличных от исходных подходов. Для оценки переносимости производительности целесообразно использовать прокси-программы, которые обычно применяются при суперкомпьютерном кодизайне микроархитектуры и компилятора [39]. Переносимость производительности оптимизированной прокси-программы может быть определена в соответствии с моделью, ранее предложенной в [14], и интерпретированной в [9] специально для случая множества конфигураций макроузлов $|M|$, каждая последующая из которых содержит в себе предыдущие и, таким образом, иерархия памяти углубляется по мере их укрупнения:

$$PP_{m-node,\,opt,\,proxy} = \begin{cases} \dfrac{|M|}{\sum\limits_{i \in M} \dfrac{1}{T(opt,\,proxy)}}, & if \ \ \forall i \in M, \\ 0 & otherwise, \end{cases} \qquad (7)$$

где $T(opt, proxy)$ – время выполнения оптимизированной прокси-программы для макроузла $i$.

Переносимость производительности будет достигнута лишь в том случае, когда улучшение производительности оптимизированной прокси-программы будет зафиксировано на всех целевых машинах. В данной модели локальность может быть рассмотрена как основной фактор, влияющий на переносимость производительности. Метод оценки локальности рассмотрен в работе [9]. Для оценки улучшения локальности применяется программа Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH), представляющая собой суррогатный код, репрезентативный широкому спектру научных вычислительных задач в области гидродинамики, выполняемых на суперкомпьютерах. LULESH является научной прокси-программой, что предполагает наличие ряда алгоритмически эквивалентных реализаций данной программы с использованием множества перспективных моделей параллельных вычислений (MPI/OpenMP, Charm++, Chapel, MPI/UPC++, X10, etc.), а также наличие результатов оценки произво-

дительности на таких целевых архитектурах суперкомпьютеров, как BlueGene/Q, Cray XE6, POWER9, AMC [39].

Формула (8) ассоциирует вычислительные прокси-процедуры, из которых состоит прокси-программа, с набором преобразований кода [9]. Время выполнения прокси-программы определяется как

$$T = \sum_{i=1}^{n} Kernel_{i,opt,proxy} T_{i,m-node,opt}, \qquad (8)$$

где $Kernel_{i,opt,proxy}$ – число оптимизируемых прокси-процедур; $T_{i,m-node,opt}$ – время выполнения прокси-процедуры $i$ на макроузле после применения преобразования *opt*.

Код LULESH можно рассматривать как в качестве прокси-программы, так и в качестве полиэдрального теста производительности для суперкомпьютеров. Для этого используется реализация LULESH, основанная на использовании модели параллельных вычислений OpenMP, что является следствием архитектурных особенностей макроузлов, тяготеющих к укрупнению (от конфигурации M1 до M3). При этом реализация OpenMP наряду с версией MPI наиболее часто применяется в области высокопроизводительных вычислений и поэтому имеет множество опубликованных результатов для других высокопроизводительных проблемно-ориентированных архитектур, что позволяет сопоставить эти системы.

Запуск более тысячи потоков OpenMP внутри одного узла типа M3 представляет собой стресс-тест для многомашинной системы с общей памятью [9]. Однако, в отличие от тестов PolyBench/C, изначально разработанных для определения SCoP, LULESH содержит более сложные SCoP, которые могут быть сформированы посредством преобразования из наиболее трудоемких параллельных регионов OpenMP. Таким образом, полученные статические области управления могут быть преобразованы в полиэдральное представление при условии, что множественные избыточные зависимости между различными операторами, содержащиеся в SCoP, будут устранены.

