20. *Вовна А., Зори А., Хламов М.* Методы и средства измерения концентрации газовых компонент. – Saarbrücken, Germany: LAP LAMBERT Academic Publishing GmbH & Co. KG, 2012. – 244 с.
21. *Соломічев Р.І., Вовна О.В.* Алгоритмічно-структурний синтез системи контролю вибухонебезпечних газових сумішей рудничної атмосфери // Збірник наукових праць XIV міжнародної науково-технічної конференції аспірантів і студентів «Автоматизація технологічних об'єктів та процесів. Пошук молодих». – Донецьк: ДонНТУ, 2014. – С. 313-317.
22. *Вовна О.В., Зорі А.А.* Розробка та дослідження швидкодіючого вимірювача концентрації метану інваріантного до запилення рудничної атмосфери // Наукові праці ДонНТУ. Серія: «Обчислювальна техніка та автоматизація». – Донецьк, 2012. – № 22 (200). – С. 143-150.
23. *Вовна А.В., Зори А.А.* Разработка и исследование экспериментального образца измерителя концентрации метана для угольных шахт // Известия ЮФУ. Технические науки. – 2014. – № 4 (153). – С. 171-177.
24. *Вовна А.В., Зори А.А.* Разработка и исследование радиоэлектронного оптического измерителя концентрации метана // Материалы 23-й Международной Крымской конференции «СВЧ-техника и телекоммуникационные технологии» (КрыМиКо'2013). – Севастополь, 2013. – С. 984-985.
25. *Вовна А.В., Зори А.А.* Оптический измеритель концентрации метана с компенсацией температурного дрейфа // Матеріали міжнародної науково-технічної конференції «Автоматизація: проблеми, ідеї, рішення». – Севастополь, 2013. – С. 142-144.

Статью рекомендовал к опубликованию д.т.н., профессор П.Г. Михайлов.

Вовна Александр Владимирович – Государственное высшее учебное заведение «Донецкий национальный технический университет»; e-mail: Vovna_Alex@ukr.net; 85300, г. Красноармейск, пл. Шибанкова, 2, Украина; кафедра электронной техники; к.т.н.; доцент.

Зори Анатолий Анатолиевич – e-mail: zori@kita.dgtu.donetsk.ua; кафедра электронной техники; заведующий кафедрой; д.т.н.; профессор.

Vovna Aleksander Vladimirovich – State Higher Education Establishment "Donetsk National Technical University"; e-mail: Vovna_Alex@ukr.net; 2, Shibankov area, Krasnoarmejsk, 85300, Ukraine; the department of electronic technics; cand. of eng. sc.; associate professor.

Zori Anatolii Anatolievich – e-mail: zori@kita.dgtu.donetsk.ua; the department of electronic technics; the head of department; dr. of eng. sc.; professor.

УДК 519.688

О.В. Шаповалов, А.Е. Андреев, С.А. Фоменков

РАЗРАБОТКА МЕТОДОВ АВТОМАТИЗАЦИИ РАСПАРАЛЛЕЛИВАНИЯ ПРОГРАММ ДЛЯ СИСТЕМ С ОБЩЕЙ ПАМЯТЬЮ

Рассматривается вопрос автоматизации разработки параллельных программ для систем с общей памятью. Целью данной работы является разработка программного обеспечения, которое предоставит возможность программисту сравнить скорость работы и максимально эффективно использовать различные технологии параллельного программирования применительно к конкретной решаемой задаче. Для сокращения времени разработки предлагается единый интерфейс для работы с технологиями OpenMP, CilkPlus, Intel TBB, PPL, BoostThreads, данный интерфейс используется для автоматического распараллеливания последовательного кода для указанных технологий. Для повышения эффективности распараллеливания предлагается метод, когда для параллельного выполнения одного и того же участка программы могут использоваться несколько технологий параллельного программирования, выбор между которыми происходит на этапе выполнения по разрабо-

