УДК 004.94:004.8

ТЕСТОВАЯ ПРОГРАММА "ПАУК" КАК ПОЛИГОН ДЛЯ АПРОБАЦИИ АЛГОРИТМОВ И ТЕХНОЛОГИЙ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

А. А. Нуждин
(ФГУП "РФЯЦ-ВНИИЭФ", г. Саров Нижегородской области)

В тестовой программе ПАУК с помощью разностного S_n-метода численно решается трехмерное одногрупповое уравнение переноса нейтронов на ортогональной пространственной сетке. Представлены результаты исследований по адаптации тестовой программы к гетерогенной вычислительной системе, содержащей сопроцессоры Intel Xeon Phi поколения Knights Corner. При этом апробированы различные алгоритмы и технологии параллельного программирования: векторизация циклов по направлениям и элементам гиперплоскостей, автовекторизация и intrinsic-программирование, явная и неявная предвыборки данных, реализации KBA-алгоритма распараллеливания в трех моделях памяти (общая, распределенная, PGAS). Эффективность адаптации программы подтверждается результатами исследований производительности в различных режимах счета на гетерогенной вычислительной системе: на универсальных процессорах (CPU-only), сопроцессорах (native) и в симметричном режиме (symmetric).

Ключевые слова: S_n-метод, алгоритм бегущего счета, KBA-алгоритм, Intel Xeon Phi, векторизация, предвыборка данных, MPI-3 SHM.

Введение

Программа ПАУК предназначена для тестирования производительности суперЭВМ применительно к решению задач переноса частиц разностными методами [1]. В программе численно решается трехмерное стационарное одногрупповое кинетическое уравнение переноса частиц, записанное в декартовой системе координат. Используются характерные для данного типа приложений численные методы и алгоритмы: аппроксимация на основе схемы типа DS_n-метода, алгоритм бегущего счета и метод итераций по источнику. Язык программирования — Fortran-90. Первоначально программа ПАУК была ориентирована на тестирование производительности однородных суперЭВМ методом увеличения задачи, или в режиме слабой масштабируемости. Для этого в ней реализован с помощью стандарта MPI алгоритм распараллеливания, который основан на трехмерной пространственной декомпозиции, — модификация KBA-алгоритма [2]. Эффективность распараллеливания является основной метрикой для оценки производительности теста на однородных суперЭВМ. Программа ПАУК входит в набор методических прикладных тестов РФЯЦ-ВНИИЭФ [3] и является миниприложением для методики САТУРН [4].

Появление гетерогенных, или гибридных, суперЭВМ, содержащих в одном узле вычислительные устройства различной архитектуры (например, multi- и many-core устройства), потребовало разработки новых алгоритмов распараллеливания и внедрения новых техник параллельного программирования в интересах эффективного отображения приложения на такие архитектуры. В данном случае имеются в виду смешанное (MPI+X)-распараллеливание и дисциплина вычислений SIMD (или SIMT в случае GPU-устройств).

Важной особенностью тестовой программы ПАУК является ее компактность. Суммарный объем кода составляет $\sim 2\,000$ строк, при этом основное счетное ядро (решение уравнения баланса) составляет ~ 50 строк кода. Такая компактность является существенным достоинством

при решении задачи отображения приложения на новые архитектуры. В работе рассматривается архитектура Intel Many Integrated Core (MIC) на примере устройств Intel Xeon Phi поколения Knights Corner. Методология отображения на гетерогенную систему с Intel Xeon Phi основана на выделении иерархических уровней: ядро, кристалл, узел/множество узлов. Подходы к параллельному программированию взяты из работы [5]. Для простоты и удобства выбрана симметричная модель MPI-программирования.

Исследования выполнены на вычислительном кластере ООО "Центр компетенций и обучения", г. Саров [6], который содержит гетерогенный сегмент с пятью узлами: 2 CPU Intel Xeon E5-2680 v2 (22 нм, Ivy Bridge, 10 ядер, 2,8 ГГц, 0,224 Тфлопс) + 2 Intel Xeon Phi Coprocessor 7120P (22 нм, Knights Corner, 61 ядро, 1,24 ГГц, 1,21 Тфлопс). Коммуникационная система — infiniBand FDR, 56 Гбит/с. Пиковая производительность одного гетерогенного узла — 2,868 Тфлопс.

Основной результат данной работы заключается в сравнительных исследованиях производительности на Intel Xeon Phi поколения Knights Corner различных алгоритмов и техник параллельного программирования для программы решения трехмерного уравнения переноса частиц на ортогональной пространственной сетке.

Аналогичные работы

Результаты применения схожих методологии и приемов адаптации высокопроизводительных приложений из других предметных областей к архитектуре Intel MIC — различных подходов к векторизации циклов, предвыборки данных, (MPI+X)-распараллеливания, применения технологии MPI-3 SHM — изложены в сборниках [7].

В рассматриваемой предметной области наиболее близки к программе ПАУК тестовые приложения (miniapps или proxy) трех лабораторий Министерства энергетики США: LANL (Sweep3D и SNAP), LLNL (KRIPKE), ORNL (miniSweep). Во всех этих приложениях решается трехмерное уравнение переноса нейтронов методом дискретных ординат (S_n) на ортогональных пространственных сетках, используются аналогичные методы и алгоритмы, в том числе бегущий счет и KBA-алгоритм распараллеливания.

Sweep3D — это компактное приложение предыдущего поколения (сейчас заменено на SNAP), разработанное в рамках программы ASCI для тестирования производительности различных однородных суперЭВМ [8]. Тест Sweep3D применялся для апробации элементов модели PGAS за счет использования `coarray` в Fortran [9], а также для отображения на различные гетерогенные архитектуры: Cell Broadband Engine (Cell BE) [10], GPU [11], Intel MIC [12].

SNAP (SN (Discrete Ordinates) Application Proxy) — это прокси-приложение, специально предназначенное для тестирования гибридных или гетерогенных суперЭВМ [13]. Тест SNAP не решает задачу переноса нейтронов, он имитирует вычислительную нагрузку и алгоритмы большого приложения PARTISN [14]. Разработчики приложения ориентировались на сопроцессоры Intel Xeon Phi поколения Knights Landing [15]. Тест активно используется для апробации новых инструментов параллельного программирования [16] и отображения на различные архитектуры [17].

Тест KRIPKE создан специально для апробаций технологий программирования и различных алгоритмов отображения на архитектуру. Он является прокси-приложением для комплекса ARDRA [18].

MiniSweep создан с целью отображения на различные архитектуры и апробации различных моделей программирования [19]. Например, в работе [20] приводятся результаты распараллеливания miniSweep с помощью технологий CUDA, OpenMP и OpenACC, полученные на различных гетерогенных платформах, в том числе Intel Xeon Phi поколения Knights Landing. Тест является прокси-приложением кода DENOVO, который эффективно адаптирован к суперЭВМ с GPU Nvidia [21].

