Федеральное государственное бюджетное учреждение

Национальный исследовательский центр «Курчатовский институт»

Федеральное государственное учреждение «Федеральный научный центр
Научно-исследовательский институт системных исследований
Российской академии наук»
(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 14 № 2

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2024

Тематика номера:
Математическое моделирование и визуализация систем виртуального окружения, моделирование многопроцессроных систем, информационные и компьютерные технологии, информационные технологии в учебной информатике, медицинская информатика

Журнал публикует оригинальные статьи по следующим областям исследований: математика, математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника, математические исследования и вопросы численного анализа, история науки и техники.

The topic of the issue:

Mathematical modeling and visualization of virtual environment systems, modeling of multiprocessor systems, information and computer technology, information technology in educational informatics, medical informatics

The Journal publishes novel articles on the following research arias: mathematics, mathematical and computer modeling, image processing, visualization, system analysis, signal processing, information security, information technologies, high-performance computing, optical-neural technologies, micro- and nanoelectronics, mathematical researches and problems of numerical analysis, history of science and of technique.

Заведующий редакцией: В.Е.Текунов

Издатель: ФГУ ФНЦ НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

CONTENT

Моделирование движения автомата перекоса виртуальных моделей марсианских летательных аппаратов вертолетного типа

Е. В. Страшнов[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, strashnov_evg@mail.ru

Аннотация. В работе рассматривается задача моделирования движения автомата перекоса несущих винтов летательных аппаратов вертолетного типа в системах виртуального окружения. Для ее решения предлагается подход, в котором вычисление координат составных частей механизма осуществляется без учета их динамики. При реализации такого подхода был задействован метод Ньютона-Рафсона для решения систем нелинейных уравнений. Апробация предлагаемых в статье методов и подходов была проведена в разработанном комплексе виртуального окружения на примере моделирования движения виртуальной модели марсианского вертолета соосной схемы. Результаты апробации показали адекватность и эффективность предложенных в статье решений и их применимость для систем виртуального окружения.

Ключевые слова: винтокрылый летательный аппарат, несущий винт, автомат перекоса, шаг винта, метод Ньютона-Рафсона, системы виртуального окружения

1. Введение

Полет винтокрылых летательных аппаратов осуществляется за счет подъемной силы, создаваемой одним или несколькими несущими винтами. При этом взлет, посадка, горизонтальный полет и маневрирование вертолетов обеспечивается за счет автомата перекоса несущего винта. Автомат перекоса представляет собой механизм с приводом одного или нескольких двигателей, который изменяет угол установки лопасти винта (шаг винта) в зависимости от того, где лопасть оказывается в определенный момент времени при вращении винта как единого целого.

В данной статье рассматриваются марсианские летательные аппараты вертолетного типа. Одна из задач для таких аппаратов состоит в обучении операторов навыкам дистанционного управления ими с помощью специальных пультов. Однако тренировка с использованием реальных летательных аппаратов может привести к их поломке. Альтернативный подход заключается в том, чтобы проводить обучение на виртуальных моделях летательных аппаратов в виртуальной среде. Качество такого обучения непосредственно зависит от точности и адекватности математических моделей, применяемых для реализации движения виртуальных летательных аппаратов. Поэтому разработка методов и подходов моделирования движения летательных аппаратов вертолетного типа в системах виртуального окружения является важной и актуальной задачей.

Существующие методы и подходы моделирования движения летательных аппаратов вертолетного типа [1] – [5] основаны на законах аэродинамики. Математическая модель динамики вертолета зависит от его конструкции и схемы расположения винтов. Например, динамика вертолета соосной схемы описывается дифференциальными и нелинейными уравнениями относительно его координат и индуцированных скоростей воздушного потока верхнего и нижнего винта [6]. В свою очередь, механизм автомата перекоса вертолета представляет собой систему шарнирно связанных тел, содержащую замкнутые кинематические цепи [7]. Один из подходов для моделирования динамики таких систем основан на применении метода множителей Лагранжа [8], [9] с обеспечением голономных связей, накладываемых на координаты тел. В этом методе задача сводится к вычислению неизвестных величин множителей Лагранжа (сил и моментов) на каждом шаге моделирования путем решения громоздкой системы линейных уравнений. Для ее решения, как правило, используются численные итерационные методы. Проблема такого подхода заключается в том, что моделирование динамики объектов в системах виртуального окружения необходимо осуществлять в масштабе реального времени. Поэтому в этих методах число итераций меньше, чем необходимо, и это приводит к тому, что в процессе моделирования из-за высокой скорости вращения винта голономные связи механизма автомата перекоса будут нарушаться.

В данной работе предлагается подход, основанный на кинематическом способе расчета движения механизма автомата перекоса вертолета. В предлагаемом подходе для вычисления коор-

динат звеньев механизма был задействован метод Ньютона-Рафсона решения систем нелинейных уравнений. Обоснование правомерности такого подхода базируется на том факте, что автомат перекоса обеспечивает только установку общего и циклического шага винта. При этом само движение автомата перекоса не оказывает существенного влияния на динамику основной части вертолета и его аэродинамические характеристики. Апробация предлагаемых в статье решений была проведена в программном комплексе виртуального окружения VirSim [10] на примере моделирования движения марсианского вертолета с соосным расположением винтов, которая показала их адекватность и эффективность.

2. Автомат перекоса

Автомат перекоса представляет собой механизм для управления углами установки лопастей винта. При его одновременном изменении для всех лопастей винта (общий шаг винта) изменяется подъемная сила, что обеспечивает управление вертикальным движением вертолета. В свою очередь горизонтальное движение вертолета достигается за счет разных углов установки лопастей винта (циклический шаг винта) при их круговом движении.

Рис. 1. Виртуальная модель автомата перекоса

На рис. 1 показана виртуальная модель автомата перекоса несущего винта двухлопастного вертолета. Для данной модели механизм управляется с помощью трех электроприводов *1*, установленных на корпусе летательного аппарата. Эти приводы воздействуют посредством тяг *5* на внешнюю тарелку *2*, изменяя ее положение и углы наклона по каналам крена и тангажа. Она не вращается и связана с внутренней вращающейся тарелкой *3* посредством шаровой опоры *7* (качающегося подшипника). Тяги *6* соединяют внутреннюю тарелку с лопастями винта *4*, изменяя их угол установки. В установившемся состоянии тарелки параллельны плоскости вращения лопастей винта. Рассматриваемый механизм автомата перекоса устроен таким образом, что общий шаг винта регулируется смещением тарелок вдоль его оси вращения, а циклический шаг винта обеспечивается за счет наклона тарелок.

3. Моделирование движения автомата перекоса

Предлагаемое решение для моделирования движения механизма автомата перекоса заключается в следующем. Сначала под действием электроприводов вычисляются углы поворотов тяг управления креном и тангажом, которые соединяют корпус летательного аппарата с внешней тарелкой автомата перекоса. Данная задача сводится к решению нелинейных уравнений относительно расстояний между точками крепления звеньев тяг с внешней тарелкой. Затем вычисляются новые координаты (положение, углы крена и тангажа) тарелок. На последнем этапе определяются новые углы поворотов тяг, идущие к лопастям несущего винта. Для этого для каждой лопасти решается система из двух нелинейных уравнений относительно координат точек крепления тяг с внутренней тарелкой автомата перекоса. На выходе получаем новые углы (общий и циклический) установки лопастей винта. Далее рассмотрим предлагаемое решение более подробно.

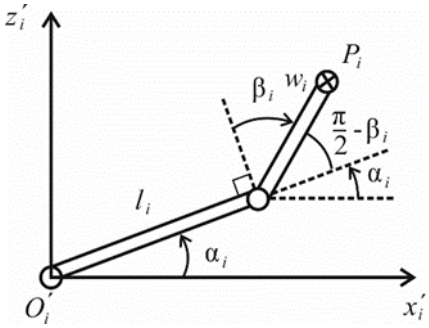

Рис. 2. Тяга управления креном и тангажом

3.1. Вычисление координат внешней тарелки автомата перекоса

Рассмотрим тягу управления креном и тангажом внешней тарелки автомата перекоса (см. рис. 2). Она представляет собой двухзвенный механизм, движение которого описывается с помощью двух углов α_i и β_i, $i = 1,2,3$. Поворот первого звена механизма на угол α_i осуществляется посредством электропривода, в то время как угол поворота β_i второго звена является неизвестным. Координаты точек P_i крепления тяг с внешней тарелкой автомата перекоса в двумерной системе координат $O_i'x_i'z_i'$ выражаются через эти углы следующим образом

$$P_{xi}' = l_i \cos\alpha_i + w_i \cos\left(\alpha_i - \beta_i + \pi/2\right) = \\ = l_i \cos\alpha_i + w_i \sin\left(\beta_i - \alpha_i\right);$$

$$P'_{zi} = l_i \sin\alpha_i + w_i \sin\left(\alpha_i - \beta_i + \pi/2\right) =$$
$$= l_i \sin\alpha_i + w_i \cos\left(\alpha_i - \beta_i\right),$$

где l_i и w_i – длины звеньев тяг.

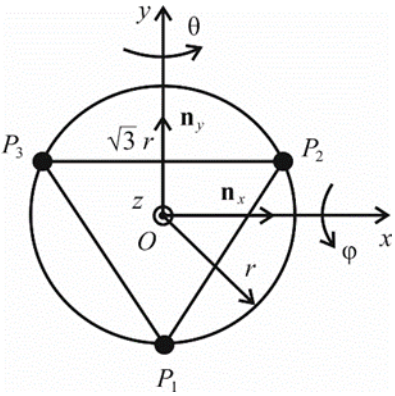

Рис. 3. Внешняя тарелка автомата перекоса

Неизвестные углы β_i определяются из условий, что точки крепления после поворота всех тяг остаются на внешней тарелке. Пусть O_1XYZ и $Oxyz$ системы координат внешней тарелки в начальном и текущем состоянии. Тогда необходимо преобразовать полученные координаты точек P_i к системе координат O_1XYZ и обеспечить, чтобы расстояния между точками оставались неизменными (см. рис. 3). Это приводит к системе нелинейных уравнений следующего вида

$$\begin{cases} f_1(\beta_1,\beta_2) = \left\|\overrightarrow{P_1P_2}\right\|^2 - 3r^2 = 0; \\ f_2(\beta_2,\beta_3) = \left\|\overrightarrow{P_2P_3}\right\|^2 - 3r^2 = 0; \\ f_3(\beta_1,\beta_3) = \left\|\overrightarrow{P_1P_3}\right\|^2 - 3r^2 = 0, \end{cases} \qquad (1)$$

где r – радиус внешней тарелки.

Для решения этой системы воспользуемся численным методом Ньютона-Рафсона. Этот метод является итерационным и на каждом шаге итерации задача (1) сводится к решению системы линейных уравнений, которая в векторной форме примет вид

$$\mathbf{J}\Delta\boldsymbol{\beta}^{(k)} = -\mathbf{f}(\boldsymbol{\beta}^{(k)}),\ \Delta\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k+1)} - \boldsymbol{\beta}^{(k)}, \qquad (2)$$

где k – номер итерации, $\boldsymbol{\beta}^{(k)} = \left(\beta_1^{(k)}, \beta_2^{(k)}, \beta_3^{(k)}\right)^{\mathrm{T}}$, $\mathbf{f} = \left(f_1, f_2, f_3\right)^{\mathrm{T}}$, $\mathbf{J}$ – матрица Якоби.

Итерации с решением системы (2) продолжаются до тех пор, пока не будет выполнено условие $\left\|\Delta\boldsymbol{\beta}^{(k)}\right\| \le \varepsilon$, где ε – заданное малое число.

Далее задача сводится к тому, чтобы определить положение и углы наклона внешней тарелки. Положение внешней тарелки вычисляется как среднее между координатами точек P_i:

$$z = O_z = (P_{1z} + P_{2z} + P_{3z})/3.$$

Ориентация внешней тарелки задается с помощью двух последовательных поворотов на углы φ и θ вокруг осей x и y. В этом случае матрица перехода $\mathbf{R}$ из системы координат $Oxyz$ в систему координат O_1XYZ примет вид

$$\mathbf{R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} =$$
$$= \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ \sin\theta\sin\varphi & \cos\varphi & -\cos\theta\sin\varphi \\ -\sin\theta\cos\varphi & \sin\varphi & \cos\theta\cos\varphi \end{pmatrix}.$$

Столбцы матрицы $\mathbf{R}$ задают орты осей системы координат $Oxyz$ в системе координат O_1XYZ. Согласно рис. 3, орты вычисляются как

$$\mathbf{n}_x = \frac{\overrightarrow{P_3P_2}}{\left\|\overrightarrow{P_3P_2}\right\|},\ \mathbf{n}_y = \frac{\overrightarrow{P_1O}}{\left\|\overrightarrow{P_1O}\right\|},\ \mathbf{n}_z = \mathbf{n}_x \times \mathbf{n}_y.$$

Тогда искомые углы выражаются через полученные орты следующим образом

$$\varphi = \mathrm{atan2}(\mathbf{n}_{y,z}, \mathbf{n}_{y,y}),\ \theta = \mathrm{atan2}(\mathbf{n}_{z,x}, \mathbf{n}_{x,x}),$$

где atan2(x, y) – функция арктангенса двойного аргумента [11].

Углы наклона внутренней тарелки совпадают с углами наклона внешней тарелки, а ее положение вычисляется путем смещения положения внешней тарелки вдоль оси вращения винта на известную величину Δh.

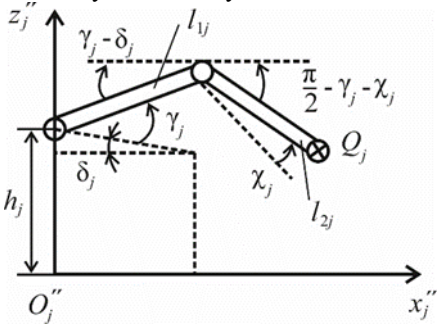

Рис. 4. Тяга управления лопастью винта

3.2. Вычисление установочных углов лопастей винта

Лопасти винта связаны с внутренней тарелкой посредством тяг. Данная конструкция (см. рис. 4) представляет собой двухзвенный механизм, одно звено которого является лопастью винта. Движение рассматриваемого механизма описывается с помощью углов γ_j и χ_j, $j = \overline{1,M}$, где M – число лопастей винта. При этом углы δ_j

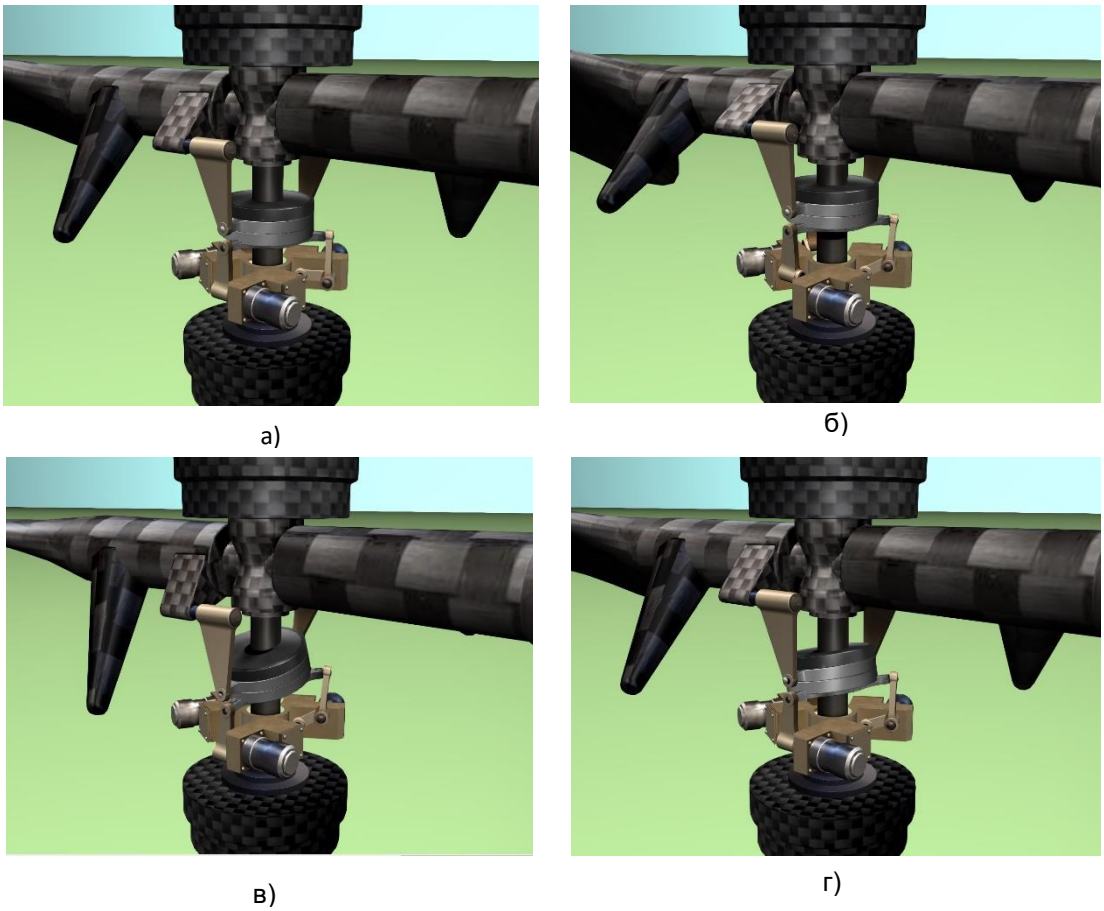

Рис. 5. Моделирование движения автомата перекоса

задают начальные значения установочных углов лопастей.

В двумерной системе координат $O_j''x_j''z_j''$ координаты точек Q_j крепления тяг с внутренней тарелкой выражаются через углы γ_j и χ_j следующим образом

$$Q_{xj}'' = l_{1j}\cos\left(\gamma_j - \delta_j\right) + l_{2j}\cos\left(\pi/2 - \gamma_j - \chi_j\right) = \\ = l_{1j}\cos\left(\gamma_j - \delta_j\right) + l_{2j}\sin\left(\gamma_j + \chi_j\right); \qquad (3)$$

$$Q_{zj}'' = h_j + l_{1j}\sin\left(\gamma_j - \delta_j\right) - l_{2j}\sin\left(\pi/2 - \gamma_j - \chi_j\right) = \\ = h_j + l_{1j}\sin\left(\gamma_j - \delta_j\right) - l_{2j}\cos\left(\gamma_j + \chi_j\right),$$

где l_{1j} и l_{2j} – длины звеньев механизма, h_j – расстояние между внутренней тарелкой и шарниром лопасти в исходном состоянии.

Так как точки Q_j тяг находятся на вращающейся внутренней тарелки, то их положение также зависит от угла поворота винта. Эти точки в системе координат O_1XYZ вычисляются как

$$Q_j = O + \Delta h\mathbf{n}_Z + \cos\psi_j\mathbf{n}_x + \sin\psi_j\mathbf{n}_y,$$

где $\mathbf{n}_Z = (0,0,1)^{\mathrm{T}}$, ψ_j – угол поворота точки Q_j при вращении внутренней тарелки.

Для вычисления координат Q_{xj}'' и Q_{zj}'' точки Q_j преобразуются в систему координат $O_j''x_j''z_j''$. Система уравнений (3) является нелинейной относительно неизвестных углов γ_j и χ_j. Для решения этой системы воспользуемся численным методом Ньютона-Рафсона, предварительно представив ее в следующем виде

$$f_4(\gamma_j,\chi_j) = l_{1j}\cos\left(\gamma_j - \delta_j\right) + \\ + l_{2j}\sin\left(\gamma_j + \chi_j\right) - Q_{xj}'' = 0;$$

$$f_5(\gamma_j,\chi_j) = h_j + l_{1j}\sin\left(\gamma_j - \delta_j\right) - \\ - l_{2j}\cos\left(\gamma_j + \chi_j\right) - Q_{zj}'' = 0.$$

Тогда на каждом шаге итерации метода задача сводится к решению системы линейных уравнений второго порядка

$$\frac{\partial f_4}{\partial\gamma_j}\Delta\gamma_j^{(k)} + \frac{\partial f_4}{\partial\chi_j}\Delta\chi_j^{(k)} = -f_4(\gamma_j^{(k)},\chi_j^{(k)}); \\ \frac{\partial f_5}{\partial\gamma_j}\Delta\gamma_j^{(k)} + \frac{\partial f_5}{\partial\chi_j}\Delta\chi_j^{(k)} = -f_5(\gamma_j^{(k)},\chi_j^{(k)}), \qquad (4)$$

где k – номер итерации, $\Delta\gamma_j^{(k)} = \gamma_j^{(k+1)} - \gamma_j^{(k)}$, $\Delta\chi_j^{(k)} = \chi_j^{(k+1)} - \chi_j^{(k)}$,

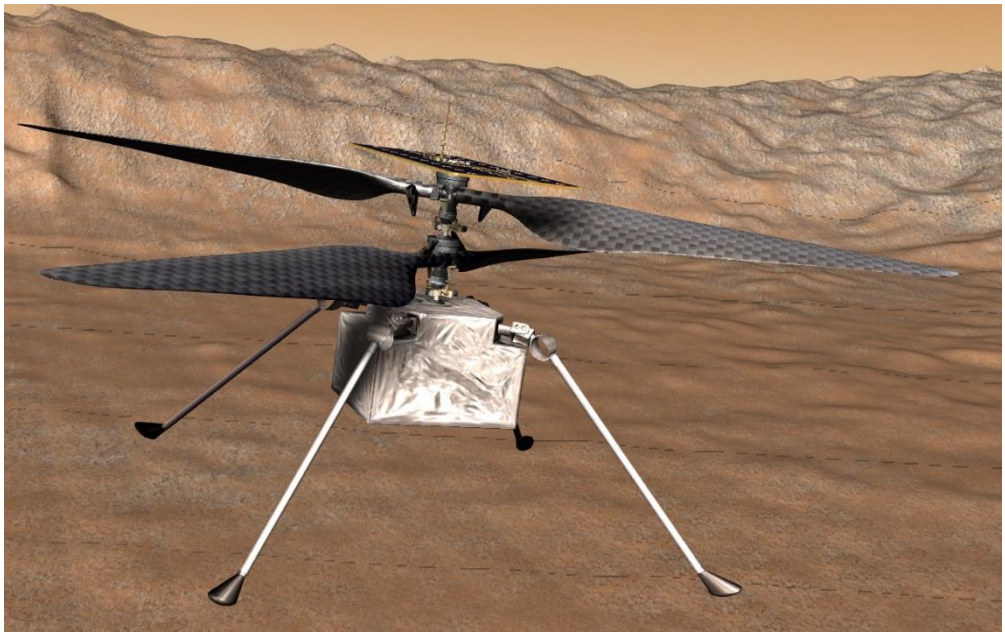

Рис. 6. Моделирование движения марсианского вертолета соосной схемы

$$\frac{\partial f_4}{\partial \gamma_j} = -l_{1j}\sin\left(\gamma_j - \delta_j\right) + l_{2j}\cos\left(\gamma_j + \chi_j\right),$$

$$\frac{\partial f_5}{\partial \gamma_j} = l_{1j}\cos\left(\gamma_j - \delta_j\right) + l_{2j}\sin\left(\gamma_j + \chi_j\right),$$

$$\frac{\partial f_4}{\partial \chi_j} = l_{2j}\cos\left(\gamma_j + \chi_j\right),\ \frac{\partial f_5}{\partial \chi_j} = l_{2j}\sin\left(\gamma_j + \chi_j\right).$$

Полученная система линейных уравнений решается стандартным образом. Итерации выполнятся до тех пор, пока не будут выполнены условия $\left|\Delta\gamma_j^{(k)}\right| \le \varepsilon$ и $\left|\Delta\chi_j^{(k)}\right| \le \varepsilon$.

На выходе моделирования движения автомата перекоса получаем новые установочные углы лопастей $\gamma_j + \delta_j$.