танному алгоритму классификации. Разработанный алгоритм для выбора параллельной технологии относится к алгоритмам многоклассовой классификации и является модификацией метода ближайших соседей для работы в условиях ограниченного количества информации. Алгоритм также берет за основу некоторые положения алгоритма множественной логит-регрессии. Для получения необходимой статистики для работы алгоритма классификации был разработан алгоритм обучения, который управляет порядком выбора расчета новых статистических данных, и полученные результаты классификации для текущего объекта используются при классификации следующих. Алгоритм обучения имеет линейную сходимость, параметры сходимости подтверждаются теоретическими и практическими исследованиями. Разработана библиотека параллельных шаблонов, облегчающая разработку эффективных параллельных программ для систем с общей памятью и ускорителями. Помимо технологий OpenMP, CilkPlus, Intel TBB, PPL, BoostThreads, для которых распараллеливание выполняется автоматически, пользователь библиотеки может добавлять свои реализации распараллеливания. Эффективность предлагаемых подходов показана на примере программы, вызывающей умножение матриц, сокращение времени работы составило 43% по сравнению со стандартным подходом к распараллеливанию с использованием одной параллельной технологии.

Автоматическое распараллеливание; адаптивный классификатор; OpenMP; CilkPlus; IntelTBB; PPL; BoostThreads.

O.V. Shapovalov, A.E. Andreev, S.A. Fomenkov

DEVELOPMENT OF AUTOPARALLELIZATION METHODSFOR COMPUTATIONAL SYSTEMS WITH SHARED MEMORY ARCHITECTURE

In this paper problem of automation of parallel software development for shared memory systems is considered. The goal of this work was to develop software that will enable the programmer to compare the performance and maximize the efficiency of using various parallel programming technologies for the specific problem. For decreasing time for development of parallel programs unified interface for parallel technologies OpenMP, Cilk Plus, Intel TBB, PPL, BoostThreads is suggested. For increasing efficiency of parallelization new method is proposed, it is used for parallel execution of some programming code with various parallel programming technologies, the choice between them is realized at runtime with developed classification algorithm. The developed algorithm of selection of the parallel technology is a multi-class classification algorithm; it is a modification of the method of nearest neighbors for a limited amount of information. The algorithm also takes as its basis some parts of the algorithm of multiple logit regression. The necessary statistics were produced for classification algorithm by a developed learning algorithm, which controls the selection of parallel programming technology for calculating the new statistical data and the results from the current classification are used in the following classification. The learning algorithm has a linear convergence, this is proved by theoretically and empirically grounded research findings. This interface is used for automatic parallelization of serial programming code with technologies which are enumerated above. Programming library, that helps to realize efficient parallel programs for shared memory systems with coprocessors and accelerators, is developed. In addition to OpenMP, Cilk Plus, Intel TBB, PPL, Boost Threads parallel technologies, for which parallelization is performed automatically, the programmer, who uses our library, can add his own implementations of parallelism. An efficiency of proposed methods is shown with example program, that calls matrix multiplication method, execution time was reduced by 4 3 % as compared to standard parallelization approach, that uses only one parallel programming technology.

Automatic parallelization; adaptive classifier; OpenMP; Cilk Plus; Intel TBB; PPL; BoostThreads.

Введение. При разработке параллельного программного обеспечения важным вопросом является выбор и эффективное использование технологии параллельного программирования. При этом число технологий постоянно растет, а существующие развиваются – все это приводит к постоянно меняющемуся балансу сил между различными технологиями. В итоге, при начале разработки нового про-

екта программист должен сравнить возможные технологии параллельного программирования, как по эффективности, так и по применимости в его проекте. Для достаточно подробного анализа может потребоваться значительное время. Несмотря на наличие разработок в этой области, в них существует общий недостаток – сравнение технологий происходит либо на типовых задачах (умножение матриц, поиск в ширину на графе и др.), либо вообще теоретически. Абсолютное большинство работ, посвященных разработке параллельных программ ([15, 16] и т.д.), предполагает сравнение параллельных технологий между собой и с последовательной версией, при этом отсутствуют инструменты для автоматизации этого процесса.

Целью данной работы является разработка программного обеспечения, которое предоставит возможность программисту сравнить скорость работы и максимально эффективно использовать различные технологии параллельного программирования применительно к конкретной решаемой задаче.