Результаты оценки пространственной локальности LULESH после применения алгоритмов ACNULT(1,2) относительно локальности первоначальной версии представлены на рис. 3. Многопоточная реализация LULESH использована с размером задачи $30^3$...$90^3$ на макроузлах с 752 Гбайт, 3 ТБ, 12 ТБ глобально адресуемой памяти. Видно, что расслоение поверхностей наблюдается по мере углубления иерархической памяти от узла 3 ТБ и далее для всех размеров задач. Вариант ACNULT1 использует технику TSS. Данный вариант представляется более эффективным по сравнению с ACNULT2+TSHS. Улучшение локальности сохраняется по мере углубления памяти макроузлов для двух вариантов алгоритма. Кроме того, были улучшены характеристики локальности LULESH при увеличении размера задачи до значения $90^3$ сложности. В результате при всех размерах задачи LULESH локальность при использовании ACNULT(1,2) оказалась лучше локальности неоптимизированного кода при использовании наиболее крупномасштабного макроузла M3 с наибольшей степенью неоднородности памяти, что, согласно формуле (7), свидетельствует о достижении переносимости производительности на все целевые машины.

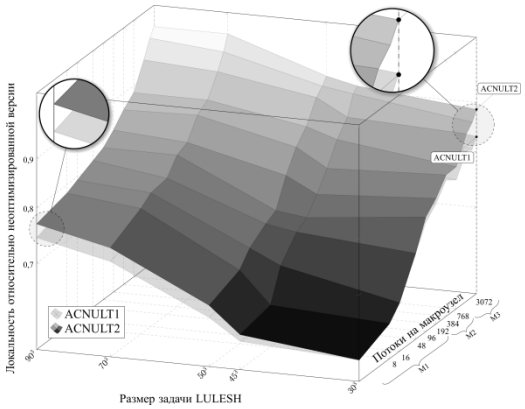



Рис. 3. Локальность многопоточной реализации LULESH для задачи $30^3$...$90^3$ на макроузлах с 752 Гбайт, 3 ТБ, 12 ТБ RAM

Fig. 3. Locality results for multithreaded LULESH with $30^3$...$90^3$ sized mesh on the macronodes with 752 Gb, 3 Tb, 12 Tb RAM

## Заключение

В статье предложен метод гибридного неоднородного тайлинга для архитектур суперкомпьютеров с многоуровневой иерархией памяти. Данный метод основан на комбинировании полиэдрального и синтаксического подходов автоматического преобразования циклов. Разработана последовательность преобразований, преимущества которой перед существующими подходами состоят в возможности осуществить приоритизацию локальности на полиэдральной стадии и моделировать аспекты неоднородности целевой архитектуры на стадии синтаксического иерархического тайлинга.

Построены алгоритмы акогерентного неоднородного тайлинга, позволяющие расширить область применения предлагаемого метода посредством использования перспективных подстановочных техник для тайлинга. Получены экспериментальные результаты построенных алгоритмов для повышения эффективности трафаретных вычислений и для улучшения локальности полиэдральной прокси-программы на макроузлах с глобально адресуемой когерентной памятью с неоднородным доступом.

Дальнейшие исследования будут направлены на совершенствование методов многоуровневого тайлинга в рамках полиэдральной модели компиляции.

## СПИСОК ЛИТЕРАТУРЫ

1. **Benabderrahmane M., Pouchet L., Cohen A., et al.** The polyhedral model is more widely applicable than you think // Proc. of the 19th Joint European Conf. on Theory and Practice of Software, Internat. Conf. on Compiler Construction. 2010. Pp. 283–303. DOI: 10.1007/978-3-642-11970-5_16

2. **Hammami E., Slama Y.** An overview on loop tiling techniques for code generation // 2017 IEEE/ACS 14th Internat. Conf. on Computer Systems and Applications. 2017. Pp. 280–287. DOI: 10.1109/AICCSA.2017.168

3. **Wijesinghe T., Senevirathne K., Siriwardhana C., et al.** Parameterized diamond tiling for parallelizing stencil computations // Moratuwa Engineering Research Conf. 2017. Pp. 99–104. DOI: 10.1109/MERCON.2017.7980464

4. **Zinenko O., Verdoolaege S., Reddy C., et al.** Modeling the conflicting demands of parallelism and temporal/spatial locality in affine scheduling // Proc. of the 27th Internat. Conf. on Compiler Construction. 2018. Pp. 3–13. DOI: 10.1145/3178372.3179507

5. **Zinenko O., Chelini L., Grosser T.** Declarative transformations in the polyhedral model. Research report RR-9243. Inria, ENS Paris – Ecole Normale Superieure de Paris, ETH Zurich, TU Delft, IBM Zurich, 2018 // URL: https://hal.inria.fr/hal-01965599 (Дата обращения: 30.04.2019).