4. Результаты моделирования

Предлагаемые в статье методы и подходы для моделирования движения автомата перекоса летательных аппаратов вертолетного типа были реализованы в комплексе виртуального окружения VirSim [10], разработанном в ФГУ ФНЦ НИИСИ РАН. Для этого были созданы программные модули для моделирования динамики винта виртуальных летательных аппаратов. Эти модули обеспечивают расчет динамики всех электроприводов винта, реализацию кинематики механизма автомата перекоса, а также вычисление подъемной силы и момента сопротивления винта. Апробация предложенных в статье решений проводилась на примере моделирования движения виртуальной модели марсианского вертолета соосной схемы [12], в котором два несущих винта вращаются в противоположном направлении относительно друг друга и управляются с помощью автоматов перекоса. На рис 5 приводятся различные состояния верхнего автомата перекоса, где а) его исходное состояние, б) изменение общего шага лопастей, в) изменение циклического шага лопастей в поперечном направлении винта путем поворота тарелок на угол φ по оси тангажа, г) изменение циклического шага лопастей в продольном направлении винта путем поворота тарелок на угол θ по оси крена. В этой модели изменение общего шага установки лопастей винта обеспечивает управление вертикальным движением летательного аппарата, в то время как изменение циклического шага верхнего винта предназначено для его стабилизации, а нижнего винта – для управления его горизонтальным движением. Апробация проводилась с использованием созданной виртуальной модели участка поверхности Марса (см. рис. 6) с реализацией основных движений вертолета соосной схемы (его взлет, посадка и маневрирование в горизонтальной плоскости). Результаты апробации показали, что для решения систем нелинейных уравнений (2) и (4) требуется не более 5 итераций и все необходимые вычисления для реализации динамики объектов укладываются в 10 мс, тем самым обеспечивая моделирование движения летательных аппаратов вертолетного типа в масштабе реального времени.

5. Заключение

В работе предложены методы и подходы моделирования движения механизма автомата перекоса несущих винтов виртуальных летательных аппаратов, основанные на вычислении координат его составных частей без учета их динамики. Преимущество такого подхода перед другими методами состоит в том, что он не требует существенных вычислительных затрат и обеспечивает правильную работу механизма при вращении винта с высокой скоростью. Полученные в статье результаты могут быть использованы в имитационно-тренажерных комплексах и виртуальных лабораториях с целью обучения операторов навыкам управления летательными аппаратами вертолетного типа.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0002 «Математическое моделирование многомасштабных динамических процессов и системы виртуального окружения».

Simulation of Swashplate Motion for Virtual Martian Rotorcraft Models

E. V. Strashnov

Abstract. The paper considers the task for rotor swashplate motion simulation of helicopter-type aircraft in virtual environment systems. To solve this task, it is proposed an approach in which the coordinates of mechanism component parts are computed without taking into account their dynamics. When implementing this approach, the Newton-Raphson method was used to solve systems of nonlinear equations. The approbation of methods and approaches proposed in the paper was carried out in developed virtual environment complex using the example of a virtual Martian coaxial helicopter model motion simulation. Approbation results showed the adequacy and effectiveness of solutions proposed in the paper and their applicability for virtual environment systems.

Keywords: rotorcraft, rotor, swashplate, blade pitch, Newton–Raphson method, virtual environment systems

Литература

1. G.J. Leishman. Principles of helicopter aerodynamics. Second edition, Cambridge university press, 2016, 866 p.

2. J.M. Seddon, S. Newman. Basic Helicopter Aerodynamics. 3rd ed. John Wiley & Sons, 2011.

3. B. Mettler. Identification modeling and characteristics of miniature rotorcraft. Kluwer, Norwell, 2002.

4. A.R.S. Bramwell, D. Balmford, G. Done. Bramwell's helicopter dynamics. Elsevier, 2001.

5. D. Schafroth. Aerodynamics, modeling and control of an autonomous micro helicopter. Ph.D. Thesis. ETH Zurich, 2010.

6. M. Sparasci. Nonlinear modeling and control of coaxial rotor UAVs with application to the Mars helicopter. Master's Thesis. Politecnico di Milano University, Milan, Italy, 2022.

7. H. Wang, P. Eberhard and Z. Lin. Modeling and simulation of closed loop multibody systems with bodies-joints composite modules. «Multibody Syst Dyn», Vol. 24 (2010), 389–411.

8. J. García de Jalón, E. Bayo. Kinematic and dynamic simulation of multibody systems: the real-time challenge. Springer Science & Business Media, 2012.

9. A.A. Shabana. Dynamics of multibody system. Fifth edition. Cambridge university press, 2020.

10. М.В. Михайлюк, А.В Мальцев, П.Ю. Тимохин, Е.В. Страшнов, Б.И. Крючков, В.М. Усов. Система виртуального окружения VirSim для имитационно-тренажерных комплексов подготовки космонавтов. «Пилотируемые полеты в космос», Т. 37 (2020), № 4, 72–95.

11. Функция atan2. URL: https://en.wikipedia.org/wiki/Atan2 (дата обращения 17.05.2024)

12. J. Balaram, M. Aung, M.P. Golombek. The Ingenuity helicopter on the Perseverance rover. «Space Science Reviews», Vol. 217 (2021), No. 4, 1–11.

Обобщение задачи составления многопроцессорного расписания с прерываниями

М.Г. Фуругян[1]

[1]ФИЦ ИУ РАН, Москва, Россия, rtsccas@yandex.ru

Аннотация. Рассматривается задача составления многопроцессорного расписания для комплекса работ, допускающих прерывания и переключения с одного процессора на другой. Предполагается, что обработка прерываний и переключений требует временных издержек. Это условие переводит задачу из класса полиномиально разрешимых в класс NP-трудных. Разработан алгоритм, основанный на методике В.С. Танаева составления многопроцессорного расписания без учета затрат на прерывания и переключения. Методика включает в себя процедуру упаковки для случая, когда работы имеют общий директивный срок, а также процедуру сведения исходной задачи к потоковой для случая произвольных директивных интервалов. При этом используется также известный псевдополиномиальный алгоритм составления допустимого многопроцессорного расписания для непрерываемых работ с общим директивным сроком.

Ключевые слова: многопроцессорная система, директивный интервал, допустимое расписание, процедура упаковки, потоковая сеть

1. Введение

По вопросам планирования работ и составления расписаний имеется большое число публикаций. Отметим такие фундаментальные работы, как [1, 2]. В [1] рассматриваются различные задачи по теории расписаний как для однопроцессорных, так и для многопроцессорных систем. Авторы предлагают алгоритмы планирования непрерываемых работ, а также для работ, допускающих прерывания и переключения с одного процессора на другой. Большое внимание уделено вопросам вычислительной сложности алгоритмов. Проводится большой список NP-трудных задач. В [2], помимо задач составления расписаний, исследуются различные задачи дискретной оптимизации.

В [3] исследуются задачи составления однопроцессорных расписаний для непрерываемых работ с критерием минимизации максимального запаздывания. Автором введено понятие расстояния между задачами, что позволило разработать эффективные алгоритмы.

В [4, 5] на основе метода ветвей и границ разработаны алгоритмы решения задач планирования в финансовой и экономической сферах. Эти публикации представляют особый интерес, поскольку в них рассмотрены задачи с нефиксированными параметрами, такими, как длительности выполнения работ и объемы имеющихся ресурсов.

В [6, 7] рассмотрены задачи планирования работ в многопроцессорных и многоядерных системах реального времени. В таких системах существуют жесткие временные ограничения на выполнения работ, которые в ряде случаев составляют доли секунды. Поэтому очень важным является составление оптимальных по быстродействию расписаний. Для этого авторы используют аппарат сетей Петри с остановкой таймера и временные диаграммы.

В работах [1 – 7] используемые ресурсы являются не складируемыми, т.е. такими, которые могут использоваться многократно. К таким ресурсам относятся, например, машины, станки, приборы. В отличие от них, складируемые ресурсы повторно использоваться не могут. К таким ресурсам относятся, например, финансы, горючие материалы, электроэнергия.

В [8, 9] исследованы задачи планирования работ в системах с неоднородным комплексом ресурсов, который включает в себя как складируемые, так и не складируемые ресурсы. Методика решения таких задач основана на их сведении к потоковым задачам в сетях специального вида.

В настоящей статье рассматривается обобщение задачи составления многопроцессорного расписания с директивными интервалами, допускающего прерывания и переключения. В отличие от указанных выше публикаций, предполагается, что прерывания и переключения требуют временных затрат. Наличие этого условия приводит к тому, что задача переходит из класса полиномиально разрешимых задач в класс NP-трудных задач. Решение задачи основано на методике, предложенной в [1] и состоящей из построения сети специального вида и поиска в ней

максимального потока. Дополнительно используется псевдополиномиальный алгоритм составления расписания выполнения непрерываемых работ с общим директивным сроком [10].

2. Постановка задачи

Для выполнения комплекса работ (заданий) $W = \{w_1, w_2, \dots, w_n\}$ имеется m идентичных процессоров $p_1, p_2, \dots, p_m$. Каждая работа может выполняться любым процессором. Для работы w_i известна длительность ее выполнения t_i и директивный интервал $[b_i, f_i]$ (работа w_i может быть начата не ранее момента времени b_i и должна быть завершена не позднее момента времени f_i), $i = \overline{1,n}$.При выполнении работ допускаются их прерывания и переключения с одного процессора на другой. В отличие от [1], предполагается, что прерывание и переключение с одного процессора на другой требуют дополнительных временных затрат в объеме σ каждого из этих двух процессоров. А именно, если некоторая работа выполняется процессором p_{l_1} и в момент времени τ_1 она прерывается, то этот процессор выполняет дополнительную работу в интервале $[\tau_1;\ \tau_1 + \sigma]$. Далее, прерванная работа может быть возобновлена в момент времени $\tau_1 \geq \tau_1 + \sigma$ на некотором процессоре p_{l_2} (возможно, на том же процессоре p_{l_1}), который сначала выполняет дополнительную работу в интервале $[\tau_2;\ \tau_2 + \sigma]$, а затем прерванную работу. Предполагается, что

$$t_i + 2\sigma \leq f_i - b_i \qquad (1)$$

при всех $i = \overline{1,n}$, т. е. каждая работа может быть выполнена одним процессором в своем директивном интервале, включая обработку одного прерывания и одного возобновления прерванной работы.

Не допускается параллельное выполнение одной работы несколькими процессорами и параллельное выполнение нескольких работ одним процессором.

Расписание выполнения комплекса работ *W* показывает для каждой работы, в какие моменты времени какими процессорами она выполняется. Допустимое расписание – это такое расписание, при котором каждая работа полностью выполняется в своем директивном интервале.

Задача состоит в том, чтобы определить, существует ли допустимое расписание выполнения комплекса работ *W*, и построить его в случае положительного ответа. Отметим, что в [11] был разработан алгоритм для случая, когда директивные интервалы всех работ совпадают. Там же показано, что сформулированная задача является NP-трудной даже в том случае, когда директивные интервалы у всех работ совпадают.

3. Модификация алгоритма упаковки

Рассмотрим сначала случай, когда директивные интервалы у всех работ совпадают, т.е. $b_i = 0, f_i = F$ при всех $i = \overline{1,n}$. В [1] описан алгоритм упаковки для случая, когда затраты на прерывания и переключения не учитываются. Ниже приводится модификация этого алгоритма для случая, когда прерывания и переключения требуют временных издержек, как это описано в разд. 2. В этом случае условие (1) трансформируется в неравенство

$$t_i + 2\sigma \leq F \qquad (2)$$

при всех $i = \overline{1,n}$.

Лемма. 1) Необходимым условием существования допустимого расписания является выполнение неравенства

$$\sum_{i=1}^{n} t_i \leq mF. \qquad (3)$$

2) Достаточным условием существования допустимого расписания является выполнение неравенства

$$\sum_{i=1}^{n} t_i + 2(m-1)\sigma \leq mF. \qquad (4)$$

Доказательство. 1) Если неравенство (3) не выполнено, то это означает, что суммарная длительность работ *W* превосходит суммарное процессорное время. В этом случае допустимого расписания не существует.

2) Существование допустимого расписания при выполнении неравенства (4) следует из описанного ниже модифицированного алгоритма упаковки. Лемма доказана.

Модифицированный алгоритм упаковки

Шаг 1. Выполнить работу w_1 на процессоре p_1 без прерываний в интервале $[o;\ t_1]$. Положить $\tau = t_2, i = 1, j = 1$.

Шаг 2. Положить $i = i + 1$. Если $i \leq n$, то перейти на шаг 3; в противном случае перейти на шаг 8.

Шаг 3. Если $\tau + t_i \leq F$, то перейти на шаг 4. Если $\tau + t_i = F$, то перейти на шаг 5. Если $\tau + t_i > F$ и $\tau + \sigma \geq F$, то перейти на шаг 7.

Шаг 4 Выполнять работу w_i на процессоре p_j в интервале $[\tau;\ \tau + t_i]$. Перейти на шаг 2.

Шаг 5. Выполнять работу w_i на процессоре p_j в интервале $[\tau;\ \tau + t_i]$. Положить $j = j + 1$. Перейти на шаг 2.

Шаг 6. Выполнять работу w_i на процессоре p_{j+1} в интервале $[0;\ t_i - (F - \tau + \sigma)]$. В момент

$t_i - (F - \tau + \sigma)$ выполнение работы w_i прерывается. Далее, в интервале $[t_i - (F - \tau - \sigma);\ t_i - (F - \tau - \sigma) + \sigma]$ процессор p_{j+1} выполняет дополнительную работу по обработке прерывания и переключения работы w_i. Далее, в интервале $[\tau;\ \tau + \sigma]$ процессор p_j снова выполняет дополнительную работу по обработке прерывания и переключения работы w_i. После чего процессор p_j возобновляет выполнение работы w_i в интервале $[\tau + \sigma; F]$. Положить $j = j + 1$. Перейти на шаг 2.

Шаг 7. Выполнять работу w_i без прерываний на процессоре p_{j+1} в интервале $[0;\ t_i]$. $j = j + 1$. Перейти на шаг 2.

Шаг 8. Расписание построено. Завершение алгоритма.

Дадим некоторые пояснения к описанному алгоритму. Выполнение работы w_1 на процессоре p_1 без прерываний и переключений (шаг 1) возможно в силу условия (2). На шаге 2 выполняется проверка, все ли работы назначены на процессоры. На шаге 3 выполняется проверка возможности исполнения очередной работы на текущем процессоре. На шаге 4 и шаге 5 очередная работа полностью выполняется на текущем процессоре, если она может завершиться не позднее момента времени *F*. В противном случае возможны два варианта. В первом варианте она сначала выполняется на следующем процессоре, а завершается на текущем процессоре (шаг 6). При этом учитываются затраты на прерывания и переключения. Во втором варианте текущая работа полностью выполняется на следующем процессоре без прерываний и переключений (шаг 7). Шаг 8 завершает алгоритм.

Отметим, что в результате работы писанного алгоритма число прерываний и переключений не более $m - 1$, а продолжительность их обработки не превосходит $2(m - 1)\sigma$. Поэтому если условие (2) будет выполнено, то это гарантирует существование допустимого расписания. Вычислительная сложность описанного алгоритма составляет $O(n)$.

4. Сокращение числа прерываний и переключений для случая одинаковых директивных интервалов

В предыдущем разделе предполагалось, что выполняется неравенство (4), которое гарантирует существование допустимого расписания в случае общего директивного срока у всех работ. В настоящем разделе будем предполагать, что неравенство (4) не выполняется, т.е.

$$\sum_{i=1}^{n} t_i + 2(m-1)\sigma > mF \qquad (5)$$

а неравенство (3) выполняется, т.к. оно является необходимым условием существования допустимого расписания.

Упорядочим работы по не убыванию их длительностей, т.е. будем предполагать, что $t_1 \le t_2 \le \cdots \le t_n$, Предлагаемый алгоритм заключается в том, чтобы сократить число прерываний и переключений. Для этого множество процессоров разбивается на два подмножества: $P_1 = \{p_1, \dots, p_k\}$ и $P_2 = \{p_{K+1}, \dots, p_m\}$. Используя алгоритм, описанный в [10]. Построим допустимое расписание без прерываний и переключений на процессорах P_1 для некоторого подмножества работ $W_1 \subseteq W$. Это расписание получается путем последовательного выбора работ из W и построения точек в *k*-мерном кубе с ребром длины *F*. Вычислительная сложность этого алгоритма $O(kT^k)$. Если для работ $W_2 = W \backslash W_1$ и $m - k$ процессоров выполняется условие существования допустимого расписания, т.е. неравенство

$$\sum_{i \epsilon W_2} \boldsymbol{t_i} + \boldsymbol{2(m-k-1)\sigma \le (m-k)F}, \qquad \boldsymbol{(6)}$$

то с помощью модифицированного алгоритма упаковки построим допустимое расписание для W_2. Объединяя его с ранее построенным расписанием для W_1, построим окончательное допустимое расписание для W. Если же неравенство (6) не выполняется, то число *k* следует увеличить на одну единицу. Таким образом, алгоритм выглядит следующим образом.

Шаг 1. Положить $k = 1$.

Шаг 2. Построить допустимое расписание без прерываний и переключений на процессорах P_1. Пусть W_1 – множество работ, вошедшее в это расписание.

Шаг 3. Если выполнено неравенство (6), то построить для W_2 расписание на процессорах P_2 с прерываниями и переключениями; завершение алгоритма. Если неравенство (6) не выполнено, то перейти на шаг 4.

Шаг 4. Если $k = m$, то решение не найдено; завершение алгоритма. Если $k < m$, то положить $k = k + 1$ и перейти на шаг 2.

Вычислительная сложность описанного алгоритма $O(m(T^m + n))$.

5. Произвольные директивные интервалы

Перейдем к рассмотрению случая произвольных директивных интервалов. Сначала приведем краткое описание алгоритма, предложенного в [1] в предположении отсутствия издержек

на обработку прерываний и переключений.

Пусть $y_0 < y_1 < \cdots < y_p$ – все различные значения $b_i, f_i, i = \overline{1,n}$; $I_j = [y_{j-1};\ y_j], \delta_j = y_j - y_{j-1},\ j = \overline{1,p}$. Определим потоковую сеть $G = (V, A)$, V – множество вершин, A – множество ориентированных дуг; $V = \{u, v, I_j, w_i\}$, u – источник, v – сток, $A = \{(u, I_j), (I_j, w_i), (w_i, v)\}, j = \overline{1,p},\ i = \overline{1,n}$. Дуга (I_j, w_i) включается в A, если $I_j \subseteq [b_i;\ f_i]$. Пропускные способности U дуг определяются следующим образом: $U(u, I_j) = m\delta_j$, $U(I_j, w_i) = \delta_j$, $U(w_i, v) = t_i$. В [1] доказано, что допустимое расписание существует в том и только том случае, когда в сети G существует поток g, для которого $g(w_i, v) = t_i$ при всех $i = \overline{1,n}$. Если такой поток существует и $g(I_j, w_i) > 0$, то работе w_i в интервале I_j выделяется $g(I_j, w_i)$ единиц процессорного времени. Расписание для каждого интервала I_j строится с помощью модифицированного алгоритма упаковки, описанного в разделах 3, 4.

6. Сокращение числа прерываний и переключений для случая произвольных директивных интервалов

Как следует из разд. 4, прерывания и переключения, помимо возникающих внутри каждого интервала $I_j, j = \overline{1,p}$, возможны и на стыках интервалов I_j и $I_{j+1}, j = \overline{1,p} - 1$. В этом разделе будем исследовать вопрос о сокращении числа таких прерываний и переключений.

Как следует из разд. 2, 3, расписание внутри каждого интервала $I_j, j = \overline{1,p}$, может быть представлено в виде матрицы $\|w_{ki_r}^j\|$, $k = \overline{1,m}$, $1 \le r \le n$. Здесь $w_{ki_r}^j$ – работа w_{i_r}, выполняемая в интервале I_j процессором p_k. Кроме того, при $k = \overline{1,k(j)}$, $k(j) \le m$, работы выполняются без прерываний и переключений.

Предлагаемый алгоритм состоит в следующем. Рассмотрим пару интервалов I_j и $I_{j+1}, j = \overline{1,p} - 1$. Предположим, что $w_{1i_1}^j = w_{k_2i_2}^{j+1}$ при некоторых $i_1, i_2, 1 \le k_2 \le k(j+1)$. Тогда меняем местами строки 1 и k_2 в матрице $\|w_{ki_r}^{j+1}\|$. Далее, работу $w_{1i_1}^j$ выполняем в конце интервала I_j, а работу $w_{k_2i_2}^{j+1}$ – в начале интервала I_{j+1}. В этом случае работа $w_{1i_1}^j$ будет выполняться без прерываний и переключений в пределах интервалов I_j и I_{j+1}.

Далее, следует рассмотреть элементы $w_{1i_2}^j$ и $w_{k_ri_r}^{j+1}$ при $2 \le k_2 \le k(j+1)$ и выполнять действия аналогичным образом.

Вычислительная сложность описанной процедуры составляет $O(n^2p)$ или $O(n^3)$.

7. Заключение

Исследована задача составления многопроцессорного расписания для комплекса работ, допускающих прерывания и переключения с одного процессора на другой. Предполагается, что обработка прерываний и переключений требует временных издержек, в следствие чего задача является NP-трудной. Разработанный алгоритм основан на методике В.С. Танаева составления многопроцессорного расписания без учета затрат на прерывания и переключения. Алгоритм включает в себя: процедуру упаковки для случая, когда работы имеют общий директивный срок, процедуру сведения исходной задачи к потоковой для случая произвольных директивных интервалов, псевдополиномиальный алгоритм составления допустимого многопроцессорного расписания для непрерываемых работ с общим директивным сроком. Для отдельных частей алгоритма получены оценки вычислительной сложности.

Generalization of the Problem of Creating a Multiprocessor Schedule with Interrupts

Meran Furugyan

Abstract. We consider the problem of creating a multiprocessor schedule for a set of jobs that allow interruptions and switching from one processor to another. It is assumed that processing interrupts and switches requires time overhead. This condition transfers the problem from the class of polynomially solvable to the class of NP-hard ones. An algorithm has been developed based on the methodology of V.S. Tanaev for compiling a multiprocessor schedule without taking into account the costs of interruptions and switching. The technique includes a packing procedure for the case when jobs have a common deadline, as well as a procedure for reducing the original problem to a network flow problem for the case of arbitrary deadlines. In this case, the well-known pseudo-polynomial algorithm for creating an admissible multiprocessor schedule for continuous jobs with a common deadline is also used.

Keywords: multiprocessor system, admissible schedule, directive interval, packing procedure, flow network

Литература

1. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийныесистемы. М.: Наука, 1984, 383 с.

2. Brucker P. Scheduling Algorithms. Heidelberg: Springer, 2007, 378 с.

3. Лазарев А.А. Теория расписаний. Методы и алгоритмы. –М.: ИПУ РАН, 2019,407 с.

4. А.В. Мищенко, П.С. Кошелев. Оптимизация управления работами логистического проекта в условиях неопределенности // Известия РАН. Теория и системы управления. (2021), № 4, 123-134.

5. М.А. Горский, А.В. Мищенко, Л.Г. Нестерович, М.А. Халиков. Некоторые модификации целочисленных оптимизационных задач с учетом неопределенности и риска // Известия РАН. Теория и системы управления. (2022), № 5, 106-117.

6.А.Б. Глонина, В.В. Балашов. О корректности моделирования модульных вычислительных систем реального времени с помощью сетей временных автоматов // Моделирование и анализ информационных систем. (2018), Т. 25, № 2, 174 – 192.

7. А.Б. Глонина. Инструментальная система проверки выполнения ограничений реального времени для конфигураций модульных вычислительных систем // Вестн. МГУ. Сер. 15. Вычисл. математика и кибернетика. (2020), № 3, 16 – 29.

8. М.Г. Фуругян. Планирование вычислений в многопроцессорных АСУ реального времени с дополнительным ресурсом //Автоматика и телемеханика. (2015), №3, 144 – 150.

9. М.Г. Фуругян. Составление расписаний в многопроцессорных системах с несколькими дополнительными ресурсами // Известия РАН. Теория и системы управления. (2017), № 2, 57 – 66.

10. М.Г. Фуругян. Некоторые алгоритмы решения минимаксной задачи составления многопроцессорного расписания // Известия РАН. Теория и системы управления. (2014), №2, 50 - 56.