При типичном подходе к распараллеливанию программ сравнение и выбор одной из технологий происходит единственный раз в начале проекта. Такой подход обуславливается тем, что в дальнейшем сравнение представляет собой достаточно трудоемкий процесс, так как все распараллеливание организовано с помощью одной конкретной технологии. Недостатком данного подхода является вероятность выбора неоптимальной по эффективности распараллеливания технологии для распараллеливания.

Актуальность работы доказывается большим количеством научных работ, в которых происходит сравнение различных параллельных технологий. При этом на данный момент нет единого подхода для сравнения разных технологий, и разработчики параллельных программ вынуждены создавать инфраструктуру для сравнения технологий каждый раз для новой программы. При этом в статьях редко указывается методика тестирования (количество запусков для усреднения, используется ли разогревающий запуск и т.д.), что ставит под сомнение достоверность результатов.

Библиотека параллельных шаблонов. Предлагается новый подход, когда в любой момент разработки программы можно выбрать один из доступных методов распараллеливания:

1. Последовательный запуск. Все циклы исполняются последовательно.
2. Лучшая технология по общему времени выполнения программы.
3. Лучшая технология для каждого запуска цикла.

Под лучшей технологией в данном случае понимается технология, обеспечивающая минимальное время выполнения.

На рис. 1 представлено сравнение времени работы технологий для программы из 4 циклов. Каждый прямоугольник соответствует времени выполнения одного запуска цикла с помощью конкретной технологии. T_S – общее время выполнения последовательного кода, T_C – общее время выполнения при использовании CilkPlus, T_O – общее время OpenMP, T_{BT} – общее время BoostThreads, T_T – общее время TBB. При использовании лучшей технологии по общему времени выполнения программы для распараллеливания будет выбрана технология CilkPlus, так как время выполнения при ее использовании наименьшее.

Разные технологии могут иметь разную эффективность распараллеливания в зависимости от распараллеливаемого цикла и числа итераций, поэтому для максимальной эффективности необходимо выбирать технологию для каждого запуска распараллеливаемого цикла.

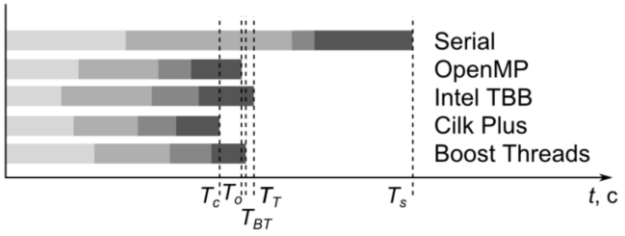

Рис. 1. Метод выбора технологии по общему времени выполнения программы

На рис. 2 показано возможное уменьшение времени работы приложения при использовании выбора технологии для каждого запуска цикла по сравнению с выбором одной технологии. Для данного способа общее время выполнения программы будет составлять $T_{opt} = \sum_{i=1}^{M} T^i{}_{\min}$, где $T^i{}_{\min}$ – минимальное среди всех технологий время работы i-го цикла, M – число запусков распараллеливаемых циклов.

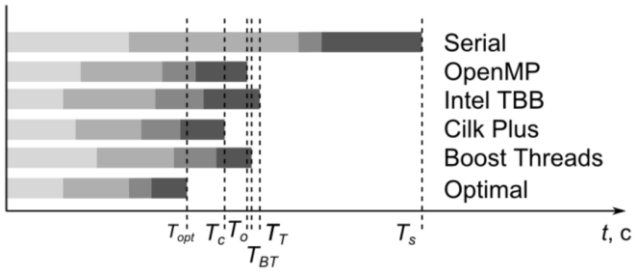

Рис. 2. Способ выбора лучшей технологии для каждого запуска цикла

Рассмотренные подходы к выполнению параллельной программы были реализованы в библиотеки параллельных шаблонов PTL для языка C++[1]. Данная библиотека позволяет пользователю выбирать на любом этапе разработки программы между тремя вышеперечисленными методами выполнения, а также позволяет сократить время разработки параллельной программы за счет автоматической генерации параллельного кода для технологий OpenMP, CilkPlus, IntelTBB, PPL [2] и BoostThreads.