6. **Zinenko O., Huot S., Bastoul C.** Visual program manipulation in the polyhedral model // ACM Transactions on Architecture and Code Optimization. 2018. Vol. 15. No. 1. Pp. 1–25. DOI: 10.1145/3177961

7. **Li F., Qiu K., Zhao M., et al.** Checkpointing-aware loop tiling for energy harvesting powered nonvolatile processors // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2019. Vol. 38. No. 1. Pp. 15–28. DOI: 10.1109/TCAD.2018.2803624

8. **Tanase A., Hannig F., Teich J.** Symbolic parallelization. Symbolic parallelization of nested loop programs. 2018. Pp. 37–92. DOI: 10.1007/978-3-319-73909-0_3

9. **Levchenko A.** Exploring trade-offs of compiler optimizations to enable performance portability for multi-level memory hierarchies // Proc. of the 4rd Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists. 2018. Pp. 85–97. Available: http://ceur-ws.org/Vol-2281/ (Accessed: 30.04.2019)

10. **Zhai J., Chen W.** A vision of post-exascale programming // Frontiers of Information Technology and Electronic Engineering. 2018. Vol. 19. No. 10. Pp. 1261–1266. DOI: 10.1631/FITEE.1800442

11. **Yang G., Fu H.** Application software beyond exascale: challenges and possible trends // Frontiers of Information Technology and Electronic Engineering. 2018. Vol. 19. No. 10. Pp. 1267–1272. DOI: 10.1631/FITEE.1800459

12. **Chen Z., Dongarra J., Xu Z.** Post-exascale supercomputing: research opportunities abound // Frontiers of Information Technology and Electronic Engineering. 2018. Vol. 19. No. 10. Pp. 1203–1208. DOI: 10.1631/FITEE.1830000

13. **Eisymont L.** Hybrid strategy development of the supercomputer components // Open Systems. DBMS. 2017. Vol. 25. No. 2. Pp. 8–11.

14. **Pennycook S., Sewall J., Lee V.** Implications of a metric for performance portability // Future

Generation Computer Systems. 2019. Vol. 92. Pp. 947–958. DOI: 10.1016/J.FUTURE.2017.08.007

15. **Zounmevo J., Perarnau S., Iskra K., et al.** A container-based approach to OS specialization for exascale computing // 2015 IEEE Internat. Conf. on Cloud Engineering. 2015. Pp. 359–364. DOI: 10.1109/IC2E.2015.78

16. **Ashraf R., Gioiosa R., Kestor G., et al.** Exploring the effect of compiler optimizations on the reliability of HPC applications // 2017 IEEE International Parallel and Distributed Processing Symp. Workshops. 2017. Pp. 1274–1283. DOI: 10.1109/IPDPSW.2017.7

17. **Perarnau S., Zounmevo J., Dreher M., et al.** Argo NodeOS: toward unified resource management for exascale // 2017 IEEE Internat. Parallel and Distributed Processing Symp. 2017. Pp. 153–162. DOI: 10.1109/IPDPS.2017.25

18. **Zou P., Allen T., Davis C., et al.** CLIP: cluster-level intelligent power coordination for power-bounded systems // 2017 IEEE Internat. Conf. on Cluster Computing. 2017. Pp. 541–551. DOI: 10.1109/CLUSTER.2017.98

19. **Goglin B.** Memory footprint of locality information on many-core platforms // 2018 IEEE Internat. Parallel and Distributed Processing Symp. Workshops. 2018. Pp. 1283–1292. DOI: 10.1109/IPDPSW.2018.00201