11. М.Г. Фуругян. Составление расписаний в многопроцессорной системе с учетом затрат на прерывания // В Сб. Математическое и компьютерное моделирование сложных систем: теоретические и прикладные аспекты. Труды НИИСИ РАН. Т. 6, № 2, 57 – 61.

О некоторых простых способах синхронизации параллельных программ

А.А. Бурцев[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, burtsev@niisi.msk.ru

Аннотация. Статья посвящена описанию простых способов синхронизации двух параллельных программ, исполняемых на разных вычислительных ядрах одной компьютерной установки, имеющих доступ к общей памяти. Описываемые способы взаимодействия программ можно обеспечить на основе обычных средств, имеющихся почти в каждом языке программирования. И для их реализации не требуется применять какие-либо особые процессорные команды или вызовы специальных функций операционной системы.

Ключевые слова: многоядерные микропроцессоры, способы синхронизации и взаимодействия параллельных программ, рандеву.

1. Введение

Современные высокопроизводительные системы, как правило, имеют в своём составе не один, а несколько микропроцессоров [1], совместно работающих с общей памятью. Это позволяет повысить общую производительность компьютерной установки, так как обеспечивается возможность запускать на ней не одну, а сразу несколько компьютерных программ, которые будут исполняться на ней в одно и то же время, т.е. параллельно.

Но если программы, параллельно исполняемые на разных микропроцессорных ядрах, не являются независимыми, а как-то связаны между собой и призваны вместе решать одну общую задачу, то исполнение таких программ необходимо тщательно синхронизировать, чтобы обеспечить их согласованную работу относительно друг друга.

Для обеспечения синхронизации параллельных программ предлагаются разнообразные средства: семафоры, сигналы, почтовые ящики, мьютексы [2-4]. Такие средства обеспечиваются, как правило, вызовами особых функций операционной системы или специализированной библиотеки. А для их эффективной реализации даже используются специальные команды процессора (test-and-set [5], LL/SC [6]), обеспечивающие исключительный кратковременный монопольный доступ к выделенным позициям памяти.

Однако, в некоторых случаях применять заготовленный арсенал средств синхронизации не представляется возможным или целесообразным. Например, при разработке программы, которая должна будет функционировать на “голой” машине, не оснащённой какой-либо ОС, и/или не имеет возможности использовать специализированные команды доступа к памяти.

Типичным примером такой программы может служить тестовая программа, проверяющая работу подключённых к компьютеру разнообразных периферийных устройств. Подобные тестовые программы должны уметь функционировать ещё до стадии загрузки какой-либо операционной системы. Обычно они должны также удовлетворять ещё и жёстким ограничениям по размеру занимаемой памяти, поэтому при их разработке приходится минимизировать использование стандартных библиотек.

Воспользоваться библиотечными функциями, приготовленными для обеспечения синхронизации, не всегда удаётся ещё по одной причине. Как правило, эти функции содержат в себе цикл ожидания отклика от той программы-партнёра, с которой требуется наладить взаимодействие. Но если такого отклика не последует, то вызванная функция «намертво» застопорит исполнение программы, её вызвавшей. Что как раз таки неприемлемо для функционирования особо критичных программ.

Так, например, тестовые программы, работающие с периферийными устройствами, должны уметь всегда корректно завершаться, невзирая ни на какие исключительные события. Будь то «зависание» устройства по причине его сбоя или критической ошибки, возникшей при передаче очередной порции данных.

Поэтому при разработке такого рода программы желательно обеспечивать синхронизацию работы отдельных её программных компонент, призванных исполняться на разных микропроцессорных ядрах, используя лишь обычные средства традиционного языка программирования высокого уровня.

Далее покажем, как можно обеспечить синхронизацию двух программ, запускаемых на разных ядрах микропроцессорной системы, используя обычные возможности языка Си.

2. Приёмы синхронизации для двух параллельных программ

Для полноценного взаимодействия двух программ, имеющих доступ к общей памяти, достаточно обеспечить им средства исключительного доступа к совместно используемым критическим ресурсам, которые бы не допускали взаимных блокировок. Требуется также обеспечить примитивные средства, которые позволили бы программам регулировать порядок выполнения своих действий относительно связанных с ними действий другого партнёра.

2.1. Критические секции

Для обеспечения взаимно исключительного доступа к критическому ресурсу со стороны двух параллельных программ можно применять известный алгоритм Деккера. Его структурированный вариант, предложенный Дейкстрой, подробно излагается, например, в [4, с.13].

Представим здесь вариант этого алгоритма на языке Си, оформив в виде функций основные его операции: инициализации, входа в критическую секцию и выхода из неё для программ каждого ядра по отдельности:

```
int z0,z1; //флажки запросов ресурса
int pr; // кому разрешается захватить ресурс
void Init(void) {z0=0;z1=0;pr=0}
void Ent0(void) // вход для программы 0-го ядра
{ z0=1;pr=1; while(z1&&(pr==1); }
void Fin0(void){z0=0};//выход для 0-го ядра
void Ent1(void) // вход для программы 1-го ядра
{ z1=1;pr=0; while(z0&&(pr==0); }
void Fin1(void){z1=0};//выход для 1-го ядра
```

Предполагается, что каждая программа соблюдает определённые правила при доступе к критическому ресурсу. Это значит, что программа 0-го ядра при входе в критическую секцию вызывает функцию **Ent0**(), а при выходе из неё − функцию **Fin0**(). А программа 1-го ядра вызывает соответственно функцию **Ent1**() при входе и функцию **Fin1**() при выходе из своей критической секции. И, конечно же, перед использованием критического ресурса сначала вызывается (из любого ядра) операция инициализации **Init**().

2.2. Флажки готовности

Допустим, на разных ядрах параллельно исполняются две программы: **A** на 0-м ядре и **B** на 1-ом ядре. И каждая из них должна выполнить два блока действий: **A1, A2** и **B1, B2**. Но при этом требуется обеспечить, чтобы каждая из них начала исполнять свой второй блок действий только после того, как другая программа закончила исполнение своего первого блока.

В этом случае требуется организовать для этих двух программ такой вид синхронизации, который давно применяется при взаимодействии программ-драйверов с периферийным устройством. Выполнив свою часть работы, устройство сообщает об этом установкой флажка готовности − определённого бита в регистре состояния. А программа-драйвер приостанавливается в цикле ожидания, пока не обнаружит, что требуемый флажок установлен.

Позаимствуем такой приём для обеспечения требуемой синхронизации двух параллельных программ. Пусть каждая из них, выполнив свою первую часть работ, устанавливает свой флажок готовности, а затем дожидается установки флажка готовности другой программы перед тем, как приступить ко второй части своей работы. Приготовим для такой синхронизации соответствующие функции на языке Си:

```
int f0,f1; //флажки готовности программ
void Init0(void) {f0=0;}
void Init1(void) {f1=0;}
void Wait0(void)// ожидание для 0-го ядра
{ f0=1; while(f1==0); }
void Wait1(void)// ожидание для 1-го ядра
{ f1=1; while(f0==0); }
```

Теперь, чтобы обеспечить требуемую синхронизацию, программам 0-го и 1-го ядра следует соблюдать такой порядок действий:

```
    Init(); // предварительная инициализация
// схема для программы 0-го ядра
 Init0();... A1; Wait0(); A2;
// схема для программы 1-го ядра
 Init1();... B1; Wait1(); B2;
```

2.3. Точки встречи

Пусть запускаемые на разных ядрах программы **A** и **B** должны исполнить последовательности из нескольких действий:

```
A1; A2; ... An; // для программы 0-го ядра
B1; B2; ... Bn; // для программы 1-го ядра
```

так, чтобы очередное действие **Aj** программы 0-го ядра совпадало по времени с очередным действием **Bj** программы 1-го ядра (**j**=1,2,…,**n**). Это значит, что для программ **A** и **B** нужно обеспечить такую синхронизацию, чтобы пары действий **Ai** и **Bi** исполнялись бы параллельно:

```
A1||B1; A2||B2; ... An||Bn;
```

Для этого организуем в этих программах своеобразные точки встречи, так называемые **рандеву**. В таких точках одна программа должна дождаться, когда на свою точку встречи подойдёт другая программа, и только после этого они обе смогут продолжать свою дальнейшую работу. Если бы у них предполагалась только одна точка встречи, то для её организации можно было применить описанное ранее решение с флажками готовности.

Но для поставленной здесь задачи синхронизации таких точек встреч у этих двух программ может быть много (**n**). Их потребуется расставить в этих программах перед каждым действием **Aj** и **Bj**, исполнение которых должно начинаться одновременно.

Способ синхронизации с ожиданием флажков готовности распространим на случай не одной, а нескольких точек встречи. Для этого вместо флажков, принимавших только два значения 0 и 1, будем использовать счётчики состояний, которые будут принимать значения от 0 до n. У каждой программы будет свой счётчик состояния, которым она фактически будет сообщать, какую часть назначенной ей работы она уже исполнила.

Будем полагать, что установкой значения **j** в переменную **s0**, представляющую счётчик состояния 0-го ядра, программа **A**, исполняемая на 0-ом ядре, будет сообщать, что она уже исполнила все предыдущие действия и готова приступить к выполнению действия **Aj**. Теперь ей необходимо дождаться момента, когда программа **B** 1-го ядра сообщит, в свою очередь, о готовности к исполнению действия **Bj**, установив в переменной **s1** своего счётчика состояния такое же значение **j**.

Пронумеруем все возможные точки встречи (от 1 до **n**). И выразим описанное правило поведения программ в точке встречи в виде функций с параметром, задающим номер встречи:

```
int s0,s1; // счётчики состояний программ
void Init0(void) {s0=0;}
void Init1(void) {s1=0;}
void Randevu0(int j)// рандеву для 0-го ядра
{ s0=j; while(s1!=j); }
void Randevu1(int j)// рандеву для 1-го ядра
{ s1=j; while(s0!=j); }
```

И покажем схематично, как следует составить программы A и B, чтобы обеспечить им требуемую синхронизацию действий:

```
#define R0(j) Randevu0(j)
#define R1(j) Randevu1(j)
// схема программы A 0-го ядра:
Init0();
 R0(1);A1; R0(2);A2; ... R0(n);An;
// схема программы B 1-го ядра:
Init1();
 R1(1);B1; R1(2);B2; ... R1(n);Bn;
```

Заметим, что во всех этих операциях исполнение программы задерживается в цикле ожидания. Но это непроизводительное ожидание вполне допустимо для подобного рода программ. Ведь в такой ситуации нет необходимости переключать программу на исполнение какой-либо другой работы или вообще передавать занимаемый ею процессор какой-то ещё другой программе. И это как раз характеризует особенность рассмотренных средств синхронизации.

3. Схема организации встречи для нескольких программ

Представленные здесь простые приёмы синхронизации позволяют согласовать совместную работу двух параллельных программ, которые исполнятся на разных процессорных ядрах, но имеют возможность взаимодействовать через общую память. Напрашивается вопрос: а можно ли такие приёмы применить для организации взаимодействия более двух программ?

Прежде, чем ответить на него, заметим, что реализация этих приёмов синхронизации, в основном, опирается на важнейший принцип: изменять значение каждой совместно используемой общей переменной имеет право только одна программа. Только при соблюдении такого принципа можно ухитриться применить описанные здесь приёмы для взаимодействия нескольких программ.

Покажем схематично, как можно было бы обеспечить организацию встречи (рандеву) для более двух программ. Пусть количество таких программ будет **m** (**m**>2), и каждая из них исполняется на одном из **m** микропроцессорных ядер, имеющих доступ к общей памяти.

Для взаимодействия параллельных программ в точках встречи будем использовать общий массив переменных s[m], представляющих счётчики состояний этих программ. Будем полагать, что k-ый элемент этого массива s[k] характеризует состояние программы на k-ом ядре. И только программа, исполняемая на k-ом ядре, имеет право изменять его значение. Остальные программы могут лишь читать это значение.

Представим процедуру рандеву для обеспечения такой «многосторонней» встречи в **j**-ой точке, которая будет вызываться **k**-ом ядром:

```
int s[m]; // счётчики состояний программ
void Init(int k) {s[k]=0;}// инициализация
//рандеву для программы k-го ядра в j-ой т.встречи
void Randevu(int k, int j)
{ int i,f;
  s[k]=j; // отметимся, что пришли на встречу
// дождёмся, когда все придут на j-ую точку встречи
  do{ f=1; i=0;
      while(f&&(i<m)){f=(s[i]==j);i++;}
  }while(f==0);//f=1,если все пришли на встречу
}//Randevu
```

И покажем схематично, как должна быть построена программа, запускаемая на **k**-ом ядре для выполнения последовательности действий **P1**, **P2**, ... **Pn** параллельно с аналогичными действиями программ других ядер:

```
#define Rk(j) Randevu(k,j)
// схема программы P k-го ядра:
Init(k);Rk(1);P1;Rk(2);P2;...Rk(n);Pn;
```

4. Синхронизация программ с учётом аварийного завершения

Все представленные выше приёмы синхронизации страдают серьёзным недостатком: в них не предусмотрены действия на случай возникновения чрезвычайных ситуаций. Если одна из параллельных программ не сможет по какой-либо причине продолжить работу, то взаимодействующая с ней другая программа «зависнет» в одном из циклов, в которых она ожидает отклика от неё, а значит, не сможет даже корректно завершить своё функционирование.

Попробуем исправить приведённые ранее алгоритмы синхронизации, чтобы избавиться от этого недостатка. Продемонстрируем, какие для этого следует внести поправки на примере организации точек встречи для синхронизации двух программ (см. п. 2.3).

Будем полагать, что выполняя каждое действие из последовательности A1, A2, … An, программа A 0-го ядра регулярно проверяет, не случилась ли какая-либо чрезвычайная ситуация, возникновение которой препятствует её дальнейшему нормальному функционированию. И в случае возникновения такой ситуации программа A прекращает свою деятельность, выполняя завершающий блок действий Ax, но перед своим завершением сообщает об этом другой программе так, чтобы она могла тоже отреагировать на возникшую ситуацию.

Известно, что когда подобная ситуация возникает при взаимодействии управляющей программы (драйвера) с периферийным устройством, то устройство сообщает о своём сбое установкой особого флажка ошибки (бита в регистре состояния). Позаимствуем этот приём и будем устанавливать флажок аварийного завершения программы A. A от программы B потребуем, чтобы она прекращала свой цикл ожидания встречи, если этот флажок взведён.

Аналогичный флажок будем устанавливать и при аварийном завершении программы B, а программа A соответственно должна будет прекращать цикл ожидания встречи, если он будет взведён. Выразим эти действия с флажками, скорректировав соответствующие алгоритмы:

```
int Abort0,Abort1;//флажки авар.завершения
int s0,s1; // счётчики состояний программ
void Init0(void){ s0=0; Abort0=0; }
void Init1(void){ s1=0; Abort1=0; }
void Randevu0(int j){ s0=j;
 while(s1!=j) if(Abort1) break; }
void Randevu1(int j){ s1=j;
 while(s0!=j) if(Abort0) break; }
```

Предусмотрим также проверку на необходимость экстренного завершения программы на случай, когда рандеву прекращено без получения ожидаемого отклика от партнёра:

```
#define R0(j) { Randevu0(j);\
 if Abort1 { Abort0=1; Ax; exit(1); }
#define R1(j) { Randevu1(j);\
 if Abort0 { Abort1=1; Bx; exit(1); }
// схема блока действий Aj! программы 0-го ядра:
{ Aj;if(ошибка){Abort0=1;Ax;exit(1);}
// схема программы A 0-го ядра: Init0();
 R0(1);A1!; R0(2);A2!; ... R0(n);An!;
// схема блока действий Bj! программы 1-го ядра:
{ Bj;if(ошибка){Abort1=1;Bx;exit(1);}
// схема программы B 1-го ядра: Init1();
 R1(1);B1!; R1(2);B2!; ... R1(n);Bn!;
```

Теперь в случае возникновения каких-либо аномальных ситуаций каждая из программ корректно завершит работу, выполнив свой заключительный блок действий и сообщив об этом другой программе.

5. Примеры взаимодействия двух параллельных программ

Описанные приёмы синхронизации двух параллельных программ были применены при разработке некоторых программ совместного тестирования периферийных устройств, которые предназначались для запуска на двух разных ядрах микропроцессорной системы. Продемонстрируем их применение на примерах двух тестовых программ, проверяющих правильность функционирования контроллеров Ethernet и DMA в многоядерной микропроцессорной системе MIPS-архитектуры.

5.1. Тест контроллера Ethernet на двух микропроцессорных ядрах

Тестовая программа состоит из двух процедур, каждая из которых запускается на отдельном ядре. Одна процедура ведёт себя как отправитель пакетов, а другая – как получатель пакетов, прогоняемых по внутренней петле одного и того же Ethernet-контроллера.

Процедура-отправитель подготавливает пакеты данных для передачи, создаёт кольцо дескрипторов передатчика, запускает передатчик в работу, после чего входит в цикл передачи пакетов. Процедура-получатель очищает (заполняет пустыми данными) места памяти, куда предполагается заносить принимаемые пакеты данных, создаёт кольцо дескрипторов приёмника, запускает работу приёмника контроллера и затем исполняет цикл приёма пакетов.

В самом начале процедура-получатель инициализирует работу контроллера. Каждая из процедур после отправки или приёма очередного пакета проверяет, не было ли зафиксировано каких-либо ошибок при их передаче. А также отслеживает временной интервал (тай-

маут), отведённый для передачи, а при его истечении сигнализирует об ошибке. После приёма каждого пакета процедура-получатель дополнительно проверяет правильность его приёма-передачи, поэлементно сравнивая массивы принятых данных с теми, которые были отправлены.

Описанные действия исполняются этими процедурами параллельно на двух разных процессорах: процедура-получатель запускается на 0-ом ядре, а процедура-отправитель – на 1-ом ядре. Но выполняться они должны согласованно, т.е. в определённом порядке относительно друг друга, и поэтому процессы их выполнения на разных ядрах требуется определённым образом синхронизовать.

Для этого в этих процедурах предусмотрены так называемые точки встречи (рандеву). Дойдя до очередной точки встречи, процедура одного ядра должна дождаться момента, когда процедура другого ядра тоже подойдёт к соответствующей точке встречи, после чего каждая из процедур может продолжать свои дальнейшие действия на своём ядре, пока не дойдёт до конца или до следующей точки встречи.

В каждой процедуре предусмотрено две точки встречи в начале до цикла приёма/передачи пакетов и по две точки встречи внутри тела этого цикла, а также одна точка встречи после выхода из такого цикла перед завершением процедуры. Расположение этих точек поясним с помощью схематичного представления этих процедур, отразив в них лишь порядок исполнения основных действий.

Схема процедуры-получателя (на 0-ом ядре):

```
----- Процедура-ПОЛУЧАТЕЛЬ (на 0-ом ядре) -----
начало_процедуры
|     Init0();
|  – инициализация Ethernet-контроллера
|     Randevu0(1);
|  – подготовка места памяти для пакетов 0,..,P-1
|  – формирование кольца дескрипторов приёмника
|    с заполнением дескрипторов для пакетов 0,..,P-1
|     Randevu0(2);
|  – запуск приёмника контроллера
| ЦИКЛ_приёма_пакетов n= P,..,Q-1
| |– подготовка места памяти для n-го пакета
| |– ожидание поступления очередного пакета
| |   Randevu0(2*n);
| |– проверка правильности приёма пакета
| |   если (ошибка) то {Abort0=1;break;}
| |   Randevu0(2*n+1);
| |   если (Abort1) то {Abort0=1;break;}
| |– заполнение дескрипторов приёма n-го пакета
| конец_цикла_приёма_пакетов
|– ожидание приёма всех уже отправленных пакетов
|– проверка правильности их приёма-передачи
|– останов приёмника контроллера
|       Randevu0(2*Q);
конец_процедуры
```

Схема процедуры-отправителя, запускаемой на 1-м ядре:

```
----- Процедура-ОТПРАВИТЕЛЬ (на 1-ом ядре) -----
начало_процедуры
|     Init1();
|     Randevu1(1);
|  – заполнение данными пакетов 0,..,P-1
|  – построение кольца дескрипторов передатчика
|    с заполнением дескрипторов для пакетов 0,..,P-1
|     Randevu1(2);
|  – запуск передатчика контроллера
| ЦИКЛ_передачи_пакетов n= P,..,Q-1
| |– подготовка (заполнение данными) n-го пакета
| |– ожидание конца передачи очередного пакета
| |   Randevu1(2*n);
| |– проверка правильности передачи пакета
| |   если (ошибка) то {Abort1=1;break;}
| |   Randevu1(2*n+1);
| |   если (Abort0) то {Abort1=1;break;}
| |– заполнение дескрипторов передачи n-го пакета
| конец_цикла_передачи_пакетов
|– ожидание передачи всех отправленных пакетов
|– останов передатчика контроллера
|       Randevu1(2*Q);
конец_процедуры
```

В этих схемах процедур используются параметры **P** и **Q**, которые определяются в программе теста как define-переменные:

```
#define P  14 /* кол-во пакетов в кольце, 1<P<Q */
#define Q  32768 /* кол-во пакетов для передачи */
```

Они задают общее количество передаваемых в тесте пакетов (**Q**), а также размер колец (**P**) дескрипторов, формируемых для приёма-передачи пакетов.

5.2. Одновременное тестирование контроллеров DMA и Ethernet

Чтобы поверить работоспособность контроллеров Internet и DMA при их совместной работе с памятью, было предложено запускать программы их тестирования одновременно на разных ядрах. И согласовать их работу так, чтобы пересылка очередного блока пакетов данных контроллером Internet с одного участка основной памяти в другой (по внутренней петле) совпадала по времени с DMA-передачей по основной памяти блока данных примерно того же размера.

Для реализации такого тестирования в программах тестов контроллера Ethernet и контроллера DMA были организованы точки встречи для обеспечения их синхронизации с использованием приёмов, рассмотренных ранее в п. 2.3 и 4. Расположение таких точек встречи (рандеву) поясняется с помощью схематичного представления алгоритмов этих программ.

Порядок действий программы теста Ethernet-контроллера, запускаемой на 0-ом ядре, проиллюстрируем следующей схемой:

```
 ----- Программа теста Ethernet-контроллера -----
начало_программы
|     Init0();
|     Randevu0(1);
| - инициализация Ethernet-контроллера
|     Randevu0(2);
| ЦИКЛ_приёма-передачи_блока_пакетов r= 0,..,R-1
| |- подготовка данных r-го блока пакетов
| |- очистка памяти для приёма r-го блока пакетов
| |- формирование кольца дескрипторов для
|    передачи P пакетов r-го блока
| |- формирование кольца дескрипторов для
|    приёма P пакетов r-го блока
| |    Randevu0(3+r);
| |    если (Abort1) то {Abort0=1;break;}
| |- запуск приёма-передачи блока пакетов
| |- ожидание поступления всех пакетов блока
| |- останов приёма-передачи блока пакетов
| |- проверка корректности передачи блока пакетов
| |    если (ошибка) то {Abort0=1;break;}
| конец_цикла_приёма-передачи_блоков_пакетов
|       Randevu0(3+R);
|- поэлементная проверка правильности передачи
|   всех переданных и принятых блоков пакетов
|       Randevu0(4+R);
конец_программы
```

Данная программа после инициализации своего контроллера входит в цикл передачи R блоков, состоящих каждый из P пакетов. В начале каждого шага цикла она приготавливает данные для передачи P пакетов и формирует для них кольца дескрипторов приёмника и передатчика. И после очередного рандеву, дождавшись от программы-партнёра готовности к DMA-передаче, запускает передачу всех приготовленных P пакетов.

Далее она ожидает завершения этой передачи (или её прекращения из-за возникшей критической ошибки или истечения таймаута). После чего проверяет, прошла ли передача всех P пакетов блока без ошибок. И в случае обнаружения ошибки прекращает цикл передачи блоков пакетов и сообщает об этом программе-партнёру установкой своего флажка аварийного завершения.