Постановка задачи

В тестовой программе ПАУК решается стационарное трехмерное уравнение переноса в одногрупповом кинетическом приближении в декартовой системе координат на ортогональных пространственных сетках [3]:

$$\operatorname{div}\left(\vec{\Omega}N\right) + \alpha N = \frac{1}{4\pi}\left(\beta n^{(0)} + Q\right), \quad (1)$$
$$\operatorname{div}\left(\vec{\Omega}N\right) = \Omega_x\frac{\partial N}{\partial x} + \Omega_y\frac{\partial N}{\partial y} + \Omega_z\frac{\partial N}{\partial z},$$

где α — коэффициент столкновения частиц; β — коэффициент размножения частиц; Q — независимый источник частиц; N — плотность потока частиц, летящих в направлении $\vec{\Omega}$ (для определенности скорость частиц $v = 1$); $\vec{\Omega}(\Omega_x, \Omega_y, \Omega_z)$ — единичный вектор направления полета частиц; $\Omega_x = \sqrt{1-\mu^2}\cos\varphi$ — проекция вектора $\vec{\Omega}$ на ось Ox; $\Omega_y = \sqrt{1-\mu^2}\sin\varphi$ — проекция $\vec{\Omega}$ на ось Oy; $\Omega_z = \mu$ — проекция $\vec{\Omega}$ на ось Oz (косинус угла между вектором $\vec{\Omega}$ и осью Oz); φ — угол между проекцией $\vec{\Omega}$ на плоскость Oxy и осью Ox; $n^{(0)} = \int\limits_{-1}^{1} d\mu \int\limits_{0}^{2\pi} N d\varphi$.

Уравнение (1) решается в области $d = \{(x, y, z) \in L,\ -1 \le \mu \le 1,\ 0 \le \varphi \le 2\pi\}$.

На внешней поверхности задаются граничные условия в виде потока частиц, входящих в тело при $(\vec{\Omega} \cdot \vec{n}) < 0$, где $\vec{n}$ — внешняя нормаль к поверхности, ограничивающей область L.

Далее рассматривается конечно-разностная аппроксимация уравнения (1) в случае, когда пространственная сетка в области L состоит из прямоугольных параллелепипедов. Значения параметра μ выбираются из интервала $(-1, 1)$, значения параметра φ — из интервала $(0, 2\pi)$.

Уравнение баланса в счетной ячейке в конечно-разностной форме получается с помощью интегро-интерполяционного метода:

$$\begin{aligned}\mathrm{div}_h\left(\vec{\Omega}N\right) \equiv \Omega_x S_{yz}(N_2 - N_1) + \\ + \Omega_y S_{xz}(N_4 - N_3) + \Omega_z S_{xy}(N_6 - N_5) + \\ + V\alpha N_0 = V\overline{F},\end{aligned} \qquad (2)$$

где $N_i\,(i = 1, 2, \ldots, 6)$ — средние значения искомой функции N на гранях ячейки; N_0 — значение функции N в центре ячейки; S_{yz}, S_{xz}, S_{xy} — площади граней ячейки; V — объем ячейки; $\overline{F} = \frac{1}{4\pi}\left(\beta\overline{n}^{(0)} + Q\right)$.

Скалярный поток $n^{(0)}$ в каждой счетной ячейке вычисляется следующим образом: $\overline{n}^{(0)} = \sum\limits_{w=1}^{n_w} N_0^w d\Omega_w$, $\sum\limits_{w=1}^{n_w} d\Omega_w = 4\pi$. Здесь w — номер направления полета частиц; n_w — число направлений полета частиц; $d\Omega_w$ — телесный угол.

Для замыкания системы сеточных уравнений по пространственным переменным используется DD-схема:

$$N_0 = \frac{N_1 + N_2}{2} = \frac{N_3 + N_4}{2} = \frac{N_5 + N_6}{2}. \qquad (3)$$

Система (2), (3) решается итерациями по источнику: $\mathrm{div}_h\left(\vec{\Omega}\overset{s+1}{N}\right) + V\alpha\overset{s+1}{N_0} = V\overline{\overset{s}{F}}$, где s — номер итерации.

Особенности используемых алгоритмов

Численное решение системы сеточных уравнений осуществляется с помощью алгоритма бегущего счета. Это алгоритм расчета ячеек пространственной сетки в определенной последовательности, которая зависит от освещенности граней ячеек. Алгоритм бегущего счета можно представить в виде ациклического орграфа $G = (V, E)$, в котором каждой вершине $v \in \{V\}$ соответствует решение уравнения баланса в ячейке фазового пространства, а каждой дуге $e \in \{E\}$ — поток частиц через грань смежных ячеек. Направление дуги определяется освещенностью грани, по которой в ячейке определяется свойство потока на грани — входящий он или выходящий. Полный телесный угол (4π) можно разделить на 8 угловых октантов по вариантам освещенности граней ячеек. Орграф алгоритма бегущего счета будет одинаковым для всех направлений полета частиц из одного октанта и разным для любых направлений из разных октантов.

На рис. 1 схематично представлен орграф бегущего счета в одном двумерном слое ячеек. Программная реализация алгоритма бегущего счета в циклах по сеточным направлениям (на

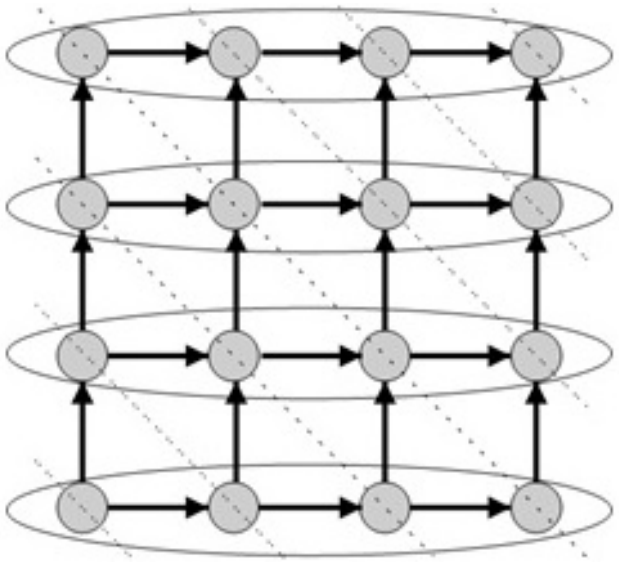

Рис. 1. Орграф $G = (V, E)$ алгоритма бегущего счета в двумерном слое

рис. 1 вершины, соответствующие одной строке расчетной сетки, выделены овалами) приводит к появлению зависимостей по данным, что не позволяет векторизовать такие циклы. С другой стороны, вершины графа можно объединить по признаку одинаковой максимальной длины пути от источника. На рис. 1 такие вершины выделены наклонными штриховыми линиями. Множество вершин с одинаковым расстоянием от источника образуют *гиперплоскость* [22], или *k-ломаную* [23]. Для вершин в одной гиперплоскости зависимостей по данным нет, что активно используется при MPI-распараллеливании алгоритма бегущего счета.