Разные технологии имеют свои особенности и разную эффективность распараллеливания в зависимости от распараллеливаемого цикла и числа итераций, поэтому для получения максимальной эффективности необходимо выбирать технологию для каждого запуска распараллеливаемого цикла. Для выбора технологии параллельного программирования для распараллеливания цикла for на этапе выполнения предложена следующая схема. Пользователь библиотеки реализует свою задачу согласно интерфейсам, представленным в табл. 1. Интерфейс IAloneFunction используется для автоматического распараллеливания с помощью технологий, работающих с общей памятью (OpenMP, CilkPlus, IntelTBB, PPL, BoostThreads и т.д.) и предназначен для выполнения одной итерации цикла, номер итерации является входным параметром. Интерфейс IManyFunction используется при распараллеливании для массивно-параллельных архитектур (графические процессоры, XeonPhi, ПЛИС), когда пользователь сам реализует распараллеливание для диапазона итераций (границы итераций являются входными параметрами), при этом функция должна возвращать время своей работы. Соответствующие обертки над телом функции делаются с помощью лямбда–функций стандарта C++11.

Таблица 1

Интерфейс	Запись на C++	Предназначение
IAloneFunction	std::function<void (int)>	для описания функций обработки одной итерации распараллеливаемого цикла
IManyFunction	std::function<double (int, int)>	для описания функции обработки диапазона итераций распараллеливаемого цикла

После того, как приведение к данным интерфейсам осуществлено, библиотека автоматически генерирует параллельный код с использованием перечисленных технологий.

Имея разные параллельные реализации можно быстро произвести сравнение времени общей работы программы с использованием разных технологий параллельного программирования, что и было реализовано в библиотеке PTL. За счет этого достигается сокращение времени разработки параллельной программы (пользователь реализует только последовательную версию, параллельная версия для разных технологий генерируется библиотекой, сравнение времени работы также происходит автоматически).

Алгоритм выбора технологии распараллеливания. Также был реализован алгоритм, сокращающий общее время выполнения программы за счет переключения используемой технологии во время работы программы. Математически данную задачу можно описать так:

$$\sum_{i=1}^{m} g\left(N_i, NumTech_i\right) \to \min, \tag{1}$$

где $g(N_i, NumTech_i)$– время выполнения распараллеливаемого цикла размера N_i с помощью технологии с номером $NumTech_i$, $i = 1..m$ – последовательность вызовов распараллеленного участка в течение работы программы (в общем случае неизвестна во время выполнения), $NumTech_i \in [0, n-1]$ – номер технологии параллельного программирования, варьируемый параметр, n – число используемых технологий параллельного программирования. Таким образом, задачей оптимизации является выбор технологии параллельного программирования (номера $NumTech$) при известном размере цикла N с целью уменьшения времени работы программы. Данная задача относится к задаче многоклассовой классификации [3, 4], которая используется для определения неизвестной целевой зависимости – отображения $N \to NumTech$. Требуется построить алгоритм для работы адаптивного классификатора, т.е. классификатора, который обучается во время работы.

Номер технологии параллельного программирования можно рассматривать как функцию от размера цикла и данных от предыдущих запусков цикла:

$$NumTech_i = NumTech\left(N_i, H\right), \tag{2}$$

где N_i – размер обрабатываемого цикла, H – исторические данные по предыдущим запускам распараллеливаемой функции. Исторические данные хранятся в виде троек «размер цикла»-«номер технологии»-«время работы», табл. 2.

Таблица 2

N	$NumTech$	$g(N_i, NumTech)$, с
100	0	0,1
1000	2	9
100	1	0,2
1000	0	10
…	…	…

Существует достаточно много алгоритмов классификации. Наиболее применимыми для данной задачи выглядят два из них. Во-первых, это метод ближайших соседей [5] – простейший метрический классификатор, где классифицируемый объект относится к тому классу, которому принадлежат ближайшие к нему объекты обучающей выборки. Во-вторых, это обобщение логистической регрессии [6] для решения задач многоклассовой классификации (множественная логит-регрессия), которая заключается в подгонке данных к некоторой логистической кривой.