Аналогично построена и программа теста контроллера DMA. После инициализации контроллера и прохождения совместного рандеву она исполняет цикл для DMA-передачи R блоков данных. В начале каждого шага цикла она подготавливает данные для передачи, настраивает регистры и дескрипторы всех задействованных DMA-каналов. После чего выходит на точку встречи для синхронизации своих действий с программой другого ядра.

После проведения рандеву запускает подготовленную DMA-передачу, затем ожидаёт её завершения и проверяет, прошла ли она без ошибок. В случае возникновения ошибки прекращает все последующие DMA-передачи и сигнализирует об этом программе-партнёру.

Описанный порядок действий программы теста DMA-контроллера, запускаемой на 1-ом ядре, проиллюстрируем следующей схемой:

```
 ----- Программа теста DMA-контроллера -----
начало_программы
|     Init1();
|     Randevu1(1);
| - инициализация DMA-контроллера
|     Randevu1(2);
| ЦИКЛ_передачи_блока_данных r= 0,..,R-1
| |- подготовка данных r-го блока
| |- очистка памяти для приёма r-го блока данных
| |- формирование DMA-дескрипторов для
| |- передачи данных r-го блока
| |    Randevu1(3+r);
| |    если (Abort0) то {Abort1=0;break;}
| |- запуск DMA-передачи блока данных
| |- ожидание завершения DMA-передачи
| |- останов DMA-передачи
| |- проверка корректности передачи блока данных
| |    если (ошибка) то {Abort1=1;break;}
| конец_цикла_DMA-передачи_блоков_данных
|       Randevu1(3+R);
|- поэлементная проверка правильности передачи
|   всех переданных DMA блоков данных
|       Randevu1(4+R);
конец_программы
```

В представленных схемах программ тестирования контроллеров Ethernet и DMA используются параметры **P** и **R**, которые определяются в этих программах как define-переменные:

```
#define P 14 /* кол-во пакетов в кольце, 1<P<Q */
#define R 1000 /* кол-во блоков для передачи */
```

Они задают общее количество блоков пакетов (**R**), передаваемых в ходе тестирования по Ethernet и DMA-каналам, а также размер колец дескрипторов, формируемых для приёма-передачи контроллером Ethernet одного блока из **P** пакетов.

6. Заключение

В статье представлены простые способы синхронизации двух параллельных программ, которые были успешно применены для разработки тестов, запускаемых одновременно на двух разных ядрах микропроцессорной системы MIPS-архитектуры для проверки совместной работы контроллеров Ethernet и DMA.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0003 «Методы разработки аппаратно-программных платформ на основе защищенных и устойчивых к сбоям систем на кристалле и сопроцессоров искусственного интеллекта и обработки сигналов».

About Some Simple Techniques to Synchronize Parallel Programs

A.A. Burtsev

Abstract. The article is devoted to the description of simple methods of synchronizing two parallel programs executed on different computing cores of the same computer installation that have access to common memory. The described methods of interaction of programs can be provided on the basis of conventional means available in almost every programming language. And their implementation does not require the use of any special processor instructions or calls to special functions of the operating system.

Keywords: multi-core microprocessors, methods of synchronization and interaction of parallel programs, rendezvous.

Литература

1. В.В. Корнеев, А.В. Киселёв. Современные микропроцессоры. М.: НОЛИДЖ, 2000.
2. А.В. Гордеев, А.Ю. Молчанов. Системное программное обеспечение. СПб: Питер, 2002. с. 221-300.
3. М. Митчел, Д. Оулдем, А. Самьюэл. Программирование для Linux. Профессиональный подход. М.: Вильямс, 2003. с. 95-120.
4. А.А. Бурцев. Параллельное программирование. Учебное пособие по курсу «Операционные системы». Обнинск, ИАТЭ, 1994.
5. Википедия. Test-and-set, https://ru.wikipedia.org/wiki/Test-and-set
6. Википедия. Load-link/store-conditional, https://en.wikipedia.org/wiki/Load-link/store-conditional

Ускорение быстрого преобразования Фурье для многомерных массивов комплексных векторов на основе технологии OpenCL

А.А. Бурцев[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, burtsev@niisi.msk.ru

Аннотация. Статья посвящена применению технологии OpenCL, позволяющей использовать мощные ресурсы графических процессоров для повышения быстродействия вычислительных программ. Рассматриваются варианты разработки в среде OpenCL эффективных параллельных программ, позволяющих ускорить операции быстрого преобразования Фурье для многомерных массивов комплексных векторов.

Ключевые слова: параллельное программирование, технология OpenCL, гетерогенные системы, операция Быстрого Преобразования Фурье (БПФ).

1. Введение

В настоящее время на современных компьютерных установках наряду с широко известными технологиями OpenMP [1, п. 5.1] и MPI [1, п. 5.2] для разработки параллельных программ можно применять также и технологию OpenCL [2], позволяющую использовать мощные ресурсы графических процессоров для повышения быстродействия вычислительных программ. Благодаря этой технологии сегодня можно ускорять вычисления даже на одном компьютере, если имеющаяся в его составе видеокарта поддерживает технологию OpenCL.

В серии статей [3-6] автором было предложено знакомство с технологией OpenCL с демонстрацией приёмов разработки программ для решения ряда вычислительных задач. В частности, рассматривались варианты построения по технологии OpenCL программ, ускоряющих перемножение больших матриц [4], вычисление интегралов [6], а также программ, ускоряющих в среде OpenCL выполнение операции быстрого преобразования Фурье (БПФ) для комплексных векторов большой длины [3,5]. Полученный опыт разработки подобных OpenCL-программ позволяет утверждать, что с помощью технологии OpenCL действительно можно существенно ускорить выполнение такого рода задач.

Но чтобы добиться требуемого ускорения, необходимо уметь распараллеливать используемую вычислительную программу, т.е. перестраивать её, разбивая на такие части, которые могли бы исполняться одновременно разными обработчиками, призванными совместно решать вычислительную задачу. И чем лучше удастся такую программу распараллелить, тем в большей степени получится ускорить её выполнение в среде OpenCL.

Программы обработки массивов данных, в которых над каждым отдельным элементом требуется совершать однотипные вычислительные действия независимо от значений других элементов, безусловно, можно отнести к классу тех вычислительных программ, которые хорошо поддаются распараллеливанию. Поэтому можно ожидать, что выполнение одних и тех же вычислительных операций параллельно над каждым элементом большого множества позволит в результате существенно ускорить решение всей вычислительной задачи в целом.

Во многих задачах цифровой обработки сигналов операцию дискретного преобразования Фурье требуется применять не просто к одному одиночному вектору комплексных чисел, а сразу к множеству комплексных векторов, уложенных в многомерный массив. Операцию Фурье, производимую над многими векторами, будем образно называть множественной операцией Фурье. Заметим, что такую множественную операцию можно зачислить в обозначенный класс задач, легко поддающихся распараллеливанию. А значит, можно ожидать, что и эту операцию над массивами удастся значительно ускорить, применяя технологию OpenCL.

Представим сначала алгоритмы и формулы, по которым осуществляется операция множественного Фурье в обычных программах, рассчитанных на последовательное их исполнение одним процессором. А затем рассмотрим разнообразные варианты программ, позволяющих ускорить исполнение такой операции в среде OpenCL. И проанализируем показатели их производительности, сравнивая результаты их прогонов для одних и тех же наборов данных.

2. Базовая программа для БПФ над комплексными векторами многомерного массива

Операция дискретного преобразования Фурье (ДПФ) над одиночным вектором комплексных чисел X_N заключается в получении результирующего вектора Y_N комплексных чисел, элементы которого Y_k вычисляются на основе исходного вектора X по правилу:

$$Y_k = \sum_{n=0}^{N-1} \{X_l \times W_N^{k \cdot n}\}$$

Здесь используются так называемые весовые коэффициенты вида W^q_N, которые вычисляются как комплексные величины по формуле:

$$W_N^q = e^{-i \cdot 2\pi \cdot q/N},$$

которую можно выразить иначе:

$$W_N^q = \cos\frac{2\pi q}{N} - i \cdot \sin\frac{2\pi q}{N},$$

применяя формулу Эйлера:

$$e^{i \cdot \varphi} = \cos\varphi + i \cdot \sin\varphi,$$

где i – мнимая комплексная единица.

Вычисление одиночной ДПФ непосредственно по этому правилу приводит к алгоритму сложности $O(N^2)$, который требует слишком длительного времени исполнения и потому редко применяется на практике, особенно для преобразований комплексных векторов большого размера. Существует, однако, особый класс алгоритмов под общим названием «Быстрое Преобразование Фурье» (БПФ), которые характеризуются вычислительной сложностью $O(N\log_2 N)$ и позволяют выполнить ДПФ гораздо быстрее.

Один из таких алгоритмов для исполнения операции Фурье над одиночным вектором был подробно описан в [3, п.2.2] и представлен там в виде функции **FFT** (**F**ast **F**ourier **T**ransform) на языке Си с таким заголовком:

```
void FFT(int N, complex *Y,
 complex *X, complex *W)
```

Такая функция совершает операцию БПФ по заданному комплексному вектору X длины N, получая в качестве результата комплексный вектор Y той же длины. Далее будем обозначать такую операцию в виде:

$$Y = FFT^N(X)$$

Предполагается, что весовые коэффициенты, используемые этой функцией для исполнения операции БПФ, вычисляются заранее и передаются ей в виде в виде таблицы – массива комплексных чисел W (размером от 0 до N).

Используя эту функцию, выразим операцию множественного БПФ для двумерных массивов (матриц) комплексных векторов:

$$Y_{M \times J} = MFFT^N(X_{M \times J})$$

с помощью Си-функции **MFFT**:

```
void MFFT(int M, int J, int N,
complex *X,complex *Y,complex *W){
int m,j,mj;
for (m=0; m<M; m++)
 for (j=0;j<J;j++){ mj=(m*J+j)*N;
  FFT(N,&Y[mj],&X[mj],W);
 }//for j,m
}//MFFT
```

В ней в теле внутреннего цикла (по j) над векторами X[m,j] и Y[m,j], являющимися элементами двумерных массивов X и Y, осуществляется одиночная операция БПФ:

$$Y_{m,j} = FFT^N\left(X_{m,j}\right)$$

Путём перебора циклов по m и j функция MFFT в результате исполнит операции БПФ над всеми парами векторов заданных матриц.

Такой последовательный алгоритм выполнения операции множественного БПФ примем в качестве базового. Далее будем рассматривать различные варианты исполнения той же операции множественного БПФ в среде параллельных исполнителей. А также будем сравнивать каждый из них (по производительности) с этим базовым, чтобы выявить, какой вариант даёт максимальное ускорение в среде OpenCL.

3. Первоначальный вариант (A) OpenCL-программы для множественной операции БПФ

Ранее (см. [3, п.3]) был предложен вариант OpenCL-программы, позволяющий ускорить операцию БПФ для одиночного вектора комплексных чисел. Приняв его за основу, попытаемся модифицировать его OpenCL-программу так, чтобы она осуществляла операцию БПФ для всех пар векторов заданных матриц X и Y, т.е. в итоге стала бы исполнять множественную операцию БПФ над матрицами векторов.

3.1. Простая модификация основной OpenCL-программы

Сначала попробуем просто переделать основную программу, оставив прежними её процедуры ядра. Осуществим такую переделку тем же естественным приёмом, как и в случае с последовательной программой. Для выполнения множественной операции БПФ в среде OpenCL составим отдельную функцию (см. далее **clMFFT**). В ней разместим два вложенных цикла с перебором по m и по j. А в теле внутреннего цикла (по j) для осуществления одиночной операции БПФ

с очередной парой векторов X[m,j] и Y[m,j] поставим вызов функции **clFFT**, которая была подробно описана ранее (она была представлена в п.3.2 в [3]).

```
void clMFFT(int M,int J,int N,
complex *Y,complex *X,complex *W){
int m,j,mj;
for (m=0; m<M; m++) {
 for (j=0;j<J;j++){ mj=(m*J+j)*N;
   clFFT(N,&Y[mj],&X[mj]);
 }//for j,m
}//clMFFT
```

Заметим, что перед вызовом функции **clFFT** в переменной **mj** по формуле **mj=(m*J+j)*N** предварительно вычисляется индекс, начиная с которого в многомерных массивах X[M][J][N] и Y[M][J][N] располагаются соответственно вектора X[m,j] и Y[m,j]. Именно для этих комплексных векторов, как элементов матриц X[M][J] и Y[M][J], на очередном шаге внутреннего цикла и вызывается функция **clFFT**, чтобы исполнить одиночную операцию БПФ. В последующих алгоритмах будем неоднократно использовать такую формулу вычисления значения mj для определения индекса месторасположения вектора в многомерном массиве.

Чтобы оценить, насколько удаётся ускорить операцию множественного БПФ с помощью такой переделанной OpenCL-программы, сравним время исполнения ($\mathbf{T_{CL}}$) функции **clMFFT** (в среде OpenCL с множеством параллельных исполнителей) со временем исполнения ($\mathbf{T_{CPU}}$) функции **MFFT** (на одном процессоре) с теми же параметрами и вычислим коэффициент ускорения **K** как отношение $\mathbf{T_{CPU}/T_{CL}}$.

Такая программа с вызовами этих функций (варианта A1) была разработана (на языке Си) как консольное приложение в MS Visual Studio. Для представления комплексного числа парой вещественных значений одинарной точности в Си-программе тип complex определялся так:

```
#define complex cl_float2
```

А в программе, содержащей процедуры ядра на языке OpenCL, – немного иначе:

```
#define complex float2
```

Программа была скомпонована в MS Visual Studio под ОС Windows-7 для исполнения на OpenCL-платформе, содержащей процессор Intel i3-2100 (3.1 Ггц) и специализированную видеокарту NVidia GeForce 1050ti (1392 Мгц).

На этой платформе (будем называть её платформа «NVidia») скомпонованная OpenCL-программа многократно прогонялась для матриц комплексных векторов различных размеров (M×J) и длин (N). Усреднённые показатели коэффициентов ускорения, полученные в результате таких прогонов, представлены в таблице 1.

Таблица 1. Ускорение множественного БПФ OpenCL-программой варианта A1 на платформе NVidia

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.023	0.024	0.0175	0.020	0.014	0.0032	0.0012
64	0.046	0.051	0.0475	0.0325	0.019	0.0079	0.0036
128	0.110	0.1166	0.1033	0.0815	0.0812	0.0085	0.0034
256	0.254	0.2240	0.2247	0.2316	0.0879	0.0181	0.0091
512	0.444	0.4332	0.4163	0.2780	0.1030	0.0348	---
1024	0.898	1.0071	0.9260	0.6293	0.2368	0.0535	---
2048	2.165	1.9904	2.0805	1.5418	0.4658	0.0864	---
4096	4.449	3.7327	3.7107	2.9872	0.6236	---	---
8192	7.461	6.2371	5.6467	4.5101	0.8492	---	---
16384	12.79	12.214	9.1541	9.7322	---	---	---
32768	26.40	21.397	22.617	21.582	---	---	---
65536	**42.35**	40.909	36.044	36.285	---	---	---

Замечание: символы --- означают, что для таких значений параметров программа не может быть корректно исполнена (например, из-за недостатка памяти)

Сравнивая показатели ускорения операции множественной БПФ (*MFFT*) из таблицы 1 с аналогичными показателями ускорения одиночной операции БПФ (*FFT*), представленными в таблице 2 в [3], можно заметить, что они примерно одинаковы для одной и той же длины векторов. Причём с ростом M и J итоговые коэффициенты ускорения мало изменяются.

3.2. Модификация процедур ядра

Теперь попробуем исходную OpenCL-программу, рассмотренную ранее (в п.3 в [3]) для одиночной операции БПФ, переделать немного по-другому. Внедрим циклы по m и j для перебора векторов не в основную программу, а в её процедуры OpenCL-ядра.

Вставим такие циклы в процедуру ядра, чтобы осуществлять бит-реверсное копирование векторов из матрицы X в матрицу Y:

```
__kernel void fft_brvprstMJ
( uint N,const uint L,
 __global complex *X,
 __global const complex *Y,
 __global const uint *BRT,
 const uint M, const uint J){
uint m,j,mj,k,i; i=get_global_id(0);
k=BRT[i]; // k=бит-реверсное значение для i
for (m=0; m<M; m++)
for (j=0; j<J; j++)
{ mj=(m*J+j)*N; Y[mj+k]=X[mj+i]; }
}// fft_brvprstMJ
```

В результате вызова такой процедуры исполнителем под номером **i** в среде OpenCL элементы с индексом **i** всех векторов, содержащихся в матрице **X**, будут скопированы в соответствующие им векторы матрицы **Y** на позицию с индексом **k**, вычисленную как бит-реверсное значение для индекса **i**.

Аналогичную модификацию сделаем и для другой процедуры ядра, которая вызывается каждым исполнителем OpenCL-среды на очередной стадии группового исполнения параллельных операций «бабочек Фурье» (БФ) над векторами (см. 5-й пункт в списке действий, описанных в п.3.2 в [3]). В результате получим такую процедуру ядра:

```
__kernel void fft_btflyMJ
(int t, int N,__global complex *Y,
__global complex *W,uint M,uint J)
{uint G,R,k,f,s,h,a,b,u,m,j,mj;
 uint i= get_global_id(0);
 G=1<<(t-1); // кол-во групп G= 2^(t-1)
 R=N>>t; // кол-во пар в группе R= 2^(P-t);
 k=i&(G-1); // номер группы = i % G
 f=i>>(t-1);// номер пары в группе= i / G
 s=G; h=2*s;
 a=k+f*h; b=a+s; u=R*k;
 complex V= W[u];
 for (m=0; m<M; m++)
 for (j=0; j<J; j++) {
   mj=(m*J+j)*N;
   BTF(Y[mj+a],Y[mj+b],V);
 }// for j,m
}//fft_btflyMJ
```

Конечно же, и в основную OpenCL-программу тоже придётся внести добавления, чтобы эти процедуры ядра могли запускаться уже с расширенным набором параметров:

```
kn1=clCreateKernel(prgrm,"fft_brvprstMJ",0);
kn2=clCreateKernel(prgrm,"fft_btflyMJ",0);
clSetKernelArg(kn1,5,szI,&M);
clSetKernelArg(kn1,6,szI,&J);
clSetKernelArg(kn2,4,szI,&M);
clSetKernelArg(kn2,5,szI,&J);
```

Коэффициенты ускорения, которые обеспечивает полученный в результате такой переделки вариант (A2) OpenCL-программы, представим в таблице 2.

Таблица 2. Ускорение множественного БПФ OpenCL-программой варианта A2 на платформе NVidia

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.091	0.600	1.667	2.667	6.25	5.10	5.25
64	0.111	0.857	4.00	6.333	8.714	8.571	9.238
128	0.364	2.00	7.00	11.750	15.556	17.625	17.308
256	0.923	3.875	12.60	34.333	29.273	31.950	31.152
512	1.438	9.650	17.875	42.999	59.150	66.073	63.240
1024	2.826	17.083	46.089	72.489	88.283	101.18	102.16
2048	6.088	33.075	79.185	109.39	128.61	134.75	---
4096	12.89	51.199	115.08	150.09	166.12	157.44	---
8192	30.88	79.759	138.40	159.42	165.28	---	---
16384	52.80	128.08	175.53	149.61	166.70	---	---
32768	80.28	136.18	166.71	182.04	---	---	---
65536	106.9	172.85	**187.84**	184.12	---	---	---

Сравнивая его с предыдущим вариантом (A1), можно заметить, что показатели ускорения заметно подросли. Чем же можно объяснить такой их рост?

Дело в том, что в процедурах ядра индексы (i,k) элементов для бит-реверсной перестановки и индексы (a,b,u) элементов, участвующих в операциях БФ («бабочки Фурье»), теперь вычисляются всего лишь один раз. А затем они многократно применяются сразу для обработки элементов (с такими индексами) у всех векторов, перебираемых далее в циклах (по m и по j). Вот поэтому процедуры ядра и стали выполняться немного быстрее.

Заметим также, что таблице 2 (в отличие от таблицы 1 предыдущего варианта) уже становится заметно, что с ростом размеров (M и J) матриц векторов коэффициенты ускорения тоже подрастают. Правда, увы, не в такой же степени, как сами эти размеры.

Чтобы добиться ещё больших показателей ускорения, попробуем перестроить OpenCL-программу так, чтобы при прогоне её в среде OpenCL использовать как можно больше параллельных исполнителей.

4. Альтернативный вариант (B) OpenCL-программы для множественной операции БПФ

Ранее уже неоднократно демонстрировалось (например, в [4]), что любую программу, которая над каждым элементом массива осуществляет вычислительные действия, независимые от значения других элементов, можно легко распараллелить с тем, чтобы ускорить её исполнение в среде OpenCL. Для этого надо лишь составить OpenCL-программу так, чтобы вычислительная обработка каждого элемента массива выполнялась отдельным исполнителем OpenCL-среды, функционирующим параллельно с другими.

Следуя этому правилу, составим OpenCL-программу выполнения операции множественного Фурье $Y_{M\times J}=MFFT(X_{M\times J})$ для двумерных массивов $X_{M\times J}$ и $Y_{M\times J}$ комплексных векторов. Будем задействовать ансамбль из M×J параллельных исполнителей OpenCL-среды, сгруппированных в двумерный массив [M][J], назначив каждому из них в качестве задания исполнить операцию $Y_{m,j}=FFT(X_{m,j})$ над выделенной ему парой векторов (с индексами m,j).

Для исполнения одиночной операции БПФ над парой векторов оформим функцию **cl_FFT**, заимствовав при её оформлении алгоритм функции FFT, подробно описанный в п. 2.2 в [3], и подкорректировав её заголовок с учётом особенностей языка OpenCL:

```
void cl_FFT ( uint N,
```

```
 __global complex *VY,
 __global complex *VX,
 const __global complex *VW) {
uint P,t,s,h,G,R,d,k,u,j,a,b,i;
complex W; complex XP[NP];
P=iLog2(N);       // so that N = 2^P
// ч1. Бит-реверсное копирование XP <=VX
 copy_brvprst(N, P, VX, XP);
s=1;h=2;G=1; R=N/2; d=N/2;
//ч2.Основной цикл исполнения бабочек Фурье:
for (t=1; t<=P; t++) {
 for (k=0,u=0; k<G; k++,u=u+d) {
  W= VW[u];
  for (j=0,a=k; j<R; j++,a=a+h)
  { b=a+s; BTF(XP[a],XP[b],W); }
 }//for k
 h=h*2;s=s*2; G=G*2; R=R/2; d=d/2;
}//for t
// ч3. Копирование XP => VY
 for (i=0; i<N; i++) VY[i]= XP[i];
}//cl_FFT
```

В этой функции для улучшения быстродействия вектор VX из глобальной памяти сначала копируется при бит-реверсной перестановке в вектор XP, размещаемый в приватной памяти исполнителя. Далее все операции БФ («бабочек Фурье») с помощью макрооперации **BTF**, подробно описанной в п.3.1 в [3], исполняются над вектором XP, который затем записывается как результат в вектор VY глобальной памяти.

Используя функцию **cl_FFT** в качестве вспомогательной, составим теперь процедуру OpenCL-ядра, которая будет запускаться на каждом из параллельных исполнителей OpenCL-среды, собранных в двумерный массив [M][J]:

```
__kernel void cl_kern_MFFT
( uint M, uint J, uint N,
 __global complex *Y,
 const __global complex *X,
 const __global complex *W ) {
 int m, j, mj;
 m= get_global_id(0);
 j= get_global_id(1);
 mj=(m*J+j)*N;
// БПФ для вектора Y[m,j] <= FFT(X[m,j])
 cl_FFT(N, &Y[mj], &X[mj], W);
}//cl_kern_MFFT
```

В этой процедуре с помощью применения функции get_global_id каждый исполнитель сам определяет значения индексов **m** и **j**, указывающие месторасположение в матрицах X и Y той пары векторов, для которых ему предстоит выполнить операцию БПФ.

Для запуска такой процедуры ядра в OpenCL-среде в основную OpenCL-программу добавим следующие действия.