Первоначальный вариант распараллеливания тестовой программы ПАУК выполнен в модели распределенной памяти с помощью средств стандарта MPI. Алгоритм распараллеливания представляет собой модификацию KBA-алгоритма [2]. Это безытерационный алгоритм распараллеливания конвейерного типа, который основан на принципе геометрической декомпозиции. В случае регулярной пространственной декомпозиции алгоритм параллельного бегущего счета также имеет представление в виде орграфа (см. рис. 1). Независимость по данным у вершин в одной гиперплоскости приводит к свойству волнового параллелизма алгоритма. Зависимость по данным у вершин разных гиперплоскостей компенсируется за счет параллельных конвейеров.

Адаптация к ядру сопроцессора

Эффективная адаптация тестовой программы ПАУК к ядру сопроцессора Intel Xeon Phi поколения Knights Corner достигнута с помощью векторизации вычислений и программной предвыборки данных.

Векторизация вычислений. Дисциплина SIMD внедрена за счет согласованной модификации структуры данных и изменения последовательности вложенных циклов. Основным требованием векторизации цикла является отсутствие зависимостей по данным. Исходный вариант теста ПАУК такому требованию не удовлетворяет.

В работе рассмотрена векторизация двух циклов — по направлениям полета частиц в октанте и по элементам гиперплоскостей.

Векторизация по направлениям полета частиц. На ортогональной сетке бегущий счет для всех направлений в октанте n_w^{oct} представляется одним орграфом $G = (V, E)$. Это свойство позволяет сделать внутренним цикл по направлениям в октанте, тогда каждой вершине $v \in \{V\}$ соответствует решение уравнения баланса в n_w^{oct} ячейках фазового пространства. Единственной зависимостью по данным является операция редукции при вычислении интеграла по направлениям в центральной точке разностного шаблона: $\bar{n}^{(0)} = \sum_w N_0^w d\Omega_w$. Однако данная операция довольно эффективно векторизуется компилятором. Кроме того, часть данных, используемых при решении уравнения баланса, не зависит от направления, что уменьшает суммарную нагрузку на подсистему памяти.

В отличие от других схожих приложений, где используется векторизация по направлениям [13, 14, 24], в тесте ПАУК внутренний цикл организуется для порции из m направлений, а не для всех направлений в октанте. Размер порции определяется по формуле $m = \langle$*число VPU-модулей в ядре*$\rangle \times$ $\times \langle$*ширина векторного регистра в числах двойной точности*$\rangle$. Для обоих рассматриваемых вычислителей (Ivy Bridge и Xeon Phi) этот размер равен 8. Такой подход позволяет использовать множество направлений полета частиц в октанте не только для векторизации вычислений, но и для различных алгоритмов распараллеливания: OpenMP и MPI.

Модификация структуры данных требуется только для массивов с зависимостью от направлений. Первое измерение таких массивов определяется числом направлений в порции, что приводит к их форме $(m, \ldots, n_w^{oct}/m)$.

В аналогичных приложениях применяются различные подходы для векторизации цикла по направлениям: автовекторизация средствами компилятора [13], intrinsic-программирование [12], векторные типы данных библиотеки Eigen [24]. На тесте ПАУК были проведены сравнительные исследования производительности автовекторизации и intrinsic-программирования. Первоначальный, или *формальный*, вариант автовекторизации получен простым переносом цикла по m направлениям без каких-либо дополнительных модификаций в уравнении баланса. На Intel Xeon Phi коэффициент ускорения от векторизации составил

$Sp_{vec} = t(-no-vec)^*/t = 2{,}3$ при идеальном значении 8. Причины такого низкого результата заключаются в следующем:

– отсутствие информации у компилятора о выровненности данных и общем числе итераций векторизуемого цикла;
– вычислительная неоднородность итераций векторизуемого цикла.

При решении уравнения переноса используются немонотонные схемы второго порядка (например, DD-схема по пространству), которые могут приводить к появлению нефизичных отрицательных решений на разностном шаблоне (например, выходящие потоки на гранях ячейки). Для борьбы с этим явлением в тесте ПАУК применяется алгоритм балансного зануления [25], который, однако, приводит к вычислительной неоднородности при решении уравнения баланса.

Возникающая проблема решается следующим образом. Коэффициент балансного зануления k_b рассчитывается безусловно, но применяется ко всем неизвестным в ячейке по *маске*

$$N_i = kN_i,\ \forall\, i \in \{0, \{i^{\text{вых}}\}\}.$$

Здесь $i^{\text{вых}}$ — номера граней ячейки с выходящим потоком; $k = \begin{cases} k_b, & \text{если} \quad \min\{N_i^{\text{вых}}\} < 0; \\ 1, & \text{если} \quad \min\{N_i^{\text{вых}}\} \geq 0. \end{cases}$

После модификации алгоритма балансного зануления были реализованы еще два варианта векторизации цикла по восьми направлениям. *Предопределенный* вариант автовекторизации с точным указанием компилятору числа итераций в цикле ($m = 8$) и информации о выровненности данных обеспечил ускорение $Sp_{vec} = 6{,}1$. *Intrinsic*-вариант, реализованный с помощью intrinsic-функций, обеспечил значение $Sp_{vec} = 5{,}6$. Полученные результаты показывают, что программирование с использованием intrinsic-функций не всегда обеспечивает наилучшую производительность.

В табл. 1 приведены результаты ускорения выполнения процедуры бегущего счета в первом октанте для предопределенного варианта автовекторизации на CPU и сопроцессоре в последовательном режиме. Здесь Sp_{vec}^{ideal} — идеальный коэффициент ускорения от векторизации, определяемый шириной векторных регистров. В последнем столбце приведены ускорения от оптимизации кода (Sp_{opt}), т. е. отношение времени счета двух версий теста ПАУК: исходной и векторизованной.

Таблица 1

Коэффициенты ускорения счета для векторизации по восьми направлениям

Архитектура	Sp_{vec}^{ideal}	Sp_{vec}	Sp_{opt}
Ivy Bridge	4	2,4	4,9
Knights Corner	8	6,1	8,2

Более высокое значение $Sp_{opt}/Sp_{vec}^{ideal}$ на CPU, чем на сопроцессоре, вероятнее всего, объясняется улучшением параллелизма на уровне инструкций (ILP) за счет дополнительной возможности разворачивания (unrolling) векторизуемого цикла.

Векторизация по элементам гиперплоскостей. Переход от циклов по сеточным направлениям к циклам по гиперплоскостям и элементам в них не является алгоритмически сложным. Данный прием подробно описан в [22]. Для эффективной векторизации на Intel Xeon Phi сложность представляют плавающее число итераций векторизуемого цикла (от 1 до $\min(n_x, n_y)$, где n_x, n_y — соответственно число столбцов и строк) и формат представления данных, не обеспечивающий компактного размещения в памяти элементов одной гиперплоскости. Для решения перечисленных проблем в тестовой программе ПАУК были реализованы алгоритм выделения гиперплоскостей фиксированного размера и запись данных в специальном формате.