Поскольку данные о предыдущих запусках хранятся не в виде прямого отображения $N \to NumTech$ (и могут не приводиться в такой вид), то известные методы решения задачи классификации здесь неприменимы и требуется их модифицировать. Одним из возможных решений может быть интерполяция по известным значениям времени работы [7] для каждой из технологий и предсказание времени выполнения на ее основе (данный вариант можно считать модификацией метода ближайших соседей, где может использоваться локальная интерполяция).

Однако данный подход имеет существенный недостаток, так как плохо подходит для адаптивной классификации. Поскольку после выбора технологии $NumTech_i$ для выполнения цикла с размером N_i уточнение интерполяционной кривой происходит только для выбранной технологии (с номером $NumTech_i$), то данный алгоритм может обеспечивать хорошие результаты в случае большого количества статистических данных для всех используемых технологий, но при адаптивном подходе данный алгоритм будет неспособен работать при отсутствии статистики, а также плохо обучаться.

В итоге, для выбора номера технологии был разработан эвристический вероятностный алгоритм, задачей которого является минимизация исходной функции (1). Данный алгоритм является вариацией метода ближайших соседей, так как одним из основных критериев выбора технологии являются ближайшие статистические данные. Используется следующий подход: для каждого очередного запуска распараллеленного цикла на основе входных данных рассчитываются вероятности выбора разных значений параметров $NumTech$, затем разыгрывается случайная величина и по ней выбирается значение функции.

Для расчета вероятностей принимаются следующие допущения:

- при отсутствии статистики все вероятности равны 1/n;
- вероятность обратно пропорциональна времени работы (берется из статистики для ближайшего к необходимому параметру N), если известна сложность алгоритма в О-нотации, то время берется в пересчете на единицу сложности;
- для оценки «доверия» к статистическим данным используется критерий дальности статистики от запрашиваемого параметра N, вероятность пропорциональна величине $e^{-R_k^2}$, где $R_k = \frac{|N - N_k|}{N}$ – относительное расстояние между требуемым количеством итераций и ближайшим в имеющейся статистике;

- чтобы избежать выбора одного и того же значения номера технологии используется зависимость от числа накопленной статистики $Stat$ для данной технологии: чем меньше статистики, чем больше вероятность выбора этого значения.

Исходя из описания, получим:

$$P_k \propto \frac{e^{-R_k^2}}{\frac{t_k}{O(N_k)}Stat} = \frac{e^{-\left(\frac{|N-N_k|}{N}\right)^2}}{\frac{t_k}{O(N_k)}Stat}. \quad (3)$$

Поскольку выбор из возможных значений $NumTech$ образует полную группу событий, сумма вероятностей должна быть равна единице:

$$\sum_{k=1}^{n-1} p_k = 1, \quad (4)$$

где p_k – вероятность выбора значения k для $NumTech$.

Определим коэффициент пропорциональности, подставив (3) в (4).

$$\sum_{k=1}^{n-1} p_k = c\sum_{k=1}^{n-1} \frac{e^{-R_k^2}}{\frac{t_k}{O(N_k)}Stat} = 1 \Rightarrow c = \frac{1}{\sum_{k=1}^{n-1} \frac{e^{-R_k^2}}{\frac{t_k}{O(N_k)}Stat}}. \quad (5)$$

Таким образом, вероятности будут равны:

$$p_i = \frac{ce^{-R_i^2}}{\frac{t_i}{O(N_i)}Stat} = \frac{e^{-R_i^2}}{\frac{t_i}{O(N_i)}Stat\sum_{k=1}^{n-1} \frac{e^{-R_k^2}}{\frac{t_k}{O(N_k)}Stat}}. \quad (6)$$

Для корректной и эффективной работы алгоритма нужно выполнение условий:

1) время работы функции $NumTech(N,H)$ должно быть намного меньше работы функции $g(N, NumTech)$;
2) также предполагается, что каждая технология может иметь не более одного диапазона, где она эффективнее остальных технологий.