1. Процедуре ядра cl_kern_MFFT назначим дескриптор **knM**:

```
cl_kernel knM=
 clCreateKernel(prgrm,"cl_kern_MFFT",NULL);
```

2. Выделим в памяти OpenCL-устройства буферы для данных, участвующих в операции:

```
cl_mem objX,objY;
cl_uint szC= sizeof(complex);
ObjX=clCreateBuffer(cntxt,
 CL_MEM_READ_ONLY,szC*M*J*N,0,0);
ObjY=clCreateBuffer(cntxt,
CL_MEM_READ_WRITE,szC*M*J*N,0,0);
```

3. Загрузим матрицу комплексных векторов X[M][J][N] в буфер объекта **objX**:

```
cl_uint szC= sizeof(complex);
clEnqueueWriteBuffer(cmndQ,objX
   CL_TRUE,0,szC*N*M*J,X,0,0,0);
```

4. Назначим процедуре ядра cl_kern_MFFT 6 фактических параметров:

```
cl_uint szI= sizeof(cl_uint);
cl_uint szM= sizeof(cl_mem);
clSetKernelArg(knM,0,szI,&M);
clSetKernelArg(knM,1,szI,&J);
clSetKernelArg(knM,2,szI,&N);
clSetKernelArg(knM,3,szM,&objY);
clSetKernelArg(knM,4,szM,&objX);
clSetKernelArg(knM,5,szM,&objW);
```

5. Запустим в OpenCL-среде процедуру ядра cl_kern_MFFT на множестве из M×J исполнителей и дождёмся её завершения всеми исполнителями:

```
size_t gWS[2]= {M,J};
cl_event evM;
clEnqueueNDRangeKernel(cmndQ,
 knM,2,NULL,gWS,NULL,0,NULL,&evM);
clFinish(cmndQ);
```

6. Полученную в буфере объекта **objY** матрицу векторов запишем в Y[M][J][N]:

```
clEnqueueReadBuffer(cmndQ,objY,
CL_TRUE,0,szC*N*M*J,Y,0,0,0);
```

Представим теперь, какие коэффициенты ускорения обеспечивает полученный таким способом вариант (B) OpenCL-программы (см. таблицу 3).

Сравнивая их с показателями предыдущих вариантов A1 и A2 (в таблицах 1 и 2), можно заметить, что при малой длине обрабатываемых векторов (N<1024), вариант B обеспечивает лучшие коэффициенты ускорения для больших размеров матриц (M×J>=20×20). Но при больших значениях N (N≥1024) вариант B, увы, уступает варианту A2 почти на всех размерах матриц M×J (и больших, и малых).

Таблица 3. Ускорение множественного БПФ OpenCL-программой варианта B на платформе NVidia

M×J: N	2×2	5×5	10 ×10	20 ×20	50 ×50	100 ×100	200 ×200
32	0.40	1.50	5.00	25.00	49.73	51.00	82.00
64	0.40	2.00	5.50	19.00	59.00	61.67	48.50
128	0.67	4.67	14.00	22.00	72.50	74.75	63.57
256	0.91	5.33	21.00	35.67	66.84	78.88	71.79
512	1.53	7.89	23.50	45.60	78.22	82.76	78.18
1024	1.79	7.00	22.29	50.71	86.78	87.78	82.44
2048	1.90	7.91	24.03	58.42	91.50	96.72	---
4096	1.61	8.75	28.15	64.95	98.73	107.80	---
8192	2.03	11.69	36.28	81.03	125.21	---	---
16384	2.08	11.61	37.60	88.17	**131.26**	---	---
32768	2.24	12.44	40.37	92.67	---	---	
65536	---	---	---	---	---	---	

Получается, что OpenCL-программы вариантов A1,A2 проявляют свои достоинства с ростом длины векторов N, а OpenCL-программа варианта B – с ростом размеров матриц M×J. А нельзя ли совместить оба этих достоинства в одной OpenCL-программе?

5. Комбинированный вариант (C) OpenCL-программы для множественной операции БПФ

Создадим теперь такую OpenCL-программу для множественной операции БПФ, которая задействует N×M×J параллельных исполнителей OpenCL-среды. И распределяет вычисления между ними так, что операцией БПФ для каждой пары векторов ($X_{m,j}$,$Y_{m,j}$) совместно занимаются N исполнителей (как в варианте A1), но при этом вычисления БПФ для всех пар векторов матриц $X_{M\times J}$ и $Y_{M\times J}$ осуществляются не последовательно, а параллельно (одновременно).

5.1. Модификация процедур ядра

Возьмём процедуры ядра, рассмотренные в п. 3.2 (варианта A2), и заменим в их теле циклы по m и по j (выделенные жирным шрифтом) на одиночные действия с парой элементов обрабатываемых векторов. А для вычисления месторасположения **mj** тех векторов в матрицах X и Y, которые назначены для обработки конкретному исполнителю, определим значения индексов для **m** и **j** с помощью функции get_global_id.

В результате заменим обозначенные циклы в процедуре ядра fft_brvprstMJ на фрагмент:

```
m= get_global_id(1);
j= get_global_id(2);
{ mj=(m*J+j)*N; Y[mj+k]=X[mj+i]; }
```

А в процедуре cl_fft_btflyMJ – на фрагмент:

```
m= get_global_id(1);
j= get_global_id(2);
mj=(m*J+j)*N;
BTF(Y[mj+a],Y[mj+b],V);
```

5.2. Модификация основной OpenCL-программы варианта C

Для запуска таких процедур ядра в OpenCL-среде в основной OpenCL-программе необходимо изменить прежнюю последовательность действий, описанных в п.3.2 в [3] и заимствованных для вариантов A1 и A2.

1. Во-первых, внесём добавления для определения дескрипторов kn1 и kn2 новых процедур ядра и назначения для них новых параметров, которые были сделаны в разделе 3.2.

2. Во-вторых, добавим фрагменты для выделения памяти объектам objX, objY, а также загрузки матрицы X в память OpenCL-устройства (в буфер objX), описанные в пунктах 2,3 списка действий в разделе 4.

3. Запустим в OpenCL-среде процедуру ядра fft_brvprstMJ на множестве из N×M×J исполнителей, сгруппированных в трёхмерный массив [N][M][J], для бит-реверсного параллельного копирования всех M×J векторов из буфера objX, представляющего матрицу $X_{M\times J}$, в соответствующие им вектора буфера objY, представляющего матрицу $Y_{M\times J}$. И дождёмся окончания её завершения всеми исполнителями:

```
size_t gWS[3]= { N,M,J };
cl_event ev1;
clEnqueueNDRangeKernel(cmndQ,
 kn1,3,NULL,gWS,NULL,0,NULL,&ev1);
clFinish(cmndQ);
```

4. Теперь поставим основной цикл (по t), в котором на каждом шаге (t) будем запускать процедуру ядра cl_fft_btflyMJ на множестве из (N/2)×M×J исполнителей для совершения всех операций БФ t-го этапа (t=1,..,P) параллельно для всех N/2 пар элементов всех M×J векторов, содержащихся в буфере матрицы Y:

```
cl_uint t; gWS[0]=N/2;
cl_event ev2;
for (t=1; t<=P; t++) {
 clSetKernelArg(kn2,0,szI,&t);
 clEnqueueNDRangeKernel(cmndQ,
 kn2,3,NULL,gWS,NULL,0,NULL,&ev2);
 clWaitForEvents(1,&ev2);
}//for t
```

В начале каждого шага цикла требуется назначить 0-му параметру процедуры ядра значение номера (t) текущего этапа, а перед его окончанием следует дождаться завершения этой процедуры ядра всеми исполнителями с помощью вызова функции clWaitForEvents.

5. Наконец, полученную в буфере objY матрицу векторов нужно скопировать из памяти OpenCL-устройства в итоговую матрицу Y, для чего добавим в основную программу фрагмент, описанный в п. 6 списка действий в разделе 4.

Представим теперь, какие коэффициенты ускорения обеспечивает полученный в результате проведённой модификации новый вариант (C) OpenCL-программы (см. таблицу 4).

Таблица 4. Ускорение множественного БПФ OpenCL-программой варианта C на платформе NVidia

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.061	0.375	1.667	14.50	36.00	26.07	124.40
64	0.312	0.680	5.567	10.067	45.15	82.72	76.45
128	0.241	3.025	8.840	34.10	66.82	112.60	94.80
256	0.778	3.281	20.36	71.30	81.60	119.45	121.77
512	1.498	11.267	29.415	56.087	129.45	120.34	119.58
1024	4.578	20.417	47.680	93.176	116.61	140.66	133.59
2048	8.276	33.489	89.667	120.26	151.15	160.42	---
4096	15.32	59.788	115.75	147.54	164.95	162.56	---
8192	27.81	86.733	143.77	172.49	184.86	---	---
16384	41.17	115.56	170.44	195.11	176.31	---	---
32768	67.48	147.76	171.20	209.55	---	---	---
65536	117.0	179.57	**222.73**	207.10	---	---	---

Из приведённой таблицы хорошо видно, что при таком варианте действительно удаётся совместить достоинства рассмотренных ранее вариантов A1, A2 и B. Во-первых, анализируя показатели в каждой строке этой таблицы, можно заметить, что при любой фиксированной длине (N) коэффициент ускорения значительно увеличивается с ростом размеров матриц (M×J). И во-вторых, анализируя показатели каждого столбца таблицы, можно отметить и другую закономерность: для матриц любого фиксированного размера (M×J) коэффициент ускорения существенно вырастает при росте длины векторов (N). И это позволяет утверждать, что в этом комбинированном варианте удаётся совместить достоинства всех прежних вариантов.

Сравнивая показатели коэффициентов ускорения таблицы 4 с аналогичными показателями из приведённых ранее таблиц 1-3, можно сделать вывод, что такой комбинированный вариант (C) обеспечивает наилучшие результаты ускорения множественной операции Фурье $Y_{M\times J}=MFFT^{N}(X_{M\times J})$ почти для всех параметров: длинах векторов N и размеров матриц M×J.

А максимального значения (**222.73**) коэффициент ускорения этой операции (на платформе «NVidia») достигает (как это и отмечено в таблице 4) при значениях параметров: N=65536 (2^{16}) и M×J=10×10. Заметим при этом, что для одиночной операции FFT^{N} на векторах той же длины ($N=2^{16}$) при запуске программы в OpenCL-среде можно добиться ускорения лишь в **45.67** раза, применив первоначальный вариант OpenCL-программы, представленной в [3], и только в **69** раз, если применить улучшенный вариант этой же OpenCL-программы, описанный в [5]. (Эти показатели приведены соответственно в таблице 2 в [3] и таблице 10 в [5]).

6. Множественные операции БПФ для платформы «Intel»

Рассмотренные варианты программ, разработанные для ускорения множественной операции БПФ (МБПФ) в среде OpenCL, были опробованы также и на других OpenCL-платформах. В частности, они компоновались на обычном настольном компьютере (в MS Visual Studio под ОС Windows-10) и запускались на аппаратной платформе, оснащённой процессором CPU Intel-i59400 (2.9 Ггц) и встроенным графическим процессором GPU UHD 630 (350 МГц).

На этой платформе (будем называть её далее платформа «Intel») скомпонованные OpenCL-программы всех рассмотренных вариантов (A1, A2, B, C) многократно прогонялись для матриц векторов комплексных чисел одинарной точности с теми же параметрами, что и для платформы «NVidia». Коэффициенты ускорения, полученные в результате произведённых прогонов с матрицами таких же размеров (M×J) и такими же длинами векторов (N), представлены соответственно в таблицах 5-8.

Таблица 5. Ускорение множественного БПФ OpenCL-программой варианта A1 на платформе Intel

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.012	0.011	0.011	0.012	0.012	0.012	0.012
64	0.024	0.024	0.021	0.026	0.024	0.025	0.025
128	0.048	0.045	0.049	0.049	0.047	0.032	0.049
256	0.096	0.093	0.094	0.097	0.097	0.096	0.112
512	0.196	0.197	0.188	0.192	0.195	0.193	0.195
1024	0.376	0.385	0.389	0.378	0.390	0.402	0.402
2048	0.766	0.768	0.795	0.784	0.763	0.745	---
4096	1.505	1.576	1.564	1.568	1.402	1.624	---
8192	3.116	3.296	2.658	2.995	2.954	---	---
16384	5.864	5.893	6.063	5.502	6.566	---	---
32768	11.02	10.797	10.994	11.307	---	---	---
65536	19.92	20.507	20.275	**21.023**	---	---	---

Таблица 6. Ускорение множественного БПФ OpenCL-программой варианта A2 на платформе Intel

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.045	0.260	0.836	1.785	2.439	2.548	1.987
64	0.094	0.519	1.638	3.350	4.508	4.223	3.980
128	0.184	0.954	3.239	5.888	9.838	7.771	7.575
256	0.379	2.276	6.149	12.707	14.681	14.563	13.625
512	0.763	4.088	11.470	23.291	25.159	24.262	22.689
1024	1.485	8.004	22.506	36.549	33.480	32.379	---
2048	3.138	15.043	37.352	44.731	33.296	29.169	---
4096	5.745	25.840	43.223	38.666	32.489	---	---
8192	10.74	37.401	44.659	31.583	27.576	---	---
16384	18.51	40.257	30.305	27.803	---	---	---
32768	29.29	40.174	26.196	26.154	---	---	---
65536	**41.23**	32.990	28.406	---	---	---	---

Таблица 7. Ускорение множественного БПФ OpenCL-программой варианта B на платформе Intel

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.175	1.006	3.867	12.976	15.648	12.423	11.666
64	0.273	1.713	6.069	17.946	14.600	8.963	9.770
128	0.368	2.012	8.131	**21.097**	8.490	6.131	7.176
256	0.460	2.487	10.206	18.855	6.061	5.127	5.018
512	0.526	2.900	9.506	15.522	5.099	4.565	3.793
1024	0.580	3.126	10.544	11.370	4.424	4.197	---
2048	0.605	3.004	10.053	9.557	4.319	4.006	---
4096	0.617	3.067	8.567	7.896	4.092	---	---
8192	0.650	2.927	8.399	6.741	3.929	---	---
16384	0.660	2.828	6.567	5.883	---	---	---
32768	0.635	2.613	5.382	5.753	---	---	---
65536	0.637	2.218	6.285	---	---	---	---

Таблица 8. Ускорение множественной операции БПФ OpenCL-программой варианта C на платформе Intel

M×J: N	**2×2**	**5×5**	**10 ×10**	**20 ×20**	**50 ×50**	**100 ×100**	**200 ×200**
32	0.046	0.292	1.084	4.321	18.408	29.340	22.365
64	0.093	0.595	2.302	7.844	29.483	29.211	21.656
128	0.187	1.190	4.067	13.439	31.170	29.595	27.637
256	0.384	2.237	7.895	22.462	36.091	28.672	29.091
512	0.757	4.706	13.512	33.059	34.963	30.628	28.058
1024	1.496	8.116	23.629	40.344	32.374	32.791	---
2048	3.010	14.688	34.601	42.086	35.342	34.897	---
4096	5.731	25.160	42.610	37.504	37.322	---	---
8192	10.77	37.797	46.239	38.721	38.265	---	---
16384	21.92	**47.783**	41.826	40.246	---	---	---
32768	31.03	43.424	39.520	37.231	---	---	---
65536	42.23	41.645	41.793	---	---	---	---

Сравнивая показатели ускорения, записанные в этих таблицах, можно сделать вывод, что и на платформе «Intel» наилучшие коэффициенты ускорения множественной операции БПФ почти для всех параметров обеспечивает комбинированный вариант C (см. таблицу 8).

В строках и столбцах таблицы результатов, полученных для этого варианта на OpenCL-платформе «Intel», также можно увидеть соблюдение отмеченных ранее закономерностей. При фиксированном значении длины векторов N коэффициент ускорения заметно подрастает при увеличении размеров матриц M×J. А для фиксированных размеров матриц M×J коэффициент ускорения вырастает с ростом длины векторов N.

Правда, эти закономерности перестают соблюдаться для самых крайних показателей каждой строки и каждого столбца, где параметры N и M×J достигают предельно высоких значений. При таких больших значениях параметров OpenCL-программа либо вообще не может корректно вычислить требуемый результат из-за нехватки памяти (тогда вместо коэффициента в таблице проставляется знак ---), либо её вычисления в OpenCL-среде оказываются недостаточно эффективными по причине реального отсутствия столь необходимого количества вычислительных ядер на используемой OpenCL-платформе.

Как видно из приведённой таблицы 5, для множественной операции БПФ на платформе «Intel» удаётся достичь максимального значения коэффициента ускорения **47.783** при значениях параметров: N=16384 (2^{14}) и M×J=5×5.

Заметим, что на этой же платформе ранее для одиночной операции БПФ той же длины N=2^{14} были получены такие значения коэффициентов ускорения: **5.333** для первоначального варианта OpenCL-программы, представленного в [3]; и **11.38** для оптимизированного варианта OpenCL-программы, описанного в [5]. (Эти значения приведены соответственно в таблице 1 в [3] и в таблице 9 в [5]).

7. Заключение

Таким образом, технологию OpenCL можно успешно применять для повышения быстродействия таких вычислительных программ, в которых требуется выполнять однотипные операции над многочисленными объектами, сгруппированными в массивы данных.

Но для этого требуется уметь распараллеливать вычислительную задачу так, чтобы каждый элемент огромного массива объектов данных мог обрабатываться отдельным исполнителем OpenCL-среды независимо от других. В таком случае можно добиться значительного ускорения выполнения такой задачи в OpenCL-среде.

Это наглядно демонстрируют представленные примеры OpenCL-программ, позволяющих существенно ускорить выполнение множественной операции быстрого преобразования Фурье для многомерных массивов комплексных векторов большого размера. Следует отметить также, что в среде OpenCL для множественной операции Фурье над массивом векторов удаётся добиться ускорения в значительно большей степени, чем для такой же операции Фурье над одиночным вектором.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0003 «Методы разработки аппаратно-программных платформ на основе защищенных и устойчивых к сбоям систем на кристалле и сопроцессоров искусственного интеллекта и обработки сигналов»

.

Acceleration of Fast Fourier Transform for Multidimensional Arrays of Complex Vectors Based on OpenCL Technology

A.A. Burtsev

Abstract. The article is devoted to the use of OpenCL technology, which allows using the powerful resources of graphic processors to increase the speed of computing programs. Development variants of efficient parallel programs in the OpenCL environment to accelerate the fast Fourier transform operation for multidimensional arrays of complex vectors are considered.

Keywords: parallel programming, OpenCL technology, heterogeneous systems, operation of Fast Fourier Transform (FFT)

Литература

1. В.В. Воеводин, Вл.В. Воеводин. Параллельные вычисления. Спб., БХВ-Петербург, 2004.
2. Официальный OpenCL–сайт организации Khronos Group, http://www.khronos.org/opencl/
3. А.А. Бурцев. Ускорение быстрого преобразования Фурье на основе технологии OpenCL. «Труды НИИСИ РАН», Т. 11 (2021), № 4, 27–37.
4. А.А. Бурцев. Оптимизация операции перемножения матриц на основе технологии OpenCL. «Труды НИИСИ РАН», Т. 10 (2020), № 5-6, 100–112.
5. А.А. Бурцев. Оптимизация операции быстрого преобразования Фурье в среде OpenCL. // «Труды НИИСИ РАН», Т.12 (2022), №1-2, 11-27.
6. А.А. Бурцев. Применение технологии OpenCL для ускорения вычисления интегралов. // «Труды НИИСИ РАН», Т.13 (2023), №1-2, 19-24.

Вопросы применения технологий дополненной реальности в пропедевтических курсах по программированию

Д. И. Кадина[1], А. Г. Кушниренко[2], К. А. Мащенко[3], М. С. Паремузов[4], Н. А. Серебрицкая[5], Е. Д. Тарасюк[6]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kadinadaria@mail.ru;
[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, agk_@mail.ru;
[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, МГУ им. М. В. Ломоносова, Москва, Россия, Государственный университет управления, Москва, Россия, kirill.mashchenko@vip.niisi.ru;
[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, matveyparem@gmail.com;
[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, serebr@vip.niisi.ru;
[6]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, ekaterina.tarasuk@math.msu.ru;

Аннотация. Статья посвящена методам демонстрации процесса выполнения программ управляющих роботами для детей дошкольного возраста. Представлены методики использования реальных и виртуальных роботов, а также технологии дополненной реальности (AR) для создания интерактивных обучающих материалов. Показаны примеры использования AR в учебной бестекстовой среде программирования ПиктоМир и особенности 3D-моделирования для реализации этих технологий. Массовое применение этих методик требуют выполнения ресурсоемких алгоритмов, которые еще несколько лет назад были посильны только мощным серверам, а сегодня могут без задержек выполняться на массово производимых планшетах экономкласса. Результаты исследований демонстрируют значительное улучшение восприятия программирования детьми через игровую форму обучения и использование современных технологий. Использование дополненной реальности при выполнении составленной ребенком программы движения робота по игровому полю позволяет избавить ребенка от «экранной» работы, жестко регулируемое федеральными Санитарными правилами и нормами.

Ключевые слова: дошкольник, младшеклассник, алгоритмика, робот, бестекстовый, безэкранный, ПиктоМир, дополненная реальность

1. Введение

Идея использовать реальных и виртуальных роботов при обучении новичков программированию принадлежит известному математику, программисту, психологу и педагогу Сеймуру Пейперту, ученику Жана Пиаже. Его знаменитая "Черепашка", управляемая внешними командами, была вдохновлена роботами известного нейрофизиолога и кибернетика Грея Уолтера, созданными в 1949 году.

Черепашка Пейперта вначале появилась в двух формах: реальная черепашка-робот, обитающая на полу, управляемая компьютером по проводам и рисующая картинки реальными чернилами на разостланной на полу реальной бумаге и экранная черепашка, живущая и рисующая картинки на экране дисплея. Реальная черепашка быстро проиграла соревнование с виртуальной. Автор статьи [1] Александр Ильинский остроумно заметил «Широкое распространение персональных компьютеров в конце 70-х годов привело к массовому вымиранию черепашек-роботов и неконтролируемому размножению экранных черепашек.» Оригинальная экранная «черепашка» Пейперта была им реализована в среде программирования на новом, предназначенном для детей, языке программирования Logo и получила широкое распространение, которое, однако, со временем выявило некоторые проблемы. Ученик Пейперта Резник Митчел, один из создателей языка программирования Scratch, описал эти проблемы следующим образом [2]: «В 1980-х годах тысячи школ обучили миллионы учащихся программировать на Logo, но первоначальный энтузиазм быстро угас. Учителям и учащимся было трудно преодолеть нелогичности в синтаксисе этого языка. К тому же знакомство с ним осуществлялось в контексте не слишком интересных задач.»

В конце 20 века на смену Logo пришли новые языки программирования для начального обучения детей, и вернулись реальные роботы, ранее покинувшие образовательную практику. В первой четверти 21 века многие ученые и педагоги придерживаются мнения, что оптимальный путь освоения азов программирования детьми включает использование как реальных, так и виртуальных роботов. С использованием современных массово производимых планшетов, технологий искусственного интеллекта и дополненной реальности и подходящих учебных пособий, к моменту достижения детьми возраста 7-8 лет

возможно систематическое освоение всех конструкций структурного программирования и элементарных приемов составления и отладки программ в курсе объемом около семидесяти получасовых занятий.

2. Демонстрация ребенку процесса выполнения программы, управляющей роботом

В мире создано несколько популярных учебных бестекстовых и безэкранных сред программирования и робототехнических наборов, рассчитанных на детей 5–8 лет [3]. Безэкранное программирование позволяет значительно уменьшить время экранной работы ребенка, регулируемое федеральными Санитарными правилами и нормами СанПиН 1.2.3685-21 (далее *СанПиН*).

Возраст до 5 лет. Поскольку действующие СанПиН полностью запрещают использование электронных средств обучения детьми возраста до 5 лет, демонстрация процесса выполнения распознанной компьютером программы возможна только в реальном мире. По нашей методике дети собирают на столе или на полу игровой комнаты игровое поле, компьютер в соответствии с распознанной им программой, подает размещенному на поле роботу звуковые команды, легко воспринимаемые и понимаемые детьми. А робот, получая слышимые детьми команды, перемещается по игровому полю, проводя действия с расположенными на поле объектами (рис. 1).