Идея заключается в переходе от представления по столбцам и строкам к представлению по столбцам и гиперплоскостям, т. е. от формы массивов (n_x, n_y, n_z) к форме (n_x, n_{hyp}, n_z), где $n_{hyp} = n_x + n_y - 1$. Для случая на рис. 1 алгоритм бегущего счета преобразуется к виду, представленному на рис. 2 слева. В новом орграфе первое сеточное направление содержит элементы одной гиперплоскости и зависимостей по данным не имеет. Множество вершин нового графа состоит из объединения двух множеств: счетного $\{V\}$ (на рис. 2 вершины серого цвета) и фиктивного $\{V_f\}$ (вершины белого цвета). Счетные вершины соответствуют ячейкам сетки в пространственной области L, фиктивные — нет. Решение уравнения баланса выполняется либо в вершинах обоих типов $\{V\} \cup \{V_f\}$, либо

* Здесь и далее запись $t(\ldots)$ означает время счета программы, скомпилированной с дополнительными ключами, указанными в скобках.

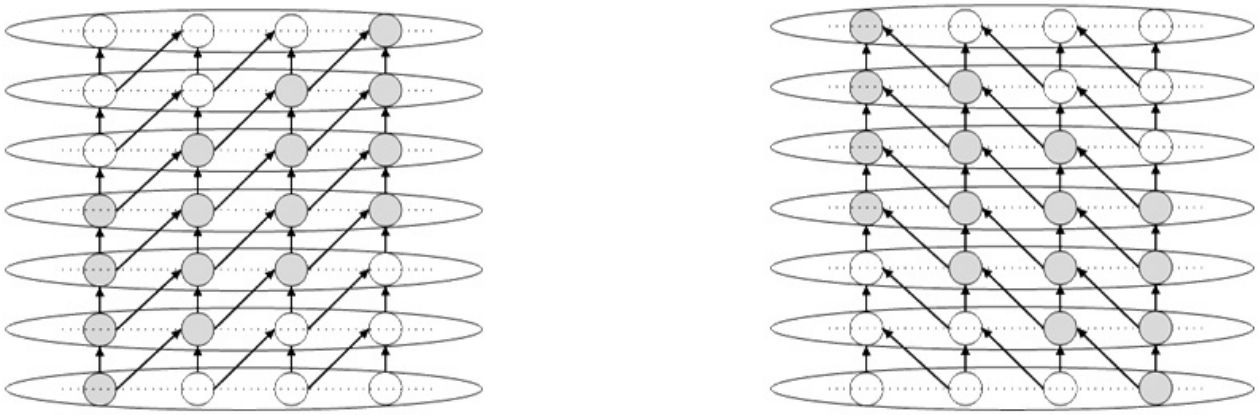

Рис. 2. Орграфы для двух вариантов бегущего счета на новых форматах данных

только в счетных $\{V\}$. В первом случае выходящий поток определяется с учетом типа вершины и по условию $N^{\text{вых}} = \begin{cases} N^{\text{вых}}, & v \in \{V\}; \\ N^{\text{вх}}, & v \in \{V_f\}, \end{cases}$ где $v \in \begin{cases} \{V\}, & \text{если} \quad i_y \in [1, n_y]; \\ \{V_f\}, & \text{если} \quad i_y \notin [1, n_y]. \end{cases}$ Взаимосвязь между индексами строк и гиперплоскостей для орграфа на рис. 2, слева определяется выражением $i_y = i_{hyp} - i_x + 1$; для орграфа на рис. 2, справа $i_y = i_{hyp} - n_x + i_x$, где i_x, i_y, i_{hyp} — индексы столбцов, строк и гиперплоскостей соответственно. Данный подход позволяет решить проблему плавающего числа итераций векторизуемого цикла по элементам гиперплоскости. При выполнении требования кратности 8 для числа столбцов сетки данный цикл эффективно векторизуется.

Требование компактного размещения данных для каждой гиперплоскости приводит к необходимости дублирования всех сеточных данных, которые не зависят от направлений. Для двумерного слоя ячеек возможны четыре варианта алгоритма бегущего счета и четыре орграфа. Для каждого из этих орграфов есть симметричная пара, которая получается сменой направления во всех дугах орграфа. При такой смене направления множества счетных вершин в гиперплоскостях не меняются, поэтому для каждой пары симметричных орграфов достаточно одного формата записи данных.

Новый формат предполагает переход от массива A с формой (n_x, n_y, n_z) к двум массивам A_1 и A_2 с формой (n_x, n_{hyp}, n_z) по следующему правилу:

$$A_1(i_x, i_{hyp}) = \begin{cases} A(i_x, i_{hyp} - i_x + 1), & v \in \{V\}; \\ A(1,1), & v \in \{V_f\}; \end{cases}$$

$$A_2(i_x, i_{hyp}) = \begin{cases} A(i_x, i_{hyp} - n_x + i_x), & v \in \{V\}; \\ A(1,1), & v \in \{V_f\}. \end{cases}$$

При выполнении требования кратности 8 для числа столбцов сетки новый формат данных обеспечивает выровненный доступ к памяти.

Очевидным недостатком предлагаемого подхода является увеличение размеров всех массивов, особенно массивов без зависимости от направлений полета частиц. Данный недостаток можно компенсировать различными вариантами усложнения формата по гиперплоскостям для минимизации $|V_f|$. Другим недостатком является увеличение суммарной вычислительной работы за счет решения уравнения баланса в фиктивных ячейках.

Главным достоинством векторизации по элементам гиперплоскостей является возможность полного сохранения ресурса направлений для различных алгоритмов распараллеливания на уровне кристалла, узла и множества узлов. Последнее может быть ценным для задач с небольшим числом направлений.

Результаты исследования ускорения вычислений от векторизации по элементам гиперплоскостей представлены в табл. 2. Здесь приведены среднеарифметические значения ускорений, полученные на сетках с разным числом столбцов (8, 16, 32, 64) при числе строк и слоев, равном соответственно 128 и 4. Для более точной оценки ускорения от оптимизации в таблице приводится метрика, учитывающая разное число счетных

Таблица 2

Значения коэффициентов ускорения счета для двух вариантов векторизации по элементам гиперплоскостей

Архитектура	$\{V\} \cup \{V_f\}$			$\{V\}$		
	Sp_{vec}	Sp_{opt}	Sp^*_{opt}	Sp_{vec}	Sp_{opt}	Sp^*_{opt}
Ivy Bridge	2,3	2,9	3,5	1,8	3,1	3,8
Knights Corner	5,6	5,0	6,1	5,3	5,2	6,4

вершин при счете по исходной и модифицированной версиям кода: $Sp^*_{opt} = \dfrac{t_{old}}{t_{new}} \dfrac{n_{hyp}}{n_y}$.

Из результатов табл. 1, 2 следует, что с учетом различного числа вершин в графах вариант векторизации по восьми направлениям производительнее варианта векторизации по элементам гиперплоскостей в 1,39 раза на Ivy Bridge и в 1,34 раза на Knights Corner.

В итоговый вариант тестовой программы ПАУК вошел алгоритм векторизации цикла по направлениям с предопределенным числом итераций ($m = 8$) и автовекторизацией средствами компилятора языка Fortran.