20. **Rodriguez G., Kandemir M., Tourino J.** Affine modeling of program traces // IEEE Transactions on Computers. 2019. Vol. 68. No. 2. Pp. 294–300. DOI: 10.1109/TC.2018.2853747

21. **Siklosi B., Reguly I., Mudalige G.** Heterogeneous CPU-GPU execution of stencil applications // 2018 IEEE/ACM Internat. Workshop on Performance, Portability and Productivity in HPC. 2018. Pp. 71–80. DOI: 10.1109/P3HPC.2018.00010

22. **Simburger A., Apel S., Grosslinger A., et al.** PolyJIT: polyhedral optimization just in time // Internat. J. of Parallel Programming, 2018. DOI: 10.1007/S10766-018-0597-3

23. **Bielecki W., Palkowski M., Skotnicki P.** Generation of parallel synchronization-free tiled code // Computing. 2018. Vol. 100. No. 3. Pp. 277–302. DOI: 10.1007/S00607-017-0576-3

24. **Liu S., Cui Y., Zou N., et al.** Revisiting the parallel strategy for DOACROSS loops // J. of Computer Science and Technology. 2019. Vol. 34. No. 2. Pp. 456–475. DOI: 10.1007/S11390-019-1919-7

25. **Zinenko O., Verdoolaege S., Reddy C., et al.** Unified polyhedral modeling of temporal and spatial locality. Research report RR-9110. Inria Paris, 2017 // URL: https://hal.inria.fr/hal-01628798 (Дата обращения: 30.04.2019).

26. **Grosser T., Grosslinger A., Lengauer C.** Polly – performing polyhedral optimizations on a low-level intermediate representation // Parallel Processing Letters. 2012. Vol. 22. No. 04. DOI: 10.1142/S0129626412500107

27. **Kruse M., Grosser T.** DeLICM: scalar dependence removal at zero memory cost // Proc. of the 2018 Internat. Symp. on Code Generation and Optimization. 2018. Pp. 241–253. DOI: 10.1145/3168815

28. **Acharya A., Bondhugula U., Cohen A.** Polyhedral auto-transformation with no integer linear programming // Proc. of the 39th ACM SIGPLAN Conf. on Programming Language Design and Implementation. 2018. Vol. 53. No. 4. Pp. 529–542. DOI: 10.1145/3192366.3192401

29. **Pradelle B., Meister B., Baskaran M., et al.** Scalable hierarchical polyhedral compilation // 45th Internat. Conf. on Parallel Processing. 2016. Pp. 432–441. DOI: 10.1109/ICPP.2016.56

30. **Shirako J., Pouchet L., Sarkar V.** Oil and water can mix: an integration of polyhedral and AST-based transformations // SC14: Proc. of the Internat. Conf. for High Performance Computing, Networking, Storage and Analysis. 2014. Pp. 287–298. DOI: 10.1109/SC.2014.29

31. **Liu J., Wickerson J., Constantinides G.** Tile size selection for optimized memory reuse in high-level synthesis // 27th Internat. Conf. on Field Programmable Logic and Applications. 2017. Pp. 1–8. DOI: 10.23919/FPL.2017.8056810

32. **Chen Y., Shang W., Fang Y.** Supernode transformation on computer clusters // 10th Internat. Conf. on Ubi-media Computing and Workshops. 2017. Pp. 1–6. DOI: 10.1109/UMEDIA.2017.8074109

33. **Bondhugula U., Bandishti V., Pananilath I.** Diamond Tiling: tiling techniques to maximize parallelism for stencil computations // IEEE Transactions on Parallel and Distributed Systems. 2017. Vol. 28. No. 5. Pp. 1285–1298. DOI: 10.1109/TPDS.2016.2615094

34. **Reguly I., Mudalige G., Giles M.** Loop tiling in large-scale stencil codes at run-time with OPS // IEEE Transactions on Parallel and Distributed Systems. 2018. Vol. 29. No. 4. Pp. 873–886. DOI: 10.1109/TPDS.2017.2778161