Второе условие позволяет сформировать критерий останова алгоритма: если существует диапазон от N_1 до N_2, и при этом и для N_1 и для N_2 лучшим оказывается одно и то же значение $NumTech = X$, то $\forall N \in [N_1, N_2]$ выбирается значение X. Это сформированный критерий останова для отрезка $[N_1, N_2]$. Если заранее известно, в каком диапазоне $[N_a, N_b]$ может находиться N, то полная остановка алгоритма происходит, когда любой N принадлежащий $[N_a, N_b]$ попадает в некий отрезок $[N_i, N_{i+1}]$, для которого $NumTech$ уже определен согласно критерию выше.

Для каждого поступающего запуска распараллеленного цикла с известным размером выполняется следующий алгоритм:

1. Проверить входной размер цикла на принадлежность к уже известным диапазонам максимальной эффективности технологий. Если не принадлежит, то перейти к пункту 2, если принадлежит – к пункту 4.

2. Рассчитать вероятности выбора разных технологий по формуле 6.

3. Разыграть случайную величину и использовать ее для выбора одной из технологий.

4. Запустить распараллеленный цикл с использованием выбранной технологии с замером времени выполнения.

5. Добавить данные о запуске в статистические данные.

Все результаты тестов сохраняются в xml-файлы и используются при дальнейших запусках программы. Библиотека PTL в первую очередь ориентирована для использования результатов обучения на той же вычислительной системе (ВС), на которой они были получены. При копировании результатов обучения на другую по характеристикам ВС адаптивная параллельная программа переобучится, но первые запуски могут быть неоптимальными.

Пример. Распараллеливание умножения квадратных матриц с использованием технологий PPL и CUDA [8].

Умножение матриц стандартным алгоритмом записывается как $C = A \cdot B$, где элементы матрицы C вычисляются по формуле $c_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj}$, где $i = 1..m, j = 1..q$.

Версия с CUDA не генерируется автоматически, в примере 1 показана реализация с использованием библиотеки cuBLAS.

```
// умножение матриц на видеокарте
voidmatrixMultiply(float *h_A, float *h_B, float *h_C, int N)
{
   // выделяем память на графическом устройстве
   ...
   // копируем матрицы A и B в графическую память
cudaMemcpy(d_A, h_A, mem_size_A, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, mem_size_B, cudaMemcpyHostToDevice);

cublasHandle_t handle;
cublasCreate(&handle);

// запускаем расчет на графическом ускорителе
constfloatalpha = 1.0f;
const float beta  = 0.0f;
cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, N, N, N, &alpha, d_B,
N, d_A, N, &beta, d_C, N);

// копируем матрицу результата C из графической памяти в оперативную
cudaMemcpy(h_C, d_C, mem_size_C, cudaMemcpyDeviceToHost);

// освобождаем память на графической карте
   ...
}
```

Пример 1. Реализация умножения на видеокарте

Чтобы добавить данную реализацию к сравнению с другими параллельными реализациями, создаем обертку над вызовом умножения с замером времени, см. пример 2. Технологии для распараллеливания с общей памятью просто добавляются в список, затем добавляем нашу реализацию, использующую CUDA. Добавляем новый обработчик для нового цикла, указав имя файла для сохранения статистики. Затем запускаем умножение матриц, создав лямбда-функцию для выполнения i-й итерации цикла. При запуске также указываются границы итераций и список технологий для выполнения. Последним параметром указывается сложность алгоритма в O-нотации относительно размера цикла, для умножения она составляет $O(N^3)$.

```
    // создаем обертку (лямбда-функцию) для умножения матриц
    // на графическом ускорителе с замером времени
autolabmdaMul = [a,b,res](int start, int end) -> double
    {
doubletimeStart = AbstractParallel::GetTime();
matrixMultiply(a, b, res, end);
returnAbstractParallel::GetTime() - timeStart;
    };
   // перечисляемтехнологиидляцентральногопроцессора
std::vector<Technology> technologies;
technologies.push_back(Technology::Serial);
technologies.push_back(Technology::PPL);

// перечисляем набор реализаций для сопроцессоров
   // в данном случае - умножение матриц на видеокарте
std::vector<IManyFunction>addImpls;
addImpls.push_back(labmdaMul );

// создаем класс, управляющий параллелизмом в проекте
ParallelUtilsBasebaseParallel;
   // добавляем обработку нового цикла
autopUtils = baseParallel.AddNewParUtils(technologies, "MatrixMul.xml");

// запускаем умножение матриц с выбранными  параметрами распараллеливания
pUtils->RunInParallel([&](inti)
   {
for(int j = 0; j < N; j++)
      {
res[i * N + j] = 0;
for (int k = 0; k < N; k++)
res[i * N + j] += a[i * N + k] * b[k * N + j];
}
} , 0, N, addImpls, 3);
```