Предвыборка данных. Из-за определенных аппаратных ограничений на Intel Xeon Phi эффективную предвыборку данных в кэш-память первого и второго уровня можно реализовать только программным путем [5]. Компилятор достаточно эффективно справляется с задачей программной предвыборки за счет вставки специальных инструкций на этапе компиляции, особенно при обращении к данным с единичным смещением в памяти. В алгоритме бегущего счета такой шаблон доступа (с единичным смещением) всегда реализуется только для двух октантов из восьми. Для остальных шести октантов в бегущем счете иногда требуются смещения в $\sim n_x$ или $\sim n_x n_y$ ячеек. Частота таких неединичных смещений, или "прыжков" по памяти — $1/n_x$ и $1/(n_x n_y)$. Влияние этих прыжков на общую производительность приложения оценено по метрике $\max\limits_{i=\overline{1,8}}\{t_i\}/\min\limits_{i=\overline{1,8}}\{t_i\}$, где t_i — время бегущего счета по всем направлениям в i-м октанте. На сопроцессоре Intel Xeon Phi данная метрика имеет значение 1,47 для варианта векторизации по восьми направлениям.

Для решения указанной проблемы были реализованы два варианта предвыборки: неявный и явный. Неявная предвыборка основана на такой модификации формата представления данных, при которой для всех вариантов бегущего счета обеспечивается шаблон доступа к памяти с единичным смещением. Последовательность размещения данных полностью соответствует последовательности расчета циклов. Это потребовало четырехкратного увеличения всех массивов, инвариантных от направлений полета частиц. Явный вариант предвыборки основан на использовании единственной intrinsic-функции языка Fortran — `mm_prefetch`. В этом случае компилятору запрещается предвыборка данных, а все данные явно загружаются в L1/L2-кэши.

Из данных, приведенных в табл. 3, видно, что для обоих вариантов предвыборки время выполнения процедуры бегущего счета выравнялось для всех октантов, а коэффициент ускорения от программной предвыборки составил $\sim 1{,}38$.

В итоговый вариант тестовой программы ПАУК вошел неявный вариант предвыборки данных, так как он обеспечил на 7 % лучшую производительность по сравнению с явным вариантом.

Таблица 3

Результаты исследований производительности вариантов предвыборки данных в последовательном режиме на Knights Corner

Предвыборка	$\max\{t_i\}/\min\{t_i\}$	$t(-no-prefetch)/t$
Неявная	1,03	1,38
Явная	1,00	1,39

Адаптация к сопроцессору

Эффективная адаптация тестовой программы ПАУК к одному сопроцессору Intel Xeon Phi поколения Knights Corner достигнута за счет внедрения (MPI+X)-распараллеливания: по направлениям полета частиц и по геометрическим фрагментам (КВА).

Распараллеливание по направлениям. Алгоритм распараллеливания по направлениям полета частиц одного октанта реализован в модели общей памяти с помощью средств стандарта OpenMP. Это внутренний алгоритм распараллеливания относительно KBA. Параллельный бегущий счет организуется не для всех, а только для $8N_{OMP}$ направлений одного октанта, где N_{OMP} — число OpenMP-потоков в одном MPI-процессе. Расчет интеграла $n^{(0)}$ по всем направлениям осуществляется через локальный промежуточный массив, что позволяет избежать обмена данными между потоками непосредственно в ходе параллельного бегущего счета. MPI-обмены выполняются только одним OpenMP-потоком. Зернистость, или *гранулярность*, такая же, как у алгоритма распараллеливания по геометрическим фрагментам (см. далее).

Распараллеливание по геометрическим фрагментам. В исходном варианте тестовой программы ПАУК KBA-алгоритм распараллеливания реализован в модели распределенной памяти с помощью средств стандарта MPI. Исследования эффективности распараллеливания на одном сопроцессоре показали значительные накладные расходы на MPI-обмены. Поэтому в рамках теста ПАУК были выполнены еще две реализации KBA-алгоритма — в моделях общей памяти и разделенного глобального адресного пространства (PGAS). Все реализации выполнены при условии, что единица вычислительного ресурса рассчитывает один геометрический фрагмент.

KBA-алгоритм в модели общей памяти реализован с помощью стандарта OpenMP. Для передачи потоков между фрагментами используются глобальные массивы, размер которых зависит от числа дуг в орграфе параллельного бегущего счета. Для улучшения локальности данных используется принцип *"запись — в свою память, чтение — из чужой памяти"*. Для организации параллельного бегущего счета задействован механизм OpenMP-замков. Комбинация с OpenMP-распараллеливанием по направлениям выполнена по вложенной схеме: перед чтением данных выполняется синхронизация между головными OpenMP-потоками соседних фрагментов, затем синхронизация между всеми OpenMP-потоками для текущего фрагмента; после записи данных последовательность синхронизаций обратная.

KBA-алгоритм в модели разделенного глобального адресного пространства (PGAS) реализован с помощью MPI-3 SHM [26]. Коммуникатор, *окно*, *эпоха* создаются для каждой дуги орграфа параллельного бегущего счета в отдельности. Для каждой дуги орграфа создаются окно для передачи данных и окно для передачи управления. Проблема синхронизации данных, размещенных в общем окне, решена с помощью дополнительного атрибута `asynchronous` стандарта Fortran-2003. Этот атрибут использован при описании всех массивов-указателей, выделенных или назначенных с помощью средств MPI-3 SHM.

В табл. 4 приводятся результаты исследований эффективности распараллеливания для трех реализаций KBA-алгоритма на одном сопроцессоре Knights Corner. Режим запусков: двумерная декомпозиция, 30 геометрических фрагментов, 4 OpenMP-потока по направлениям, 120 параллельных процессов суммарно, по 2 процесса на одно ядро сопроцессора. Эффективность распараллеливания получена методом слабой масштабируемости, или увеличения задачи. В случае одного фрагмента задача посчитана с 4 OpenMP-потоками по направлениям и явной привязкой по 2 процесса на одно ядро.

Результаты табл. 4 демонстрируют значительные накладные расходы библиотеки MPI в случае двухточечных обменов внутри одного сопроцессора. Эффективность распараллелива-

Таблица 4

Эффективность распараллеливания (в %) в зависимости от гранулярности

Тип распараллеливания	Гранулярность			
	960	1 920	3 968	7 168
MPI+OpenMP	62	68	77	81
OpenMP+OpenMP	87	88	88	90
MPI-3 SHM+OpenMP	84	88	90	90

ния для реализаций в моделях общей памяти и PGAS имеет близкие значения и слабо зависит от объема вычислительной работы между обменами данных.

В итоговый вариант тестовой программы ПАУК вошла реализация KBA-алгоритма, выполненная в PGAS-модели. Реализация в модели общей памяти обладает существенным недостатком в части производительности MPI-обменов между различными устройствами/узлами, так как в стандарт MPI пока не входит функционал multi-EndPoints [27].

Адаптация к гетерогенному узлу

Для упрощения задачи отображения теста ПАУК на гетерогенную архитектуру был изменен исходный алгоритм MPI-распараллеливания. Пространственная 3D-декомпозиция заменена на 2D-декомпозицию (по столбцам и строкам), что упростило коммуникационный шаблон и снизило общую нагрузку на коммуникационную подсистему. Исключен дополнительный MPI-конвейер по слоям пространственной сетки, однако его работу можно сымитировать увеличением числа направлений. Режим старта MPI-конвейеров заменен с последовательного на одновременный, что повысило значение теоретической эффективности MPI-распараллеливания. Стратегия обработки множества одновременных конвейеров — самоупорядочение.