35. **Rawat P., Vaidya M., Sukumaran-Rajam A., et al.** Domain-specific optimization and generation of high-performance GPU code for stencil computations // Proc. of the IEEE. 2018. Vol. 106. No. 11. Pp. 1902–1920. DOI: 10.1109/JPROC.2018.2862896

36. **Qiu K., Ni Y., Zhang W., et al.** An adaptive non-uniform loop tiling for DMA-based bulk data transfers on many-core processor // 2016 IEEE

34th Internat. Conf. on Computer Design. 2016. Pp. 9–16. DOI: 10.1109/ICCD.2016.7753255

37. **Razanajato H., Bastoul C., Loechner V.** Lifting barriers using parallel polyhedral regions // IEEE 24th Internat. Conf. on High Performance Computing. 2017. Pp. 338–347. DOI: 10.1109/HIPC.2017.00046

38. **Zhang M., Gu N., Ren K.** Optimization of computation-intensive applications in cc-NUMA architecture // Internat. Conf. on Networking and Network Applications. 2016. Pp. 244–249. DOI: 10.1109/NaNA.2016.12

39. **Jacob A., Nair R., Chen T., et al.** Progressive codesign of an architecture and compiler using a proxy application // 27th Internat. Symp. on Computer Architecture and High Performance Computing. 2015. Pp. 57–64. DOI: 10.1109/SBAC-PAD.2015.18

*Статья поступила в редакцию 30.04.2019.*

## REFERENCES

1. **Benabderrahmane M., Pouchet L., Cohen A., et al.** The polyhedral model is more widely applicable than you think. *Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction*, 2010, Pp. 283–303. DOI: 10.1007/978-3-642-11970-5_16

2. **Hammami E., Slama Y.** An overview on loop tiling techniques for code generation. *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications*, 2017, Pp. 280–287. DOI: 10.1109/AICCSA.2017.168

3. **Wijesinghe T., Senevirathne K., Siriwardhana C., et al.** Parameterized diamond tiling for parallelizing stencil computations. *Moratuwa Engineering Research Conference*, 2017, Pp. 99–104. DOI: 10.1109/MERCON.2017.7980464

4. **Zinenko O., Verdoolaege S., Reddy C., et al.** Modeling the conflicting demands of parallelism and Temporal/Spatial locality in affine scheduling. *Proceedings of the 27th International Conference on Compiler Construction*, 2018, Pp. 3–13. DOI: 10.1145/3178372.3179507

5. **Zinenko O., Chelini L., Grosser T.** Declarative transformations in the polyhedral model. Research report RR-9243. Inria, ENS Paris – Ecole Normale Superieure de Paris, ETH Zurich, TU Delft, IBM Zurich, 2018. *Available: https://hal.inria.fr/hal-01965599* (Accessed: 30.04.2019).

6. **Zinenko O., Huot S., Bastoul C.** Visual program manipulation in the polyhedral model. *ACM Transactions on Architecture and Code Optimization*, 2018, Vol. 15, No. 1, Pp. 1–25. DOI: 10.1145/3177961

7. **Li F., Qiu K., Zhao M., et al.** Checkpointing-aware loop tiling for energy harvesting powered nonvolatile processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019, Vol. 38, No. 1, Pp. 15–28. DOI: 10.1109/TCAD.2018.2803624

8. **Tanase A., Hannig F., Teich J.** *Symbolic parallelization. Symbolic parallelization of nested loop programs*, 2018, Pp. 37–92. DOI: 10.1007/978-3-319-73909-0_3

9. **Levchenko A.** Exploring trade-offs of compiler optimizations to enable performance portability for multi-level memory hierarchies. *Proceedings of the 4rd Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists*, 2018, Pp. 85–97. Available: http://ceur-ws.org/Vol-2281/ (Accessed: 30.04.2019).