Рис. 1. Использование реального робота, передвигающегося по коврикам, в учебном процессе

Возраст от 5 до 6 лет. Согласно действующим СанПиН, индивидуальная работа на планшетах запрещена, но возможно коллективное использование электронной доски. В этом случае, на доску может быть выведена либо виртуальная обстановка, полностью сгенерированная компьютером, либо изображение реальной остановки, собранной детьми на полу игровой комнаты, дополненное изображениями виртуальных роботов, сгенерированных компьютером в процессе выполнения программы (см. подробнее в разделе 3).

Возраст старше 6 лет. Согласно действующим СанПиН, в этом возрасте возможна индивидуальная работа на планшетах ограниченной продолжительности. Поэтому ребенок, не нарушая правил, может наблюдать на экране выполнение составленной им программы, составленной либо на экране, либо в материальном мире. В возрасте старше 6 лет разрешено использование сенсорного экрана планшета для а) составления игровой обстановки, б) составления программы и в) наблюдения за ходом выполнения программы. Однако представляется более эффективным перенесение части этих этапов работ в реальный мир: программы ребенок составляет из материальных объектов, игровую остановку также собирает из материальных объектов, размещая на игровом поле изготовленные на 3D принтере или склеенные из картона модельки роботов, и лишь собственно процесс выполнения программы ребенок наблюдает на экране в дополненной реальности. При этом ребенок видит на экране не только виртуальные объекты, но и предметы реального мира, поэтому эта демонстрация воспринимается детьми, как происходящая скорее в реальном, чем в виртуальном мире.

3. Технологии дополненной реальности в цифровой образовательной среде ПиктоМир

Одной из наиболее интересных и перспективных современных технологий является дополненная реальность (AR), которая позволяет создавать виртуальные объекты и размещать их в реальном мире. В последние годы AR получила все большую популярность в различных областях, включая образование.

Дополненная реальность имеет давнюю историю развития. Впервые термин "дополненная реальность" был использован в 1992 году Томасом К. Крамером, который предложил концепцию добавления виртуальных объектов к реальной сцене. Однако, только с развитием современных технологий и компьютеров AR стала доступной для широкого использования.

Сегодня дополненная реальность нашла свое применение в образовании. AR позволяет созда-

вать интерактивные учебные материалы, которые могут значительно улучшить процесс обучения. Например, студенты могут использовать AR для изучения анатомии, истории, географии и других предметов. Также AR позволяет создавать виртуальные экскурсии и лаборатории, что существенно расширяет возможности образовательного процесса.

Не следует путать методики дополненной реальности (AR) с методиками виртуальной реальности (VR). Для работы в виртуальной реальности нужны специальные очки или шлем, надевая которые ты оказываешься в полностью другой, симулированной реальности. Использование подобных очков или шлемов VR может вызывать значительный дискомфорт, особенно у детей, поэтому многие производители подобного оборудования не рекомендуют его использование детьми младше 13 лет.

А вот для применения технологии AR нужен всего лишь обычный планшет, на котором, поверх транслируемой картинки с камеры, появляются созданные объекты. У детей при этом складывается впечатление, что эти объекты находятся рядом с ними, они могут обойти с планшетом вокруг них и рассмотреть их со всех сторон, а появляются эти объекты в привычной им обстановке реального мира, например, на полу игровой комнаты детского сада.

Технология дополненной реальности сложнее технологии виртуальной реальности и требует сложных математически ресурсоемких алгоритмов и мобильных устройств, снабженных дополнительными датчиками (акселерометр, гироскоп, компас и др.). К счастью, сегодня, в конце первой четверти 21 века, прогресс в ИКТ привел к тому, что технологии дополненной реальности реализуемы практически на любых массово производимых мобильных компьютерных устройствах. Более того, имеется свободно распространяемое программное обеспечение, которым можно воспользоваться для реализации методик дополненной реальности на мобильных устройствах.

Математическая составляющая использования дополненной реальности включает в себя теорию геометрии и алгоритмы распознавания плоскостей. Для того, чтобы виртуальные объекты корректно отображались в реальном мире, необходимо определить положение камеры и распознать плоскость, на которую будет проецироваться виртуальный объект.

Алгоритм распознавания плоскостей основан на использовании камеры с маркером, который помогает определить положение камеры и распознать плоскость. После этого виртуальные объекты могут быть размещены на этой плоскости.

Для того, чтобы обойти объект в AR со всех сторон, используются датчики, такие как акселерометр, гироскоп и компас. Они помогают определить положение устройства в пространстве и изменить направление взгляда на объект. Также используется технология SLAM (Simultaneous Localization and Mapping), которая позволяет определить положение устройства в реальном времени и создать карту окружающей среды.

Отделом учебной информатики была разработана свободно распространяемая среда бестекстового программирования для дошкольников «ПиктоМир» [4-7]. В ПиктоМире использована широко распространенная библиотека кампании Google «Сервисы Google Play для AR». Для использования приложения на планшетах эконом класса появилась необходимость сделать достаточно гладкие и сложные, но в то же время не слишком большие по весу модели, чтобы при отображении сразу большого количества объектов и роботов из ПиктоМира приложение не начинало «подтормаживать».

3D моделирование бывает разных типов, но в данном случае было использовано полигональное моделирование. Полигональное моделирование является фундаментальным и широко используемым методом 3D моделирования. Оно основано на соединении вершин и ребер для формирования многоугольников – полигонов, что позволяет точно контролировать геометрию (рис. 2).

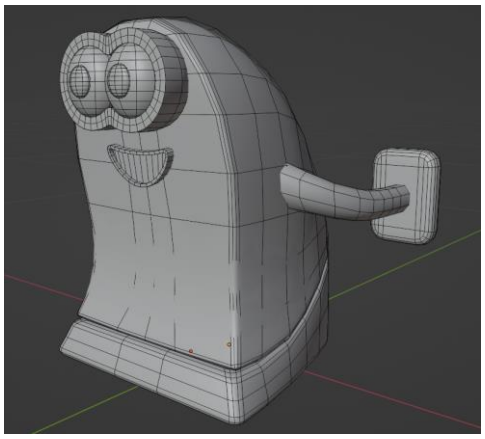

Рис. 2. Визуальное отображение полигонов в 3D модели

Этот метод эффективен при создании как детализированных, так и простых объектов, но у него есть недостаток: на силуэте 3D модели видны ребра полигонов. Эта проблема решается увеличением количества полигонов, либо использованием карты нормалей. Технология карт нормалей плохо поддерживается в AR и VR, поэтому 3D модели были смоделированы со средним количеством полигонов, чтобы сохранить

плавность силуэтов и при этом не увеличивать вес самих моделей в программе, а модель сглажена с помощью манипулирования направлением нормалей вершин.

Чтобы наложить цвет на модель, необходимо создать развертку, или карту UV, и на базе этой карты создать карту цвета. UV преобразование — это процесс 3D моделирования, при котором поверхность 3D модели проецируется на 2D изображение для наложения текстуры. Буквы «U» и «V» обозначают оси 2D текстуры, поскольку «X», «Y» и «Z» уже используются для обозначения осей 3D объекта в пространстве модели (рис. 3).

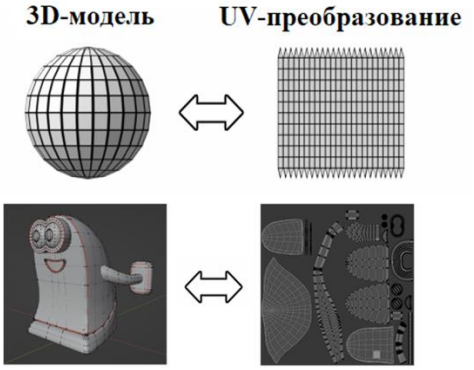

Рис. 3. 3D модель и UV развертка модели

Для подготовки модели к разворачиванию UV карты необходимо группам ребер 3D модели назначить определенный угол сглаживания, так как от этого напрямую зависит качество UV карты в 2D пространстве и качество наложения карты цвета в последующем, а также внешний вид 3D модели.

Чтобы подробнее объяснить работу с группами сглаживания необходимо коснуться темы нормалей. Нормально – это вектор, перпендикулярный поверхности полигона (рис. 4).

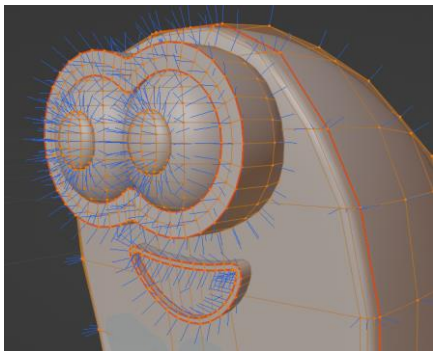

Рис. 4. Визуальное отображение векторов нормалей

В программах 3D моделирования нормали используются для определения ориентации грани полигона (нормаль грани) и того, как ребра граней будут визуально выглядеть по отношению друг к другу при освещении (нормали вершин). Нормали вершин — это вектора, которые указывают направление вершин. Нормали вершин определяют визуальную мягкость или жесткость между полигонами и то, как модель выглядит, когда свет падает на полигон, а управление ими позволяет управлять реакцией полигона на свет. Когда все нормали для определенной вершины направлены в одном направлении (так называемые мягкие или общие нормали вершин), между гранями возникает плавный переход света (рис. 5).

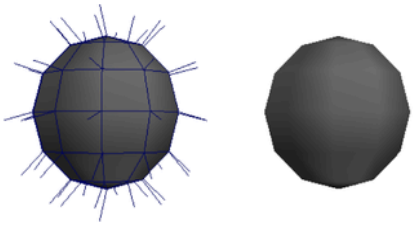

Рис. 5. Мягкие нормали вершин

Когда нормали вершин направлены в том же направлении, что и их грани (так называемые жесткие нормали вершин), переход между гранями становится жестким, что создает граненый вид (рис. 6).

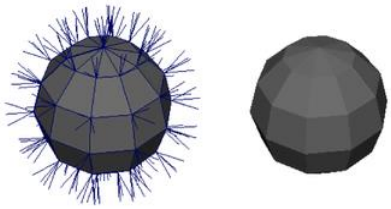

Рис. 6. Жесткие нормали вершин

При работе с UV разверткой необходимо, чтобы на углах меньше или равных 90 градусам, была выставлены жесткие нормали вершин, а на углах более 90 градусов - мягкие нормали. Шов разреза UV развертки обычно проходит именно по жестким нормалям. Если это невозможно, как в случае с моделями роботов, ребрам шва назначают жесткие нормали. В случае моделей роботов ребра некоторых швов остались мягкими нормалями. Это стало возможно, потому что не было возможности в запекании карт нормалей (рис. 7).

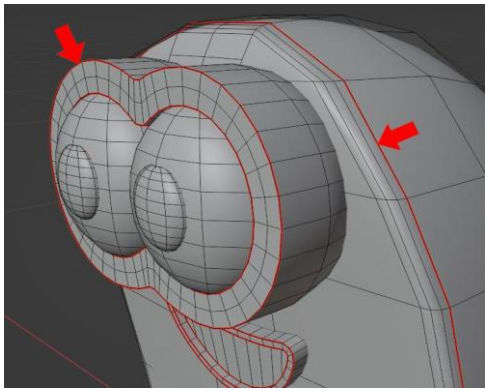

Рис. 7. Швы UV разветки

Далее UV развертка была импортирована в графический редактор, где каждой ее части был присвоен нужный цвет (рис. 8).

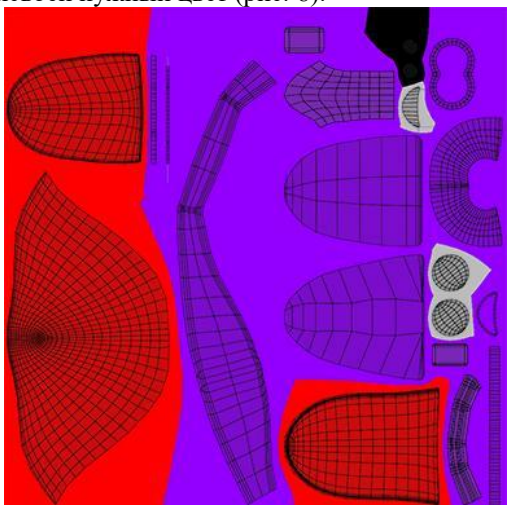

Рис. 8. Карта цвета с наложенной на нее UV картой

Цвета модели экспортируются как отдельное изображение в формате jpeg и при импортировании в программу AR используется как Color map/Diffuse map.

Благодаря всем данным методам в модуль дополненной реальности ПиктоМира были интегрированы, за небольшим исключением, все виртуальные роботы ПиктоМира, и даже при отрисовке на массово производимых планшетах умеренной производительности одновременного движения большого количества роботов на одной сцене не возникает никаких «подтормаживаний». Помимо этого, знакомых детям виртуальных роботов ПиктоМира, в дополненной реальности можно рассмотреть подробно, вблизи и со всех сторон (рис. 9-13).

Рис. 9. Робот Двигун в дополненной реальности.

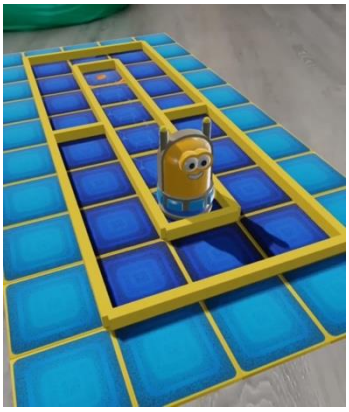

Рис. 10. Робот Вертун в дополненной реальности.

Рис. 11. Робот Ползун в дополненной реальности.

Рис. 12. Совместная работа роботов Двигун и Тягун с ящиками

Рис. 13. Совместная работа роботов на карте с большим количеством объектов.

Для передачи из ПиктоМира информации о процессе выполнения программы и реализации движения сразу большого количества объектов по карте с отслеживанием правильности тактов выполнения программы (например, когда один робот выполняет команду «Мигнуть» или проверяет условие) была использована архитектура сбора «состояний» карты, созданная для Копилки в ПиктоМире. Данная архитектура на каждом такте выполнения программы собирает всю информацию у всех объектов в ПиктоМире и кладет ее в массив, где каждый элемент отвечает за состояние карты на очередной такт выполнения программы. Вся эта информация передается в модуль дополненной реальности, после чего сравниваются состояния каждого объекта на предыдущем такте и на текущем. Если состояния отличаются – то производится движение или изменение внешнего вида объектов. Например, если изменились координаты робота – производится его передвижение в нужную точку, если изменилось его направление – производится его вращение в нужную сторону, если клетка стала закрашена – то меняется ее текстура. Благодаря этому, в модуль дополненной реальности не пришлось встраивать код компиляции программы из ПиктоМира, в него сразу передается результат процесса выполнения программы, который ему просто нужно проиграть.

4. Заключение

Программная поддержка подходов, описанных в настоящей статье, реализована в цифровой образовательной среде ПиктоМир, которая сегодня внедрена в реальный учебный процесс в сотнях детских садах и начальных школ, участвующих в работе, организованной ФГУ ФНЦ НИИСИ РАН сетевой инновационной площадки ПиктоМир.

Работа выполнена в рамках темы государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0001 (1023032100070-3-1.2.1).

Application of Augmented Reality Technologies in Conducting Programming Courses for Preschool Children and Primary School Children

D. I. Kadina, A. G. Kushnirenko, K. A. Mashchenko, M. S. Paremuzov, N. A. Serebritskaia, E. D. Tarasuk

Abstract. The article is dedicated to methods of demonstrating the process of programming robots for preschool children. The paper presents techniques for using both real and virtual robots, as well as augmented reality (AR) technology to create interactive educational materials. Examples of using AR in the educational non-textual programming environment PictoMir and the specifics of 3D modeling to implement these technologies are demonstrated. The widespread application of these methods requires the execution of resource-intensive algorithms, which just a few years ago were only feasible for powerful servers but can now be executed without delay on mass-produced economy-class tablets. The research results demonstrate a significant improvement in children's perception of programming through gaming-based learning and the use of modern technologies. The use of augmented reality in executing a child-composed robot motion program on a gaming field allows the child to avoid "screen" work, which is strictly regulated by federal sanitary rules and norms.

Keywords: preschooler, elementary school student, algorithms, robot, textless, screenless, PictoMir, augmented reality.

Литература

1. Ильинский А. Бунтарь Сеймур Пейперт: от черепашек- роботов к экранным черепашкам // Коммерсантъ. Наука. 19.01.2021. https://www.kommersant.ru/doc/4653334

2. Резник М. Спираль обучения: 4 принципа развития детей и взрослых. М.: Манн, Иванов и Фербер, 2018. 192 с.

3. Кушниренко А. Г., Леонов А. Г., Мащенко К. А., Райко М. В., Грибанова И. Н. Начальное обучение алгоритмике дошкольников и других новичков с помощью умных роботов-игрушек // Труды Научно-исследовательского института системных исследований Российской академии наук. 2023. Т. 13. No 1-2. С. 52–68. EDN: AUYZXT. DOI: 10.25682/ NIISI.2023.1-2.0008.

4. Бесшапошников Н. О., Кушниренко А. Г., Леонов А. Г., Райко М. В., Собакинских О. В. Цифровая образовательная среда «ПиктоМир»: опыт разработки и массового внедрения годового курса программирования для дошкольников // Информатика и образование. 2020. Т. 35. No 10. С. 28–40. EDN: JSAWOV. DOI: 10.32517/0234-0453-2020-35-10-28-40.

5. Бесшапошников Н. О., Леонов А. Г. Пиктограммный язык программирования «Пикто» // Вестник кибернетики. 2017. Т. 28. No 4. С. 173–180. EDN: VVPPQT.

6. Бетелин В. Б., Кушниренко А. Г., Леонов А. Г. Основные понятия программирования в изложении для дошкольников // Информатика и ее приложения. 2020. Т. 14. Вып. 3. С. 56–62. DOI: 10.14357/19922264200308.

7. Леонов А. Г., Райко М. В., Собакинских О. В., Собянина Н. В. Результаты освоения годовой программы «Алгоритмика для дошколят» подготовительными группами муниципального ДОУ // Труды Научно-исследовательского института системных исследований Российской академии наук. 2020. Т. 10. No 5-6. С. 195–199. EDN: LLAERB. DOI: 10.25682/ NIISI.2020.5_6.0023.

Применение методов свободного синтаксиса для распознавания пиктокубиков в курсе «Алгоритмика для дошкольников»

А. Г. Леонов[1], К. А. Мащенко[2], Н. С. Мартынов[3], М. В. Райко[4], А. И. Стрекалова[5], Т. Г. Хан[6].

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, МГУ им. М. В. Ломоносова, Москва, Россия, МПГУ, Москва, Россия, Государственный университет управления, Москва, Россия, dr.l@vip.niisi.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, МГУ им. М. В. Ломоносова, Москва, Россия, Государственный университет управления, Москва, Россия, kirill.mashchenko@vip.niisi.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nikolai.martynov@math.msu.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, rayko@niisi.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, anastasiia.strekalova@math.msu.ru;

[6]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, tatyanakhan@bk.ru.

Аннотация. Статья рассматривает вопросы безопасной интеграции цифровых технологий в образовательные процессы детей старшего дошкольного возраста в соответствии с нормами СанПин. Основное внимание уделяется методике обучения алгоритмике с использованием образовательной среды ПиктоМир, которая включает в себя программное обеспечение и объекты реального мира, такие как радиоуправляемые роботы-игрушки, мягкие игрушки, кубики и магнитные карточки с пиктограммами команд, а также сочленяемые коврики. Описывается поэтапный процесс перехода от материального мира к виртуальному для углубленного освоения понятий. Кроме того, статья представляет подход к решению задачи детектирования объектов с использованием сверточных нейронных сетей, применяемый для проведения алгоритмиад. Учащиеся выполняют задачи, используя физические объекты, такие как пиктокубики, для создания программ, после чего полученные программы переносятся преподавателем в приложение ПиктоМир. Этот метод позволяет улучшить результаты освоения основ программирования у детей старшего и среднего дошкольного возраста.

Ключевые слова: дошкольник, младшеклассник, алгоритмика, робот, бестекстовый, безэкранный, кубики, пиктограммы, ПиктоМир, распознавание пиктокубиков

1. Введение

В условиях стремительной цифровизации образовательных процессов важным пунктом является их безопасная интеграция. Согласно нормам СанПин, детям старшего дошкольного возраста (5-7 лет) допустимо взаимодействовать с цифровыми устройствами 5-7 минут в день. При обучении детей данной возрастной группы преимущественно используются развивающие, безопасные для их здоровья игры.

По методике, разработанной в ФГУ ФНЦ НИИСИ РАН [1], обучение дошкольников алгоритмике проводится с помощью образовательной среды ПиктоМир [2]. Эта среда состоит не только из программной системы ПиктоМир, установленной на компьютерах детей, но и из Учебно-Методического комплекта, включающего объекты реального мира. В комплект входят:

- радиоуправляемые роботы-игрушки,
- мягкие игрушки, изображающие экранных роботов,
- кубики и магнитные карточки с пиктограммами команд для безэкранного составления программ,
- сочленяемые коврики, собирающиеся в игровые поля, по которым движутся роботы

На первых занятиях дети в основном взаимодействуют с материальными объектами (рис. 1). Затем к этим взаимодействиям подключается компьютер, управляющий реальными роботами-игрушками по программам, составленными детьми из привычных им деревянных кубиков. После освоения основных понятий в смешанной компьютерно-материальной среде, дети легко погружаются в виртуально-экранный мир, где действуют цифровые двойники реальных роботов. Такой метод постепенного перехода от материального мира к виртуальному способствует глубокому и прочному освоению важных понятий.

Рис. 1. Дети составляют программу из пиктокубиков и пиктокарточек с командами роботов.

На гранях деревянных кубиков и на магнитных карточках изображены пиктограммы – знаки, обозначающие команды для роботов-исполнителей в цифровой среде ПиктоМир. Решая предложенные задачи, дети собирают программу из пиктограмм.

Процесс переноса программы выполняется педагогом-воспитателем и является рутинным процессом, занимающим достаточно много времени. В связи с этим перед разработчиками ЦОС встала задача интегрировать в приложение модуль компьютерного зрения, который позволит переносить решение из реального мира в виртуальный посредством одной фотографии.

Такого рода проблема в компьютерном зрении относится к задаче детектирования объектов. На текущий момент это с успехом решается за счет использования сверточных нейронных сетей. Эти сети имитируют биологический процесс обработки визуальной информации и способны автоматически выделять характеристики объектов, позволяя компьютеру распознавать их на изображениях. Эффективность сверточных нейронных сетей в области детектирования объектов делает их важным инструментом для различных приложений, таких как автоматическое распознавание лиц, медицинская диагностика и автономные транспортные системы.

2. Технические требования

Сегодня, благодаря разработанной Российской Академией Наук инновационной методике, азы программирования успешно внедрены в 600+ детских садах России. Десятки тысяч дошкольников в возрасте 6-7 лет сейчас с удовольствием и без труда осваивают этот подход [3].

Приложение «ПиктоМир» распространяется на операционные системы Android и iOS и после установки может работать без сети Интернет. Также планшеты и мобильные устройства, используемые в учреждениях, как правило, имеют скромные технические характеристики, что также необходимо учитывать при разработке. В связи с этим к разрабатываемому модулю детектирования пиктограмм предъявляются требования интеграции в приложение без необходимости подключения к сети Интернет и оптимизации под работу на слабых устройствах. Само детектирование объектов должно происходить по фотографии с выложенным из пиктограмм решением. Поскольку решением является комбинация пиктограмм в определенном порядке, результатом детектирования должна быть таблица, где они располагаются в той же последовательности [4].