Для учета особенностей гетерогенной архитектуры в тест ПАУК внедрены возможность балансировки вычислительной нагрузки и процедура `mapping` отображения графа задачи на архитектуру с учетом некоторых особенностей последней.

Балансировка вычислительной нагрузки реализована статическим образом за счет задания разного числа строк в геометрических фрагментах на вычислителях двух типов.

Процедура `mapping` выполняется в два этапа.

На первом этапе множество всех MPI-процессов переупорядочивается по типу вычислителя: сначала следуют все MPI на процессорах, а затем — на сопроцессорах. Общее число геометрических фрагментов (MPI-процессов) в запуске определяется по формуле

$$P = P_x\Big(P_y^{CPU} + P_y^{MIC}\Big),$$

где P_x — число разбиений по столбцам; P_y^{CPU}, P_y^{MIC} — число разбиений по строкам соответственно на процессорах и сопроцессорах.

На втором этапе множество всех MPI-процессов переупорядочивается по принадлежности устройств одному гетерогенному узлу, чтобы минимизировать число внешних дуг в орграфе параллельного бегущего счета. Внешней дуге в данном случае соответствует передача данных, выполняемая через коммуникационную подсистему гетерогенной суперЭВМ. Общее число дуг в орграфе бегущего счета рассчитывается по формуле

$$|E| = P_x(P_y - 1) + P_y(P_x - 1),$$

а число внешних дуг для симметричного режима — по формуле

$$|E_{ext}| = P_x + \sum_{dev}\left[P_x\left(\frac{P_y^{dev}}{m_y^{dev}} - 1\right) + P_y^{dev}\left(\frac{P_x}{m_x} - 1\right)\right].$$

Здесь m_x — новый параметр теста, позволяющий определить правило локального переупорядочения; $m_y^{dev} = MPI_{node}^{dev}/m_x$, где MPI_{node}^{dev} — число MPI-процессов одного типа на одном узле гетерогенной суперЭВМ; индекс dev обозначает тип вычислителя и принимает значение CPU или MIC.

Минимизация числа внешних дуг в орграфе позволяет снизить нагрузку на коммуникационную подсистему за счет выполнения большей части обменов через эффективный механизм MPI-3 SHM, а не через коммуникационное оборудование.

Результаты исследований производительности

При тестировании использовались следующие параметры дискретизации задачи: на один вычислительный узел — 300 столбцов, 480 строк, 4 слоя пространственной сетки; на четыре узла — 600 столбцов, 960 строк, 4 слоя. Число направлений в задаче $n_w = 18\,432$ позволяет имитировать MPI-конвейеры по другим переменным фазового пространства.

Процедура тестирования заключалась в запуске задачи на одинаковом числе вычислительных узлов в трех режимах: на процессорах (CPU-only), на сопроцессорах (native) и в симметричном режиме (symmetric).

Производительность на процессорах. Основной метрикой в режиме счета на процессорах является эффективность распараллеливания $E_n = t_1/t_n \cdot 100\,\%$. В расчетах в данном режиме использовалось только MPI-распараллеливание. Для варианта КВА-алгоритма, реализованного в новой версии программы ПАУК, значение теоретической эффективности распараллеливания (для идеальной суперЭВМ без шумов и с мгновенными обменами) оценивается по формуле

$$E_{theor} = \frac{n_w/\left(8N_{OMP}\right)\cdot 100\,\%}{n_w/\left(8N_{OMP}\right) + (P_x - 1) + (P_y - 1)}.$$

Значение E_{theor} выступает в качестве критерия качества. В табл. 5 приведены значения двух эффективностей распараллеливания в запусках на одном и четырех вычислительных узлах.

По метрике E_n получены высокие показатели на одном узле — 97 %, что объясняется эффективностью обменов через MPI-3 SHM. Некоторое снижение эффективности на четырех узлах обусловлено влиянием коммуникационного оборудования, так как значение E_{theor} слабо изменилось при переходе от одного к четырем узлам.

Таблица 5

Эффективность распараллеливания на процессорах

Число узлов	E_{theor}, %	E_n, %
1	99,7	97,2
4	99,3	91,3

Производительность на сопроцессорах. Основной метрикой в режиме счета на сопроцессорах является ускорение счета: $Sp_{native} = t_{CPU}/t_{MIC}$. Критерий качества рассчитывается по пиковой производительности: $Sp^{ideal}_{native} = R^{MIC}_{peak}/R^{CPU}_{peak} = 5{,}4$.

В табл. 6 представлены результаты фактически достигнутых ускорений на одном и четырех вычислительных узлах при варьировании как числа MPI-процессов на каждом сопроцессоре (M_{MPI}), так и числа OpenMP-потоков на одном ядре (M_{OMP}). Число потоков в MPI-процессе (N_{OMP}) выбиралось так, чтобы обеспечить нагрузку в 2, 3 и 4 процесса на одно ядро сопроцессора.

Наилучшее ускорение достигает всего 3 раз при идеальном значении в 5,4 раза. Причина такого отставания заключается в FMA (Fused Multiply-Add) инструкциях, относительно которых рассчитан пик сопроцессоров. Исследуемый CPU Intel Ivy Bridge такие инструкции выполнять не может. FMA-инструкции оказывают достаточно небольшое влияние на производительность теста ПАУК: $Sp_{fma} = t(-no - fma)/t = 1{,}2$, что позволяет скорректировать критерий качества: $\overline{Sp}^{ideal}_{native} = 0{,}5 Sp_{fma} Sp^{ideal}_{native} = 3{,}2$. По новому критерию результаты табл. 6 позволяют утверждать об эффективной адаптации теста к сопроцессорам.

Таблица 6

Коэффициент ускорения счета на сопроцессорах

Число узлов	M_{OMP}	M_{MPI}		
		20	30	60
1	2	2,7	2,7	2,6
	3	3,0	2,9	3,0
	4	3,0	3,0	2,9
4	2	2,6	2,6	2,4
	3	2,8	2,8	2,6
	4	2,9	2,8	2,6

Производительность в симметричном режиме. Основной метрикой в симметричном режиме является ускорение счета $Sp_{sym} = t_{CPU}/t_{CPU+MIC}$. Критерий качества $Sp^{ideal}_{sym} = B + 1$, где $B = \dfrac{P^{MIC}_y n^{MIC}_y}{P^{CPU}_y n^{CPU}_y}$ — коэффициент статической балансировки вычислительной нагрузки. Здесь n^{CPU}_y, n^{MIC}_y — число строк в одном геометрическом фрагменте на процессорах и сопроцессорах соответственно. Коэффициент балансировки определен по лучшему результату ускорения в режиме на сопроцессорах $Sp_{native} \equiv B = 3$, что означает $Sp^{ideal}_{sym} = 4$.