10. **Zhai J., Chen W.** A vision of post-exascale programming. *Frontiers of Information Technology and Electronic Engineering*, 2018, Vol. 19, No. 10, Pp. 1261–1266. DOI: 10.1631/FITEE.1800442

11. **Yang G., Fu H.** Application software beyond exascale: challenges and possible trends. *Frontiers of Information Technology and Electronic Engineering*, 2018, Vol. 19, No. 10, Pp. 1267–1272. DOI: 10.1631/FITEE.1800459

12. **Chen Z., Dongarra J., Xu Z.** Post-exascale supercomputing: research opportunities abound. *Frontiers of Information Technology and Electronic Engineering*, 2018, Vol. 19, No. 10, Pp. 1203–1208. DOI: 10.1631/FITEE.1830000

13. **Eisymont L.** Hybrid strategy development of the supercomputer components. *Open Systems. DBMS*, 2017, Vol. 25, No. 2, Pp. 8–11.

14. **Pennycook S., Sewall J., Lee V.** Implications of a metric for performance portability. *Future Generation Computer Systems*, 2019, Vol. 92, Pp. 947–958. DOI: 10.1016/J.FUTURE.2017.08.007

15. **Zounmevo J., Perarnau S., Iskra K., et al.** A container-based approach to OS specialization for exascale computing. *2015 IEEE International Conference on Cloud Engineering*, 2015, Pp. 359–364. DOI: 10.1109/IC2E.2015.78

16. **Ashraf R., Gioiosa R., Kestor G., et al.** Exploring the effect of compiler optimizations on the reliability of HPC applications. *2017 IEEE*

*International Parallel and Distributed Processing Symposium Workshops*, 2017, Pp. 1274–1283. DOI: 10.1109/IPDPSW.2017.7

17. **Perarnau S., Zounmevo J., Dreher M., et al.** Argo NodeOS: toward unified resource management for exascale. *2017 IEEE International Parallel and Distributed Processing Symposium*, 2017, Pp. 153-162. DOI: 10.1109/IPDPS.2017.25

18. **Zou P., Allen T., Davis C., et al.** CLIP: cluster-level intelligent power coordination for power-bounded systems. *2017 IEEE International Conference on Cluster Computing*, 2017, Pp. 541–551. DOI: 10.1109/CLUSTER.2017.98

19. **Goglin B.** Memory footprint of locality information on many-core platforms. *2018 IEEE International Parallel and Distributed Processing Symposium Workshops*, 2018, Pp. 1283–1292. DOI: 10.1109/IPDPSW.2018.00201

20. **Rodriguez G., Kandemir M., Tourino J.** Affine modeling of program traces. *IEEE Transactions on Computers*, 2019, Vol. 68, No. 2, Pp. 294–300. DOI: 10.1109/TC.2018.2853747

21. **Siklosi B., Reguly I., Mudalige G.** Heterogeneous CPU-GPU execution of stencil applications. *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC*, 2018, Pp. 71–80. DOI: 10.1109/P3HPC.2018.00010

22. **Simburger A., Apel S., Grosslinger A., et al.** PolyJIT: polyhedral optimization just in time. *International Journal of Parallel Programming*, 2018. DOI: 10.1007/S10766-018-0597-3

23. **Bielecki W., Palkowski M., Skotnicki P.** Generation of parallel synchronization-free tiled code. *Computing*, 2018, Vol. 100, No. 3, Pp. 277–302. DOI: 10.1007/S00607-017-0576-3

24. **Liu S., Cui Y., Zou N., et al.** Revisiting the parallel strategy for DOACROSS loops. *Journal of Computer Science and Technology*, 2019, Vol. 34, No. 2, Pp. 456–475. DOI: 10.1007/S11390-019-1919-7

25. **Zinenko O., Verdoolaege S., Reddy C., et al.** Unified polyhedral modeling of temporal and spatial locality. Research report RR-9110. Inria Paris, 2017. Available: *https://hal.inria.fr/hal-01628798* (Accessed: 30.04.2019).