3. Решение задачи

Сверточные нейронные сети для детектирования объектов [5, 6] архитектурно подразделяются на два вида: одноэтапные и двухэтапные. Одноэтапные архитектуры подразумевают одиночный прогон изображения через сеть, двухэтапные же имеют этап уточнения задетектированных объектов. За счет единичного прогона изображения одноэтапные архитектуры имеют меньшее число слоев, что позволяет им работать быстрее двухэтапных. Одной из одноэтапных архитектур является YOLOv5 (You Only Look Once) [7]. К её преимуществам можно отнести: высокое качество детектирования объектов, высокую скорость работы, наличие нескольких версий, которые зависят от размера сети, а также удобный функционал для трансферного обучения и валидации результатов, реализованный с помощью фреймворка PyTorch [8]. Также одним из решающих факторов является возможность конвертации сети в форматы, поддерживаемые операционными системами Android и iOS (.tflite, .torchscript и .mlmodel). Программная реализация модели, а также скрипты для обучения и конвертации были взяты из официального репозитория YOLOv5 на GitHub [9].

Для обучения модели был собран набор данных на 10 000 изображений, где были размечены 15 классов пиктограмм. Пример размеченного изображения представлен на рисунке 2.

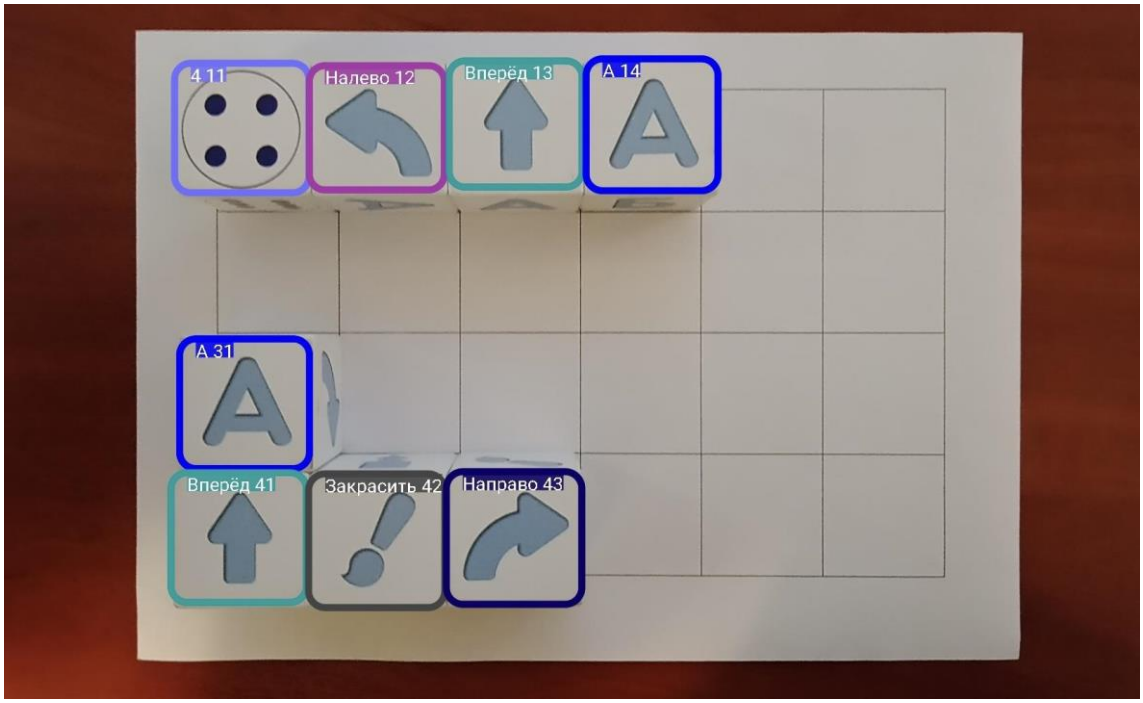

Рис. 2. Пример размеченного изображения.

Набор размеченных данных был поделен на две составляющие: 80% изображений оказались в обучающей выборке, а 20% попали в валидационную выборку.

Важным аспектом обучения являются методы аугментации [10]. Аугментация позволяет расширить обучающий набор данных различными преобразованиями изображений, но выбирать их следует с осторожностью. Так, например, классы “налево” и “направо” при горизонтальном отображении инвертируются, и модель начинает определять их неверно, в связи с чем этот метод был вручную удален из списка аугментации.

Процесс обучения модели, внутри которого 3 раза запускалось дообучение модели на новых данных, длился в общей сложности 100 эпох. На вход подавались изображения размером 416 на 416 пикселей. Усредненные по всему процессу обучения метрики качества модели представлены на рисунке 3. Подробно о данных метриках рассказано в статье [7].

В результате обучения была получена модель, метрики качества распознавания [11] которой проиллюстрированы на рисунке 4.

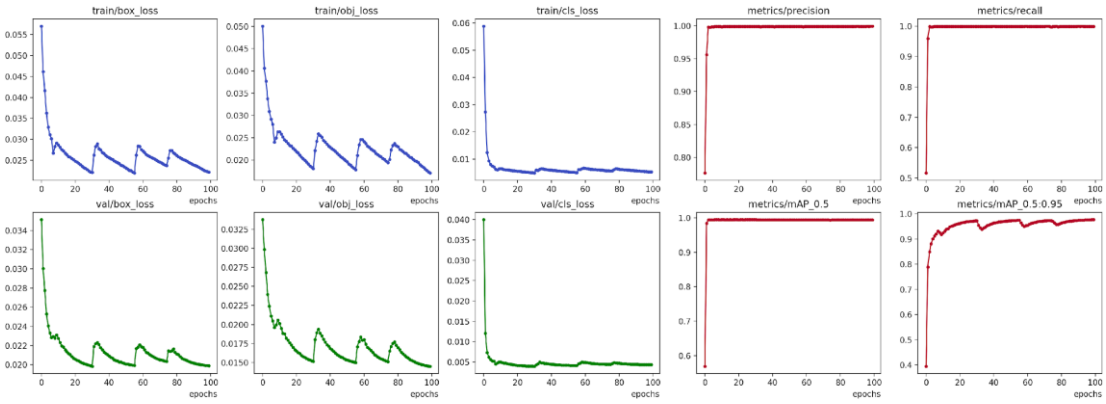

Рис. 3. Метрики, усредненные по процессу обучения модели.

Classes statistics

class	instances	P	R	mAP50	mAP50-95
Алгоритм А	5471	1.0	1.0	0.995	0.975
Алгоритм Б	4287	1.0	1.0	0.995	0.99
Алгоритм В	4435	1.0	1.0	0.995	0.989
Мигнуть	4436	1.0	1.0	0.995	0.993
Вперёд	5325	0.997	0.997	0.995	0.946
Тащить	5618	1.0	1.0	0.995	0.979
Закрасить	4862	0.996	0.998	0.994	0.972
Повторитель 1	4023	0.999	1.0	0.995	0.973
Повторитель 2	3992	0.999	1.0	0.995	0.977
Повторитель 3	4318	0.999	1.0	0.995	0.979
Повторитель 4	3958	1.0	1.0	0.995	0.988
Повторитель 5	4589	1.0	1.0	0.995	0.988
Повторитель 6	4064	0.999	1.0	0.995	0.98
Налево	5534	0.999	1.0	0.995	0.958
Направо	4724	0.999	0.998	0.994	0.966

Рис. 4. Метрики средней точности для каждого класса по результатам обучения

Завершающим шагом является конвертация модели в форматы, поддерживаемые операционными системами Android и iOS. Используя скрипт для экспорта в разные форматы, было получено три модели в форматах tflite, torchscript и mlmodel. В результате тестирования конвертированных моделей обнаружилось, что модель в формате torchscript значительно ухудшилась в качестве распознавания. Это явно иллюстрировалось в предсказании неправильных классов. Однако даже в предсказаниях правильных классов модель показывала низкую “уверенность” в ответе. Модели в формате tflite и mlmodel, напротив, на валидационной выборке показали хорошие результаты и не уступали в качестве распознавания оригинальной модели. Результаты работы моделей tflite на Android и mlmodel на iOS схожи по полученным метрикам.

4. Результаты

В ходе работы для решения задачи детекции элементов пиктограммного языка была дообучена сверточная нейронная сеть YOLOv5. Авторами был собран и размечен набор данных, содержащий 10 000 изображений. По итогу обучения, за 100 эпох целевая метрика mAP:0.5-0.95 на валидационной выборке достигла значения 98%. Для автономной работы модели на устройствах с операционными системами Android и iOS была произведена ее конвертация в форматы tflite, torchscript и mlmodel. В ходе тестирования модель в формате torchscript значительно ухудшила качество распознавания. Модели tflite и mlmodel на валидационной выборке продемонстрировали такой же результат, как и оригинальная модель, однако их размер был в два раза меньше – всего 13 МБ. По итогу работы модели в форматах tflite и mlmodel были интегрированы в приложение ЦОС “ПиктоМир” и прошли апробацию на первых алгоритмиадах.

5. Применение на практике

В рамках усовершенствования образовательного процесса в ФГУ ФНЦ НИИСИ РАН разработали методику проведения олимпиады для дошкольников и учащихся начальной школы – алгоритмиады с использованием пиктокубиков и пиктокарточек.

В рамках алгоритмиады [12] учащиеся кооперативно или поодиночке выполняют олимпиадные задачи на время. Участники старшего дошкольного возраста при решении задачи получают бумажную карту с заданием, фигурку робота и бумажный вариант шаблон программы (таблица для размещения пиктокубиков). Дети проводят фигурки исполнителей по карте и выкладывают соответствующие кубики в нужной последовательности, размещая их в напечатанном шаблоне. На рисунке 5 дети составляют программу из кубиков.

Рис.5. Дети передвигают роботов по карте на столе и составляют программы из пиктокубиков

Полученную программу из кубиков педагог фотографирует, после чего эта программа переносится в приложение «ПиктоМир». (рис. 6)

Преимущества подобной методики «физического» процесса составления программы состоят в следующем:

- Во-первых, здесь не требуется освоение интерфейса системы ПиктоМир для манипуляции пиктограммами команд на экране.
- Во-вторых, согласно СанПиН, период экранной работы ребенка в приложении ограничивается и сводится к короткому просмотру процесса выполнения составленной из кубиков программ.

Нужно отметить, что во время составления маршрута движения робота по карте задания, обсуждения с партнером по команде работы роботов и выкладывания программы на столе из кубиков или карточек, экран компьютера не задействован.

Рис.6. Педагог фотографирует программу, составленную ребенком из пиктокубиков

6. Заключение

Методика была опробована в десятках дошкольных учреждений инновационной площадки «Апробация и внедрение основ алгоритмизации и программирования для дошкольников и младших школьников в цифровой образовательной среде ПиктоМир».

Это позволило опробовать описанную модель распознавания на нескольких десятках устройств с операционными системами Android и iOS.

В результате удачного эксперимента многие педагоги внедрили данный метод переноса программ в цифровую среду ПиктоМир на плановых занятиях, что позволяет улучшить результаты освоения основ дошкольного программирования у детей старшего и среднего дошкольного возраста.

Работа выполнена в рамках темы государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0001 (1023032100070-3-1.2.1)

Application of Free Syntax Methods for Recognizing Piktocubes in the Course «Algorithmics for Preschoolers»

A. G. Leonov, K. A. Mashchenko, N. S. Martynov, M. V. Rayko, A. I. Strekalova, T. G. Khan.

Abstract. The article addresses issues related to the safe integration of digital technologies into the educational processes of older preschool children in accordance with sanitary rules and norms. The main focus is on teaching algorithmics using the educational environment PiktoMir, which includes software and real-world objects such as radio-controlled robot toys, soft toys, cubes, magnetic cards with command pictograms, and articulated mats. It describes a

step-by-step process of transitioning from the physical world to the virtual world for a deeper understanding of concepts. Additionally, the article presents an approach to object detection tasks using convolutional neural networks, applied in algorithmic competitions. Students solve tasks using physical objects like piktocubes to create programs, which are then transferred by the teacher to the PiktoMir application. This method helps improve the understanding of programming fundamentals in older and middle preschool children.

Keywords: preschooler, elementary school student, algorithmics, robot, textless, screenless, cubes, piktograms, PiktoMir, piktocubes recognition

Литература

1. В. Б. Бетелин, А. Г. Кушниренко, А. Г. Леонов, Основные понятия программирования в изложении для дошкольников // Информатика и ее применения. – 2020. – Том 14, № 3. – С. 56–62.

2. Стартовая страница проекта «ПиктоМир» на сайте ФГУ ФНЦ НИИСИ РАН. URL: https://www.niisi.ru/piktomir/ (дата обращения 01.05.2024)

3. N.Besshaposhnikov, A.Kushnirenko, and A.Leonov. Piktomir: how and why do we teach textless programming for preschoolers, first graders and students of pedagogical universities. CEE-SECR '17: Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia, October 2017. No. 21. P. 1–7. 2017

4. А. Г. Кушниренко, А. Г. Леонов, С. А. Поликарпов. БЕЗОШИБОЧНЫЙ ДВУМЕРНЫЙ ПИКТОГРАММНЫЙ СИНТАКСИС В УЧЕБНОЙ СРЕДЕ ПРОГРАММИРОВАНИЯ ДЛЯ ДОШКОЛЬНИКОВ. ДОКЛАДЫ РОССИЙСКОЙ АКАДЕМИИ НАУК. МАТЕМАТИКА, ИНФОРМАТИКА, ПРОЦЕССЫ УПРАВЛЕНИЯ, 2023, том 511, с. 13–19, DOI: 10.31857/S2686954323700169

5. Kaidong Li, Wenchi Ma, Usman Sajid, Yuanwei Wu, Guanghui Wang. Object Detection with Convolutional Neural Networks. arXiv preprint arXiv:1912.01844, 2019.

6. Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, Brian Lee. A Survey of Modern Deep Learning based Object Detection Models. arXiv preprint arXiv:2104.11892, 2021.

7. Qingqing Xu, Zhiyu Zhu, Huilin Ge, Zheqing Zhang, Xu Zang. Effective Face Detector Based on YOLOv5 and Superresolution Reconstruction, 2021. doi: 10.1155/2021/7748350

8. PyTorch, https://pytorch.org/

9. YOLOv5 https://github.com/ultralytics/yolov5

10. Suorong Yang, Weikang Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, Furao Shen. Image Data Augmentation for Deep Learning: A Survey. arXiv preprint arXiv:2204.08610 , 2022.

11. Pappu Kumar Yadav, J. Alex Thomasson, Stephen W. Searcy, Robert G. Hardin, Ulisses Braga-Neto, Sorin C. Popescu, Daniel E. Martin, Roberto Rodriguez, Karem Meza, Juan Enciso, Jorge Solorzano Diaz, Tianyi Wang. Assessing The Performance of YOLOv5 Algorithm for Detecting Volunteer Cotton Plants in Corn Fields at Three Different Growth Stages. arXiv preprint arXiv:2208.00519, 2022.

12. Леонов А.Г., Райко М.В., Райко И.Г., Ковыршина В.А., Хольхина А.А. Алгоритмиады как элементы ускорения обучения информатике в сборнике ИНФОРМАТИЗАЦИЯ ОБРАЗОВАНИЯ И МЕТОДИКА ЭЛЕКТРОННОГО ОБУЧЕНИЯ: ЦИФРОВЫЕ ТЕХНОЛОГИИ В ОБРАЗОВАНИИ Материалы VI Международной научной конференции, место издания Красноярский государственный педагогический университет им. В.П. Астафьева Красноярск, тезисы, с. 179-186

Информационная карта течения беременности у женщин с рубцом на матке после кесарева сечения в анамнезе

Н.Ю.Земскова[1], С.Ю.Лукашенко[2]

[1]ГБУЗ МО МОНИИАГ, Москва, Россия, fluimucil@yandex.ru

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, s_lukashenko@mail.ru

Аннотация. Приведена информационная карта наблюдения течения беременности пациенток с рубцом на матке после кесарева сечения в анамнезе. Карта является результатом структурной организации данных для анализа характера осложнений беременности, ассоциированных с наличием рубца на матке, изучения особенностей родоразрешения пациенток исследуемых групп и роли метропластики, как метода восстановления репродуктивной функции. Обсуждаются особенности использования карты для создания банка данных на ЭВМ.

Ключевые слова: медицинская информатика, искусственный интеллект, структурная организация данных, анализ данных, акушерство, беременность, кесарево сечение, УЗИ, рубец на матке.

1. Введение

Предложенная работа является результатом совместных исследований врачей отделения ультразвуковой диагностики Московского НИИ акушерства и гинекологии (ГБУЗ МО МОНИИАГ) и математика, сотрудника ОПМИ НИИСИ РАН (ФГУ ФНЦ НИИСИ РАН).

Авторами опубликованы несколько работ посвященных изучению течения беременности женщин, перенесших в анамнезе кесарево сечение (КС), в том числе и в одном из номеров этого журнала за прошлый год [1], [2].

В статьях авторов были изучены данные 150 пациенток с рубцом на матке после кесарева сечения в анамнезе с различным исходом беременности. Выделена группа с толщиной остаточного миометрия 3 мм и более в первом триместре беременности (срок до 12 недель), рубец на матке которых можно считать "состоятельным", для которых динамика изменений рубца не отягощает течение беременности. Для 86 пациенток, с исходно тонким рубцом, был построен алгоритм раннего (к 26 неделе гестации) выявления группы с относительно удовлетворительной динамикой истончения рубца в течение беременности и группы высокого риска развития несостоятельного рубца. Изучено течение беременности для различных клинических групп, характер осложнений беременности, ассоциированных с наличием рубца на матке, особенности родоразрешения пациенток исследуемых групп и роль метропластики, как метода восстановления репродуктивной функции женщины.

Мы получили большой и активный отклик на эти публикации, что неудивительно. Доля КС достигает четверти от всех родоразрешений в мире, а во многих странах она существенно больше. Только в МОНИИАГ и Московской области за 15 лет (2009 - 2023 годы) произведено более 300 тысяч кесаревых сечений, даже и в нашем институте, в связи с определенным контингентом беременных, частота КС за последние 15 лет увеличилась более чем на 20% и составила в 2023 году - 45%, хотя сотрудники МОНИИАГ являются последовательными сторонниками естественных родов. То есть огромное количество женщин при следующей беременности вынашивают ребенка, имея рубец на матке, а значит и риск развития несостоятельности этого рубца, вплоть до разрыва матки.

При наших исследованиях мы использовали разработанные академиком И.М.Гельфандом и его группой сотрудников-математиков методы медицинской информатики и инженерии знаний, [3], [4]. Междисциплинарность подхода к научным проблемам являлась отличительной чертой этого замечательного ученого.

Принципы совместной работы специалистов в области точных наук и врачей в такой слабо формализованной области, как медицина, вызывала и сейчас вызывает огромный интерес. Книга об опыте этой работы выдержала три издания и есть уверенность, что интерес к этим технологиям и методом структурной организации данных, изучением алгоритмов принятия решений высококлассными специалистами, будет только расти. Методы исследования, такие как "диагностические игры", моделирование ситуации близкой к профессиональной деятельности специалиста-медика и использование при

этом особенностей мышления и врача и специалиста в области точных наук, применение знаний в области психологии да и философских теорий познания этого мира, дают тот комплексный подход (синергию), который содержит огромный потенциал. Выявление алгоритмов решения прогностических задач, построения классификаций, решение задач дифференциальной диагностики врачами самой высокой квалификации, позволяют также быстро и качественно структурировать медицинские данные, создавать банки данных на ЭВМ и проводить статистическую "проверку" научных результатов и гипотез. На основе исследований такого типа создаются компьютерные программы, которые ранее было принято называть информационными системами, а сейчас больше принято обозначать как системы искусственного интеллекта (ИИ).

25-30 лет назад в мире был очень оптимистичный взгляд на будущее таких систем. Предполагалось, что "научив" компьютер думать как врач-профессор самого высокого уровня квалификации, мы можем в каких-то случаях заменить этого врача компьютером с программой искусственного интеллекта, например, в отдаленных частях мира, где у жителей просто отсутствует медицинское обслуживание. Однако время показало, что переложить ответственность за принятие решений на плечи "железного", а точнее, "кремниевого" друга не получится, и соответствующие программные продукты (ИИ) заняли свое достойное место дополнительного исследования, помогающего врачу принимать решения, оптимизирующие выбор лечения и алгоритмы наблюдения за пациентами. В любой ситуации ответственность за принятые решения перед пациентом, богом и прокурором лежит на лечащем враче.

Частью результатов группы И.М.Гельфанда являлась разработка информационных карт для создания электронных банков данных. Карты, как правило, разрабатывались как для проведения какого-то конкретного научного исследования, но иногда и для создания электронной истории болезни или какого-то ее фрагмента. Карты публиковались и отдельными изданиями, и в виде разделов более крупных публикаций, в том числе книг, [3], [5], [6], [7].

Данная публикация является продолжением этой традиции. Работа по созданию карты, в каком бы виде она не разрабатывалась - "бумажным" или "электронном", является долгой и кропотливой, и включает в себя много итераций. Обычная ситуация, которая встречается в этой работе - собрав довольно большой блок информации, исследователи обнаруживают, что какого-то раздела или просто параметра все-таки не хватает и надо поднимать опять несколько десятков историй болезни или/и созваниваться с пациентками. Поэтому структура используемой карты сбора информации является одним из самых актуальных вопросов, интересующих коллег из других научных коллективов, занимающихся аналогичными исследованиями. И в тоже время эта структура данных является одним из секретов, тщательно охраняемых, во всяком случае до тех пор, пока по этой теме не будет защищена планируемая по ней диссертация.

2. Карта беременной с рубцом на матке, правила и особенности ее заполнения

В нашем исследовании данные о наблюдениях беременных состоят из разделов, отличающихся по своей структуре. Большинство разделов содержат одну строчку данных о пациентке, например, о наличии сопутствующих заболеваний, характере осложнений беременности, ее исходе, а также данные УЗИ, включающие одно наблюдение для каждой пациентки, содержащие интегральные характеристики особенностей течения беременности. Другой блок содержит данные результатов УЗИ, проводившихся неоднократно для каждой пациентки, то есть каждой пациентке соответствует таблица (до 7 строк) значений параметров, изменяющихся в течении беременности. В нашем случае такая таблица динамических параметров всего одна, то есть структура банка данных довольно простая.

Однако сбор информации отнюдь не является простой задачей и такая слабая изученность столь актуальной тематики объясняется, в первую очередь, сложностью сбора материала. Материал собирался врачом ультразвуковой диагностики в тесном контакте с врачами-клиницистами, сначала акушерами-гинекологами, а затем и педиатрами. Длительность наблюдения каждой пациентки была значительной - от момента постановки беременной на учет в лечебном учреждении в первом триместре беременности (в сроке 5-12 недель), до родоразрешения (в случае доношен-ной беременности до срока 37-40 недель).