В табл. 7 представлены результаты фактически достигнутых ускорений на одном и четырех вычислительных узлах при варьировании как числа MPI-процессов на каждом сопроцессоре (M_{MPI}), так и числа OpenMP-потоков на одном ядре (M_{OMP}). Число потоков в MPI-процессе (N_{OMP}) выбиралось так, чтобы обеспечить нагрузку в 2, 3 и 4 процесса на одно ядро сопроцессора. На процессорах $N_{OMP} = 1$.

Результаты на одном узле близки к идеальному значению, на четырех узлах наблюдается некоторое уменьшение коэффициента уско-

Таблица 7

Коэффициент ускорения счета в симметричном режиме

Число узлов	M_{OMP}	M_{MPI}		
		20	30	60
1	2	3,6	3,5	3,3
	3	3,8	3,8	3,8
	4	3,9	3,7	3,8
4	2	3,1	3,0	3,0
	3	3,6	3,2	3,5
	4	3,6	3,1	3,5

рения. Данный эффект объясняется снижением теоретической эффективности распараллеливания в расчетах с использованием сопроцессоров. Число счетных тактов в MPI-конвейерах обратно пропорционально максимальному по устройствам числу OpenMP-потоков, а в режиме CPU-only $N_{OMP} = 1$.

Заключение

Тестовая программа ПАУК, изначально разработанная для тестирования производительности однородных суперЭВМ, за счет своей компактности является удобным полигоном для апробации различных алгоритмов и технологий параллельного программирования в интересах освоения гетерогенных архитектур.

На примере адаптации теста ПАУК к сопроцессорам Intel Xeon Phi поколения Knights Corner в работе представлены сравнения производительностей:

- двух алгоритмов векторизации циклов (по направлениям и элементам гиперплоскости);
- двух способов векторизации циклов (автовекторизация и intrinsic-программирование);
- двух способов предвыборки данных (возможности компилятора и intrinsic-функция);
- трех реализаций KBA-алгоритма на одном сопроцессоре в различных моделях памяти (общей, распределенной, PGAS).

Исходная версия программы ПАУК с алгоритмом только MPI-распараллеливания и невекторизованным счетным ядром при адаптации к архитектуре Intel MIC была трансформирована в итоговый вариант с алгоритмом (MPI + MPI-3 SHM + OpenMP)-распараллеливания. Векторизация выполнена для цикла по восьми направлениям одного октанта за счет автовекторизации. Эффективность предвыборки данных средствами компилятора достигнута с помощью специального формата данных, обеспечившего шаблон доступа к памяти с единичным смещением для всех вариантов алгоритма бегущего счета.

Результаты тестирования на процессорах, сопроцессорах и в симметричном режиме продемонстрировали эффективность адаптации программы ПАУК к гетерогенной архитектуре с Intel Xeon Phi.

Список литературы

1. *Бочков А. И., Нуждин А. А.* Параллельный алгоритм решения трехмерного кинетического уравнения переноса. Программа ПАУК для тестирования многопроцессорных вычислительных систем // Параллельные вычислительные технологии (ПАВТ'2008): Тр. межд. науч. конф. (С.-Пб., 28 января — 1 февраля 2008 г.) Челябинск: Изд-во ЮУрГУ, 2008.
   Bochkov A. I., Nuzhdin A. A. Parallelny algoritm resheniya trekhmernogo kineticheskogo uravneniya perenosa. Programma PAUK dlya testirovaniya mnogoprotsessornykh vychislitelnykh sistem // Parallelnye vychislitelnye tekhnologii (PAVT'2008): Tr. mezhd. nauch. konf. (S.-Pb., 28 yanvarya — 1 fevralya 2008). Chelyabinsk: Izd-vo YuUrGU, 2008.
2. *Koch K. R., Baker R. S., Alcouffe R. E.* Solution of the first-order form of the 3-D discrete ordinates equation on a massively parallel processor // Trans. of the Amer. Nuc. Soc. 1992. Vol. 65. P. 198.
3. *Алексеев А. В., Беляев С. П., Бочков А. И., Быков А. Н., Ветчинников М. В., Залялов А. Н., Нуждин А. А., Огнев С. П., Самсонова Н. С., Сапронов И. С., Чистякова И. Н., Шемякина Т. В., Шагалиев Р. М., Янилкин Ю. В.* Методические прикладные тесты РФЯЦ-ВНИИЭФ для численного исследования параметров высокопроизводительных вычислительных систем // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2020. Вып. 2. С. 86—100.
   Alekseev A. V., Belyaev S. P., Bochkov A. I.,

Bykov A. N., Vetchinnikov M. V., Zalyalov A. N., Nuzhdin A. A., Ognev S. P., Samsonova N. S., Sapronov I. S., Chistyakova I. N., Shemyakina T. V., Shagaliev R. M., Yanilkin Yu. V. Metodicheskie prikladnye testy RFYaTs-VNIIEF dlya chislennogo issledovaniya parametrov vysokoproizvoditelnykh vychislitelnykh sistem // Voprosy atomnoy nauki i tekhniki. Ser. Matematicheskoe modelirovanie fizicheskikh protsessov. 2020. Vyp. 2. S. 86—100.

4. *Алексеев А. В., Беляков И. М., Бочков А. И., Евдокимов В. В., Ириничев Е. А., Морозов В. Ю., Москвин А. Н., Нуждин А. А., Пепеляев М. П., Резчиков В. Ю., Сучкова В. В., Шагалиев Р. М., Шарифуллин Э. Ш., Шемякина Т. В., Шумилин В. А.* Методика САТУРН-2005. Математические модели, алгоритмы и программы решения многомерных задач переноса частиц и энергии // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2013. Вып. 4. С. 17—30.
Alekseev A. V., Belyakov I. M., Bochkov A. I., Evdokimov V. V., Irinichev E. A., Morozov V. Yu., Moskvin A. N., Nuzhdin A. A., Pepelyaev M. P., Rezchikov V. Yu., Suchkova V. V., Shagaliev R. M., Sharifullin E. Sh., Shemyakina T. V., Shumilin V. A. Metodika SATURN-2005. Matematicheskie modeli, algoritmy i programmy resheniya mnogomernykh zadach perenosa chastits i energii // Tam zhe. 2013. Vyp. 4. S. 3—16.

5. *Jeffers J., Reinders J.* Intel Xeon Phi Coprocessor High Performance Programming. USA: Morgan Kaufmann, 2013.

6. Информация о вычислительном кластере ООО "Центр компетенций и обучения". https://compcenter.org/predostavlenie-vychislitelnyh-resur.
Informatsiya o vychislitelnom klastere OOO "Tsentr kompetentsiy i obucheniya". https://compcenter.org/predostavlenie-vychislitelnyh-resur.

7. *Reinders J., Jeffers J.* High-Performance Parallelism Pearls. Volume 1 and Volume 2: Multicore and Many-core Programming Approaches. USA: Morgan Kaufmann, 2015.

8. *Hoisie A., Johnson G., Kerbyson D. J., Lang M., Pakin S.* A Performance comparison through benchmarking and modeling of three leading supercomputers: Blue Gene/L, Red Storm, and Purple // SC'06: Proc. 2006 ACM/IEEE Conf. on Supercomputing. Tampa, FL, 2006. P. 3-3.
DOI: 10.1109/SC.2006.4.