26. **Grosser T., Grosslinger A., Lengauer C.** Polly – performing polyhedral optimizations on a low-level intermediate representation. *Parallel Processing Letters*, 2012, Vol. 22, No. 04. DOI: 10.1142/S0129626412500107

27. **Kruse M., Grosser T.** DeLICM: scalar dependence removal at zero memory cost. *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, Pp. 241–253. DOI: 10.1145/3168815

28. **Acharya A., Bondhugula U., Cohen A.** Polyhedral auto-transformation with no integer linear programming. *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2018, Vol. 53, No. 4, Pp. 529–542. DOI: 10.1145/3192366.3192401

29. **Pradelle B., Meister B., Baskaran M., et al.** Scalable hierarchical polyhedral compilation. *2016 45th International Conference on Parallel Processing*, 2016, Pp. 432–441. DOI: 10.1109/ICPP.2016.56

30. **Shirako J., Pouchet L., Sarkar V.** Oil and water can mix: an integration of polyhedral and AST-based transformations. *SC14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, Pp. 287–298. DOI: 10.1109/SC.2014.29

31. **Liu J., Wickerson J., Constantinides G.** Tile size selection for optimized memory reuse in high-level synthesis. *2017 27th International Conference on Field Programmable Logic and Applications*, 2017, Pp. 1–8. DOI: 10.23919/FPL.2017.8056810

32. **Chen Y., Shang W., Fang Y.** Supernode transformation on computer clusters. *2017 10th International Conference on Ubi-media Computing and Workshops*, 2017, Pp. 1–6. DOI: 10.1109/UMEDIA.2017.8074109

33. **Bondhugula U., Bandishti V., Pananilath I.** Diamond tiling: tiling techniques to maximize parallelism for stencil computations. *IEEE Transactions on Parallel and Distributed Systems*, 2017, Vol. 28, No. 5, Pp. 1285–1298. DOI: 10.1109/TPDS.2016.2615094

34. **Reguly I., Mudalige G., Giles M.** Loop tiling in large-scale stencil codes at run-time with OPS. *IEEE Transactions on Parallel and Distributed Systems*, 2018, Vol. 29, No. 4, Pp. 873–886. DOI: 10.1109/TPDS.2017.2778161

35. **Rawat P., Vaidya M., Sukumaran-Rajam A., et al.** Domain-specific optimization and generation of high-performance GPU code for stencil computations. *Proceedings of the IEEE*, 2018, Vol. 106, No. 11, Pp. 1902–1920. DOI: 10.1109/JPROC.2018.2862896

36. **Qiu K., Ni Y., Zhang W., et al.** An adaptive non-uniform loop tiling for DMA-based bulk data transfers on many-core processor. *2016 IEEE 34th International Conference on Computer Design*, 2016, Pp. 9–16. DOI: 10.1109/ICCD.2016.7753255

37. **Razanajato H., Bastoul C., Loechner V.** Lifting barriers using parallel polyhedral regions. *2017 IEEE 24th International Conference on High Performance Computing*, 2017, Pp. 338–347. DOI: 10.1109/HIPC.2017.00046

38. **Zhang M., Gu N., Ren K.** Optimization of computation-intensive applications in cc-NUMA architecture. *2016 International Conference on Networking and Network Applications*, 2016, Pp. 244–249. DOI: 10.1109/NaNA.2016.12

39. **Jacob A., Nair R., Chen T., et al.** Progressive codesign of an architecture and compiler using a proxy application. *2015 27th International Symposium on Computer Architecture and High Performance Computing*, 2015, Pp. 57–64. DOI: 10.1109/SBAC-PAD.2015.18

*Received 30.04.2019.*

## СВЕДЕНИЯ ОБ АВТОРАХ / THE AUTHORS

**ЛЕВЧЕНКО Алексей Викторович**
**LEVCHENKO Aleksei V.**
E-mail: levchenko_av@spbstu.ru