КАРТА ТЕЧЕНИЯ БЕРЕМЕННОСТИ У ЖЕНЩИН С РУБЦОМ НА МАТКЕ ПОСЛЕ КЕСАРЕВА СЕЧЕНИЯ В АНАМНЕЗЕ

Номер карты _____
ФИО пациентки _____
_____

Возраст_____ лет

ИСХОД НАСТОЯЩЕЙ БЕРЕМЕННОСТИ

+ **Прерывание беременности в сроке** _____ нед
Операция:
+ Экстирпация матки с плодным яйцом
+ Удаление плодного яйца и метропластика
+ Другая _____

+ **Родоразрешение при пролонгированной беременности**
в сроке _____ нед
+ **Операция:** + экстренная + плановая
+ КС с экстирпацией матки в родах
+ КС и метропластика
+ КС без метропластики
+ **Роды** через естественные родовые пути
_____
+ **Дополнительные операции**
+ Миомэктомия во время беременности в сроке (нед)_____
+ Миомэктомия при родоразрешении
_____

АНАМНЕЗ

Паритет
Количество КС в анамнезе _____
Давность последнего КС_____ (лет)
Всего беременностей_____
Естественных родов_____
Абортов_____ Выкидышей_____
Живых детей у женщины_____

+ **Отягощенный акушерский анамнез**
+ Гибель плода:
+ антенатальная
+ интранатальная
+ постнатальная
+ Привычое невынашивание
+ Бесплодие
+ первичное в анамнезе
+ вторичное
с попытками эко _____
+ Преждевременные роды
_____

+ **СОПУТСТВУЮЩИЕ ЗАБОЛЕВАНИЯ**
+ **Экстрагенитальные**
+ **Заболевание глаз**
+ Миопия (степень)
+ слабая + средняя + выраженная
_____ диоптрий
+ Заболевание сетчатки глаза
+ Астигматизм
_____
+ **ЛОР-заболевания**
+ Фарингит + Тонзиллит
_____
+ **Дыхательной системы**
+ Бронхит + Астма
_____ + ХДН
+ **Сердечно-сосудистые**
+ Порок сердца
+ врожденный + ревматический
+ ПМК
+ Гипертензия
+ Кардиосклероз
+ Варикоз
+ вен НК + малого таза
+ ВСД
+ по гипотоническому типу
+ по гипертоническому типу
_____
+ **НМК** в анамнезе _____
+ **Заболевания ЖКТ**
+ Гастрит + Гастродуоденит
+ Язвенная болезнь:
+ желудка + ДПК
+ Холецистит + ЖКБ
+ Панкреатит
+ Гепатит
_____
+ **Эндокринные**
+ Ожирение, степень_____
+ СД, тип (1 или 2)_____
+ Гипотиреоз
+ Эутиреоз
+ эутиреоидный зоб + узловой
+ АИТ
+ Гиперандрогения
_____
+ **Заболевания почек**
+ Пиелонефрит
+ Пиелэктазия
_____
+ **Опорно-двигательного аппарата**
+ Остеохондроз
+ Ревматизм
+ Симфизиопатия
+ Остеомиелит
+ Остеопения + Остеопороз
_____

Рисунок 1. Карта беременной, часть 1

+ **Нервной системы**
  + Энцефалопатия
  + Эпилепсия
  + ДЦП

_____

+ **Тромбофилия**
  + носитель
  + наследственная
  + сочетанная
+ **Антифосфолипидный синдром**
+ **Антитела к ХГЧ**
+ **Инфекционно-воспалительные**
  + Пельвиоперитонит
  + Перитонит
  + LUES_____

_____

+ **Инфицирование внутрклеточными микроорганизмами**:
  + Уреаплазмоз + Хламидиоз
  + ВПГ + ВПЧ + ЦМВ + ВЭБ
+ **Гинекологические заболевания в анамнезе**
  + Миома матки
    + единичная + множественная
  + Киста и кистомы яичника
  + Дисфункция яичников
  + Аномалия матки _____
  + Эндометриоз:
    + аденомиоз
    + гениталий
    + распространенный
  + Кольпит
  + Хр. цервицит
  + Эктопия шейки матки_____

_____

+ **Травмы в анамнезе**
  + Переломы_____
  + Черепно-мозговые_____
  + Тяжелые сочетанные_____

_____

+ **Операции в анамнезе**
  + На глазах:
    + ЛКС др_____
  + На трахее
  + Спленэктомия
  + Резекция печени
  + На черепе после травмы
  + Эндопротезирование после травмы
  + По поводу варикоза НК
  + Герниопластика
  + Операции на молочных железах

_____

  + По поводу кист яичников

НАСТОЯЩАЯ БЕРЕМЕННОСТЬ

+ **Прегравидарная подготовка**
  + Метропластика в анамнезе

_____

_____

+ **Осложнения этой беременности**
  + Анемия (степень тяжести)

_____

  + Отеки беременной
  + Угроза прерывания беременности в сроке_____
  + Гиперензия беременной
    + риск преэклампсии
  + Гипотония беременной
  + Глюкозурия
  + ГСД в сроке_____
    + с инсулинотерапией
  + Протеинурия
  + Резус-отрицательная кровь
    + резус-сенсибилизация
      + внутриутробные ПК плода

_____

Показания к прерыванию беременности
  + Несостоятельный рубец
  + Отслойка плаценты

_____

Показания к оперативному родоразрешению
+ К экстренной
  + Разрыв матки
  + Несостоятельный рубец
  + Отслойка плаценты
  + Начало родовой деятельности
  + Излитие вод

_____

+ К плановой
  + 2 и более КС в анамнезе
  + Метропластика в анамнезе
  + Экстрагенитальное заболевание

_____

  + Осложненное течение беременности

_____

  + Тонкий рубец
  + Предлежание плаценты
  + Истмоцеле
  + Врастание плаценты в рубец
    + и в др органы
  + Возраст пациентки

_____

_____

Рисунок 2. Карта беременной, часть 2

ДАННЫЕ УЗИ О СОСТОЯНИИ РУБЦА НА МАТКЕ И ПЛАЦЕНТЫ

\+ **Ниша, втяжение**
степень выраженности (1-4) ____
выявлены в сроке _____ нед
\+ **Грыжа**
степень выраженности (1-4) ____
выявлена в сроке _____ нед
\+ **Истмоцеле**
выявлено в сроке _____ нед
\+ **Признаки врастания плаценты в рубец**
\+ Выявлены по данным УЗИ
в сроке _____ нед
\+ Выявлены при гистологическом исследовании
\+ **Предлежание плаценты**
выявлено в сроке _____ нед

ДИНАМИКА ТОЛЩИНЫ РУБЦА
(толщины остаточного миометрия)

Номер УЗИ измерения у пациентки _____
Номер УЗИ измерения у пациентки в обратном порядке _____

Срок беременности при измерении рубца _____(нед)

Толщина рубца (мм):
Минимальное значение _____
Максимальное значение _____

Перцентильная оценка минимальной толщины рубца_____

Новорожденный:
\+ один ребенок \+ двойня
Масса/рост _____/_____
Перцентильная оценка массы
Оценка по шкале Апгар ____/_____
\+ Недоношенность _____
\+ ЗВУР _____
\+ Признаки ВУИ _____

Динамика истончения рубца
Срок выявления выраженного истончения рубца (толщина минимального остаточного миометрия менее 1 мм) _____нед
Срок выявления критического истончения рубца (практически полное отсутствие остаточного миометрия в зоне зубца, отмечаем толщину рубца как 0,1 мм) _____недель

Классификаторы (пример)
Номер группы по исходу беременности:
1 – состоятельный рубец, в 1 триместре минимальная толщина остаточного миометрия (МТОМ) в зоне рубца - от 3 мм и более
2 – МТОМ в 1 триместре менее 3 мм, пролонгированная беременность, нет раннего истончения рубца, МТОМ в 26 недель не менее 1 мм
3 – МТОМ в 1 триместре менее 3 мм, пролонгированная беременность, раннее истончение рубца, МТОМ в 26 недель менее 1 мм
4 – прерывание беременности в сроках 5-21 недель

Рисунок 3. Карта беременной, часть 3

Все это время каждую пациентку надо наблюдать, изучая динамику как ультразвуковых, так и клинических параметров. Непросто врачу не только не «потерять из виду» включенных в исследование пациенток, но и вовремя провести все необходимые «свои» исследования (УЗИ) и своевременно собрать полный комплект клинико-анамнестических и других лабораторных данных. Это трудная и кропотливая работа, связанная с очень высоким уровнем ответственности специалиста.

При заполнении бумажной карты (рис. 1-3) либо вводится значение самого параметра, например возраст, либо отмечается наличие соответствующего заболевания, его типа, осложнения, синдрома и т.д. В последнем случае перед параметром стоит знак +, который мы и обводим, подтверждая наличие этого заболевания. Например, заполняя раздел + Экстрагенитальные заболевания, при их наличии мы обводим знак + и переходим к заполнению подразделов этого раздела, которые напечатаны в карте с увеличенным отступом и которые также помечены знаком + или содержат отмеченное место для заполнения числовых данных. Внутри этих подразделов возможно тоже наличие подразделов, которые сдвинуты, благодаря еще большему отступу перед началом текста. Такая визуальная структура в виде "лесенки" позволяет "видеть" начало и конец раздела. Например, если у пациентки не было экстрагенитальных заболеваний, то легко пропустить весь этот раздел с его подразделами и перейти к заполнению следующего, что упрощает работу с картой, а значит и снижает количество ошибок ввода данных. При работе с бумажной картой фактически используется бинарная кодировка (да, нет), дополненная числовыми

и текстовыми (небольшим количеством) данными. Создание компьютерной карты для проведения какого-либо научного исследования, часто диссертационного, незначительно отличается от создания "бумажной" карты. Хотя кодировку раздела (подраздела), где значения являются взаимоисключающими, удобнее объединять в один параметр с большим количеством значений. Это особенно ускоряет написание программ-запросов, если программы анализа данных имеют режим пакетной обработки.

Обычно используются простейшие карты ввода данных программ типа ACCESS или (для числовых таблиц) EXCEL, которые легко конвертируются в любой формат, необходимый для программ обработки данных. У одного из авторов был опыт работы с использованием более сложных программных оболочек для сбора данных протоколов операций, которые позволяли создавать более сложные интеллектуальные переходы к подразделам внутри системы ввода данных, а бонусом для врача была распечатка протокола операции, написанная с соблюдением всех правил грамматики, за что врачи ее не просто легко приняли, но и, можно сказать, высоко оценили.

Эта программная оболочка была создана в ИППИ РАН и передана нам для тестирования. Созданные на ее базе программы долгие годы работают в МОНИИАГ и количество введенных данных насчитывают десятки тысяч, [8]. Создание таких программ ввода данных требует значительных усилий и оправданы при создании систем с вводом больших объемов наблюдений, например, при создании электронных историй болезни. В нашей работе количество наблюдений было всего 150, то есть, как обычно в научном исследовании, было меньше количества параметров и наша карта модернизировалась в процессе работы. В таких исследованиях создание сложных программ ввода данных просто не оправдывает затраченных усилий.

Идентификационные данные. К этим данным мы отнесли номер карты - идентификационный номер записи пациентки в нашем архиве данных, ФИО беременной, ее возраст. Исходя из нашего опыта, можем сказать, что этих трех параметров достаточно для идентификации пациентки, хотя в материале могут встречаться и полные тезки, и ровесницы, в номер пациентки может "закрасться ошибка". Нужной информацией для связи с пациенткой являются данные для возможности связи с ней на протяжении беременности, такие как адрес проживания, номер телефона ее (а может и ее ближайших родственников). Однако эти "личные данные в расширенном формате" мы предпочитаем хранить в отдельной таблице базы данных или обычным списком, для проведения анализа материала эти данные не нужны, но такое хранение повышает степень защиты конфиденциальной информации.

Исход настоящей беременности. Этот раздел содержит данные о главном результате исследования, о том удалось ли сохранить беременность или ее пришлось прервать, для пролонгированной беременности - насколько успешной оказалась эта пролонгация. Кому-то из пациенток понадобилось провести операцию метропластики, восстанавливающую фертильность. Напомним, что срок доношенности начинается от 37 недель беременности. В соответствии со стандартным определеним преждевременных родов ВОЗ (источник - официальный доклад ВОЗ «Born Too Soon: The Global Action Report on Preterm Birth», 22 мая 2012, Нью-Йорк), преждевременно рожденные дети подразделяются на 3 категории:

1) преждевременно рожденные на поздних сроках беременности – на сроке 32-36 недель 6 дней (большинство таких детей выживает при необходимом уходе);

2) значительно преждевременно рожденные (ранние роды) – на сроке 28-31 недель и 6 дней беременности. Этим детям требуется особый уход, многие из них выживают;

3) сверхранние роды – дети, рожденные на сроке до от 22 до 27 недель и 6 дней беременности. Для выживания этим детям требуется интенсивный, дорогостоящий уход. В развитых странах их выживаемость составляет 90%, но они могут страдать от пожизненных форм физической и неврологической инвалидности и испытывать трудности в обучении. В странах с низким уровнем дохода выживает лишь 10% таких детей.

Завершение беременности в сроки до 21 недели считается ее прерыванием. В нашей стране в сроки до 12 недель беременности ее прерывание возможно по желанию женщины, в сроки 13-21 неделю - по медицинским показаниям и с согласия женщины (несостоятельность рубца, начавшийся выкидыш и т.д.).

Двум пациенткам была выполнена миомэктомия, одной во время беременности, другой - при родоразрешении. Поэтому мы включили в этот раздел "дополнительные операции", в МОНИИАГ иногда практикуются при КС и другие дополнительные хирургические манипуляции, например, удаление кист яичников, поэтому добавлена пустая строка.

Заметим, что в нашем материале были сверхранние роды, но не было перинатальных потерь. При наличии таковых в этот раздел следует добавить их описание, которое можно взять из раздела "Отягощенный акушерский анамнез".

Паритет. Количество кесаревых сечений в анамнезе является очень важной характеристикой для нашего исследования. В нашем материале количество КС в анамнезе было от 1 до 3, этот параметр влияет как на исход беременности (при наличии 3-х КС в анамнезе исход беременности хуже, [9]), так и на выбор между оперативным родоразреше-нием или естественными родами, 2 КС в анамнезе считали показанием к оперативному родоразрешению, относительным показанием считали отсутствие живого ребенка у женщины. Напомним, что все пациентки являются повторнородящими.

Разделы **Отягощенный акушерский анамнез** (ОАА), а также все разделы блоков данных **Сопутствующие заболевания** и **Настоящая беременность** (клинические данные) занимают значительную часть карты и были тщательно проанализированы на предмет связи с исходом беременности и развитием осложнений, ассоциированных с рубцом на матке. Эти параметры влияют лишь на выбор оперативного родоразрешения и все они отражены (повторены) в разделе Показания к оперативному родоразрешению. Если мы хотим только проверить на новом материале полученные нами классификации, то можно перейти к сокращенной ("редуцированной") карте, удалив эти разделы, оставив лишь раздел показаний к операции. Редуцированная карта будет по объему практически вдвое меньше. Но здесь есть еще такой важный момент, связанный с традицией описания материала в медицинских публикациях (статьях и диссертациях). Распределения клинико-анамнестических параметров принято описывать очень подробно. Мы в данной публикации представляем тот вариант карты, который мы использовали при нашем исследовании. Набор каких-то параметров в этих разделах определялся заболеваниями, которые мы наблюдали у наших пациенток. Надеемся, что эта работа будет полезна, а что в ней сократить и что еще добавить наши читатели решат сами.

Данные УЗИ о состоянии рубца на матке и плаценты. Этот раздел карты фактически посвящен выявленным на УЗИ осложнениям течения беременности, ассоциированным с наличием рубца на матке. Наличие ниши при УЗИ рубца считается свидетельством истончения слоя миометрия. Истмоцеле - мешковидный дефект без миометрия («грыжа») передней стенки матки на уровне перешейка матки, соединенный с полостью матки. Грыжа и ее крайний вариант истмоцеле, а также врастание плаценты в рубец - являются грозными осложнениями течения беременности у женщин с рубцом на матке. Предлежание плаценты встречается и у беременных без рубца на матке, но удивляет достоверно большая частота встречаемости у пациенток с рубцом на матке. Наличие этих осложнений, наряду с другими проблемами наблюдения пациенток с рубцом на матке, еще раз подтверждают, что течение беременности после КС невозможно отнести к физиологическому.

Динамика изменений толщины рубца (толщины остаточного миометрия) на протяжении беременности, то есть все измерения толщины рубца, проведенные при УЗ-исследованиях. Этот раздел отличается по структуре от других разделов данных, для каждой пациентки может быть несколько измерений, то есть таблица измерений, в нашем материале было от 1 до 7 наблюдений у беременной. У женщин, которым была прервана беременность, УЗИ-исследований было меньше, чем у пациенток с пролонгированной беременностью. Мы приводим в карте список параметров, составляющих таблицу, а количество строк в этой таблице будет отражать количество УЗИ, выполненных пациентке за время беременности. Обратим внимание на два параметра, первый - технический, наряду с обычной нумерацией строк таблицы очень удобно добавить параметр нумерации в обратном порядке. Это позволяет в базе данных сразу отметить последнее перед родоразрешением исследование, а в первой строке таблицы показывает количество всех УЗИ у пациентки. Этот параметр очень удобен и заполняется после окончания сбора материала.

Параметр "Перцентильная оценка минимальной толщины рубца" возник уже как результат анализа собранного материала и относится к алгоритму построения групп "благополучия" и группы высокого риска течения беременности. Эти исследования выполнены с использованием оригинального пакета программ, разработанного д.ф.-м.н. Котовым Ю.Б. [10], [11]. Так этот параметр определяется по-разному для группы пациенток с "состоятельным" рубцом (минимальная толщина остаточного миометрия 3 мм и более в первом триместре) и для остальных пациенток [2].

Аналогично и раздел **Динамика истончения рубца** является результатом выявления алгоритмов принятия решения при построении классификаций течения беременности у пациенток с рубцом на матке.

Новорожденный. Этот раздел занимает небольшую часть карты и заполнялся по данным наблюдения новорожденного педиатрами. Для наших исследований важны, пожалуй, только наличие и частота недоношенности и связанные с ней распределения оценки состояния плода по Апгар. Для остальных из перечисленных параметров достоверных связей с "качеством рубца" у женщин с пролонгированной беременностью

не было выявлено. Заметим, что подход педиатров к оценке состояния недоношенности новорожденного несколько отличается от подхода акушеров-гинекологов. Педиатры выявляют признаки недоношенности у новорожденного, и при наличии их определенного комплекса выставляют (или нет) диагноз недоношенности. Иногда признаки недоношенности могут выявляться и у детей родившихся в 37 недель беременности, а диагноз «недоношенность», подразумевающий необходимость специального лечения, ставится далеко не всем новорожденным, родившимся в пограничные сроки 35-36 недель беременности. Для срока менее 35 недель диагноз «недоношенность» не вызывает сомнения.

Показания к прерыванию беременности и **Показания к оперативному родоразрешению** (плановому и экстренному). Эти разделы должны бы находиться рядом с разделом **Исход настоящей беременности**, но при работе с бумажной картой удобнее разместить раздел **"Исход..."** в начале карты, упомянутые чуть выше разделы в конце карты, так как эти списки могут содержать параметры почти из любого раздела карты, показания могут иметь сочетанный характер, какие-то из них можно рассматривать как абсолютные показания, какие-то - как относительные. Например, наличие метропластики в анамнезе, предлежание плаценты, врастание плаценты в рубец, выраженная миопия, возраст 58 лет (одно наблюдение) и т.д. - рассматривались нами как абсолютные показания к оперативному родоразрешению; наличие ОАА, артериальная гипертензия и т.д. - как относительные. Необходимым условием проведения естественных родов, кроме отсутствия противопоказаний, было желание пациентки.

Параметры - классификаторы. В нашей карте в процессе исследования мы использовали в процессе сбора и анализа данных некоторое количество служебных параметров, которые обозначали принадлежность к каким-то группам данных. Эти параметры могут быть и временными и постоянными и создаются для удобства - быстрого доступа к группам данным и упрощения написания заданий для программ анализа данных. В качестве примера в карте приведен параметр принадлежности к группам разного уровня риска пациенток по исходу беременности.

3. Заключение

Предложенная информационная карта наблюдения течения беременности пациенток с рубцом на матке после кесарева сечения в анамнезе разработана и тестирована авторами при проведении научных исследований и может быть применена при дальнейших исследованиях в изучении репродуктивной функции пациенток с рубцом на матке. Карта является результатом структурной организации данных для анализа характера осложнений беременности, ассоциированных с наличием рубца на матке, изучения особенностей родоразрешения пациенток исследуемых групп и изучении методов восстановления репродуктивной функции. Авторы, публикуя результаты проведенной работы, ощутили на себе внимание к этой тематике. Данная тема актуальна во всём мире, что выражается в приглашении авторов к участию в ряде престижных международных конференций, посвящённых как женскому репродуктивному здоровью в целом, так и исследованиям в области акушерства и гинекологии.

Публикуемая карта включает в себя параметры, являющиеся результатом научных исследований авторов, которые получены с использованием сочетания как методов статистического анализа данных, так и технологий искусственного интеллекта и инженерии знаний, разработанных группой академика И.М.Гельфанда, памяти которого был посвящен только что опубликованный сборник воспоминаний его сотрудников и учеников "И.М.Гельфанд. К 110-летию".

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН Проведение фундаментальных научных исследований (47 ГП) по теме FNEF-2024-0001 "Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислитель-ных системах (рег. № 1023032100070-3-1.2.1)".

Information Card of the Pregnancy Course of Women with Uterine Scar after Cesarean Section

Zemskova N.Yu.[1], Lukashenko S.Yu.[2]

Abstract: An information card is provided for monitoring the pregnancy course of patients with a uterine scar after a cesarean section in the anamnesis. The card is the result of a structural organization of data for analyzing the nature of pregnancy complications associated with the presence of a scar on the uterus, studying the characteristics of delivery of patients in the study groups, and the role of metroplasty as a method of restoring reproductive function. The peculiarities of the usage of such card for the purpose of creation of a computer data bank system are discussed.

Key words: medical informatics, artificial intelligence, structural organization of data, data analysis, obstetrics, pregnancy, cesarean section, ultrasound, uterine scar.

Литература

1. Земскова Н.Ю., Чечнева М.А., Петрухин В.А., Лукашенко С.Ю. Ультразвуковое исследование рубца на матке после кесарева сечения в прогнозе исхода беременности. "Акушерство и гинекология". N 10, 2020, Москва, "Медицина", стр. 99-104.

2. Земскова Н.Ю., Лукашенко С.Ю., Чечнева М.А. Мониторинг течения беременности у женщин с рубцом на матке после кесарева сечения. "Труды НИИСИ РАН", том 13, № 3, стр. 55-74

3. Гельфанд И.М., Розенфельд Б.И., Шифрин М.А. Очерки о совместной работе математиков и врачей. 3-е издание. Москва, Едиториал УРСС, 2004, 309 с.

4. Гельфанд И.М., Розенфельд Б.И., Шифрин М.А. Структурная организация данных в задачах медицинской диагностики и прогнозирования. Препринт Академии Наук СССР, Научный совет по комплексной проблеме "Кибернетика", Москва, 1982, 55 с.

5. Алексеевский А.В., Гришин В.Л., Клименко П.А., Котов Ю.Б., Лукашенко С.Ю., Сахнина Э.И., Сичинава Л.Г., Федорова М.В. Структура клинических данных в проблеме диагностики и лечения плацентарной недостаточности. Препринт Академии Наук СССР, Научный совет по комплексной проблеме" Кибернетика", Москва, 1988, 42 с.

6. Извекова М.Л., Гринберг А.А., Бабкова И.В., Лукашенко С.Ю. Опыт организации медицинского архива на ЭВМ (На примере банка данных о больных, оперированных по поводу язвенной болезни), Препринт Академии Наук СССР, Научный совет по комплексной проблеме "Кибернетика", Москва, 1987, 42 с.

7. Алексеев В.М., Алексеевская М.А., Гельфанд И.М., Гогин Е.Е., Головня Л.Д., Заславская Р.М., Извекова М.Л., Клюшин Е.С., Мартынов И.В., Саблин В.М,, Сыркин А.Л. Многоцелевая карта больного инфарктом миокарда (для создания банка данных на ЭВМ) Препринт Академии Наук СССР, Научный совет по комплексной проблеме "Кибернетика", Москва, 1981, 32 с.

8. Федорова М.В., Галкина М.С., Котов Ю.Б., Лукашенко С.Ю., Гурьева В.М., Шалаев О.Н. Информационные системы "Кесарево сечение" и "Оперативная лапароскопия". Сборник научных трудов Всероссийской конференции "Основные направления развития информатизации здравоохранения и системы ОМС на 1999-2002 годы". Воронеж,
31 мая - 2 июня 1999 г, стр. 86-87.

9. Земскова Н.Ю., Чечнева М.А., Щукина Н.А., Лукашенко С.Ю. Изучение рубца на матке и сравнение исходов беременности у женщин с однократным и повторными кесаревыми сечениями в анамнезе Материалы XXIV Всероссийского научного форума «Мать и дитя», 27-39 сентября 2023 г., Красногорск МО, стр. 24-25.

10. Котов Ю.Б. Новые математические подходы к задачам медицинской диагностики. Москва, Едиториал УРСС, 2004, 328 с.

11. Котов Ю.Б. Программа получения скользящих нормативов по набору реализаций процесса. ИПМ РАН, Препринт №27, Москва, 1993, 23 с.

Подписано в печать 28.06.2024 г.
Формат 60x90/8
Печать цифровая. Печатных листов 6.75
Тираж 100 экз. Заказ №557

Отпечатано в ФГБУ «Издательство «Наука»
(Типография «Наука»)
121099, Москва, Шубинский пер., 6