9. *Coarfa C., Dotsenko Y., Mellor-Crummey J.* Experiences with Sweep3D implementations in co-array Fortran // J. Supercomp. 2006. Vol. 36 (2). P. 101—121.
DOI: 10.1007/s11227-006-7952-7.

10. *Petrini F., Fossum G., Fernandez J., Varbanescu A. L., Kistler M., Perrone M.* Multicore surprises: Lessons learned from optimizing Sweep3D on the cell broadband engine // 2007 IEEE Int. Parallel and Distributed Processing Symposium. Rome, Italy. March 26—30, 2007. P. 1—10.
DOI: 10.1109/IPDPS.2007.370252.

11. *Gong C., Liu J., Gong Z., Qin J., Xie J.* Optimizing Sweep3D for Graphic Processor Unit. Algorithms and Architectures for Parallel Processing. ICA3PP 2010. Lecture Notes in Computer Science / Ed. by C. H. Hsu, L. T. Yang, J. H. Park, S. S. Yeo. Vol. 6081. Berlin: Heidelberg Springer, 2010.

12. *Wang Q., Xing Z., Liu J., Qiang X., Gong C., Jiang J.* Parallel 3D deterministic particle transport on Intel MIC architecture // 2014 Int. Conf. on High Performance Computing & Simulation (HPCS). Bologna, 2014. P. 186—192. DOI: 10.1109/HPCSim.2014.6903685.

13. Информация о SNAP. https://github.com/lanl/SNAP.
Informatsiya o SNAP. https://github.com/lanl/SNAP.

14. *Baker, R.* An SN Algorithm for modern architectures // Nucl. Sci. Eng. 2017. Vol. 185, No 1. P. 107—116.
DOI: 10.13182/NSE15-124.

15. *Rajan M., Doerfler D., Hammond S.* Trinity Benchmarks on Intel Xeon Phi (Knights Corner). SAND2015-0454. Report. Sandia National Laboratories, 2015.

16. *Amer A., Iwasaki S., Raffenetti K., Shiryaev M., Si M., Taura K., Thapaliya S., Balaji P., Archer C., Blocksome M., Cao C., Chuvelev M., Fujita H., Garzaran M., Guo Y., Hammond J.* Software combining to mitigate multithreaded MPI contention // ICS'19:

Proc. ACM Int. Conf. Supercomputing. NY, USA: Association for Computing Machinery, 2019. P. 367—379.
DOI: 10.1145/3330345.3330378.

17. *Deakin T., McIntosh-Smith S., Gaudin W.* Many-core acceleration of a discrete ordinates transport mini-app at extreme scale // High Performance Computing: Proc. 31^{st} Int. Conf. Frankfurt, Germany: Springer International Publishing, Cham, 2016. P. 429—448.
18. *Kunen A. J., Bailey T. S., Brown P. N.* Kripke — A massively parallel transport mini-app // ANS MC2015 — Joint Int. Conf. on Mathematics and Computation, Supercomputing in Nuclear Applications and the Monte Carlo Method. Nashville, Tennessee. April 19—23, 2015.
19. *Messer B., D'Azevedo E., Hill J., Joubert W., Berrill M., Zimmer C.* Miniapps derived from production hpc applications using multiple programming models // Int. J. High Performance Comp. Appl. 2018. Vol. 32, Issue 4. P. 582—593.
DOI:10.1177/1094342016668241.
20. *Searles R, Chandrasekaran S., Joubert W., Hernandez O.* MPI + OpenACC: accelerating radiation transport mini-application, minisweep, on heterogeneous systems // Comput. Phys. Commun. 2019. Vol. 236. P. 176—187.
21. *Evans T. M., Joubert W., Hamilton S. P., Johnson S. R., Turner J. A., Davidson G. G., Pandya T. M.* Three-dimensional discrete ordinates reactor assembly calculations on GPUs // ANS MC2015 — Joint Int. Conf. on Mathematics and Computation, Supercomputing in Nuclear Applications and the Monte Carlo Method. Nashville, Tennessee. April 19—23, 2015.
22. *Lamport L.* The parallel execution of DO loops // Communications of the ACM. 1974. Vol. 17, No 2. P. 83—93.
23. *Трощиёв В. Е.* О классах сеток, допускающих консервативные аппроксимации двумерного оператора переноса треугольным разностным оператором // Журнал вычисл. мат. и мат. физ. 1976. Т. 16, № 3. С. 793—797.
Troshchiyev V. E. O klassakh setok, dopuskayushchikh konservativnye approksimatsii dvumernogo operatora perenosa treugolnym raznostnym operatorom // Zhurnal vychisl. mat. i mat. fiz. 1976. T. 16, № 3. S. 793—797.
24. *Moustafa S., Fevotte F., Faverge M., Plagne L., Ramet P.* Efficient parallel solution of the 3D stationary Boltzmann transport equation for diffusive problems // J. Comp. Phys. 2019. Vol. 388. P. 335—349.
25. *Елесин В. А., Трощиёв В. Е., Федянин В. И., Юдинцев В. Ф.* Численная методика и организация программы для решения многогруппового нестационарного кинетического уравнения // Комплексы программ математической физики. Новосибирск: ВЦ СО АН СССР, 1972. С. 18—23.
Elesin V. A. , Troshchiyev V. E., Fedyanin V. I., Yudintsev V. F. Chislennaya metodika i organizatsiya programmy dlya resheniya mnogogruppovogo nestatsionarnogo kineticheskogo uravneniya // Kompleksy programm matematicheskoy fiziki. Novosibirsk: VTs SO AN SSSR, 1972. S. 18—23.
26. *Hoefler T., Dinan J., Buntinas D., Balaji P., Barrett B., Brightwell R., Gropp W., Kale V., Thakur R.* MPI + MPI: A new hybrid approach to parallel programming with MPI plus shared memory // Computing. 2013. Vol. 95 (12). P. 1121—1136.
27. *Dinan J., Balaji P., Goodell D., Miller D., Snir M., Thakur R.* Enabling MPI Interoperability Through Flexible Communication Endpoints. EuroMPI, 2013.

Статья поступила в редакцию 10.06.20.

TEST PROGRAM "PAUK" AS A TESTING RANGE FOR PARALLEL PROGRAMMING ALGORITHMS AND TECHNIQUES / A. A. Nuzhdin (FSUE "RFNC-VNIIEF", Sarov, Nizhniy Novgorod region).

PAUK is a test program that numerically solves the three-dimensional one-group neutron transport equation on orthogonal spatial grids by the difference Sn-method. The paper presents the results of its adaptation for a heterogeneous computing system with Intel's Knights Corner Xeon Phi coprocessors. The adaptation included testing of various parallel programming algorithms and techniques, namely loop vectorization in directions and hyperplane elements, automatic vectorization and intrinsic programming, explicit and implicit data prefetching, implementation of the KBA algorithm in three memory models (shared, distributed, PGAS). Adaptation efficiency was verified by program performance studies in various execution modes at the heterogeneous computing system: CPU-only, native and symmetric.

Keywords: S_n-method, sweep algorithm, KBA algorithm, Intel Xeon Phi, vectorization, data prefetching, MPI-3 SHM.