Пример 2. Реализация метода для использования нескольких технологий для распараллеливания одного цикла

Рассмотрим результаты (время) работы программы, в случае, если умножение матриц вызывается 3 раза с размерами 256, 512 и 1024 (табл. 3). При выборе одной технологии, в данном случае CUDA суммарное время оказывается почти в 2 раза больше, чем при использовании разных технологий для разных запусков.

Таблица 3

Размер матриц	Время выполнения с указанной технологией, с			Лучшая по каждому циклу, с
	Последовательная	PPL	CUDA	
256	0.056	**0.015**	0.242	**0.015**
512	0.551	**0.151**	0.248	**0.151**
1024	4.749	1.225	**0.261**	**0.261**
Общее время	**5.356**	**1.391**	**0.751**	**0.427**

Алгоритм адаптивной классификации в данном случае будет эффективен, если количество запусков умножения будет больше. Если алгоритм начнет свою работу при отсутствии статистики, то первые 3 запуска (независимо от размеров цикла) пройдут с разными технологиями (последовательная, PPL и CUDA), так как наибольшее влияние будет оказывать критерий количества статистики. Обучение будет остановлено, когда будет найдена граница эффективности между технологиями PPL и CUDA (размер цикла – около 600). В лучшем случае обучение может закончиться за $L = 4 \cdot (n-1)$ шагов, где n – количество технологий. Для трех технологий получается соответственно 8 запусков.

На рис. 3 показана минимальная схема запусков для завершения обучения для программы, вызывающей умножения матриц. С последовательной технологией будет необходимо запустить с минимальным (min) и максимальным (max) из интересующих размеров, PPL – min, 600 и 601 (если граница эффективности проходит между 600 и 601), CUDA – 600, 601 и max.

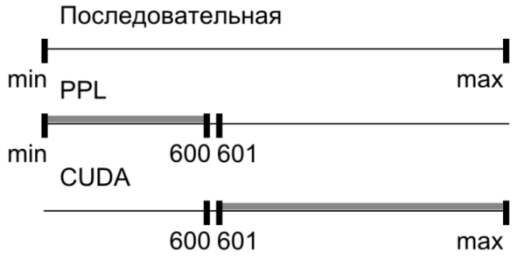

Рис. 3. Минимальная схема запусков для завершения обучения

Среднее количество запусков также будет прямо пропорционально количеству технологий, так как основное количество запусков будет приходиться на определение границ между технологиями. При требовании только одного диапазона максимальной эффективности для каждой из технологий получаем 2 таких участка для каждой из технологий минус 2 обработки для граничных областей. Таким образом, предлагаемый алгоритм адаптивной классификации имеет линейную сходимость.

Выводы. Была разработана программная библиотека PTL, облегчающая разработку эффективных параллельных программ для вычислительных систем с общей памятью и для систем с использованием ускорителей или сопроцессоров. Был разработан новый метод управления выполнением параллельной программы, отличающийся от известных возможностью выбора технологии параллельного программирования индивидуально для каждого вызова распараллеленного кода, что позволяет сократить время работы программы при эквивалентности результатов расчета. Был разработан модифицированный алгоритм многоклассовой классификации, отличающийся от метода ближайших соседей возможностью адаптивного обучения в условиях неполноты информации, который позволяет находить соответствие между входными параметрами (например, размером цикла) и технологией параллельного программирования во время режима обучения автоматизированной системы. Все это позволяет сократить время разработки параллельной программы и повысить ее эффективность. Для представленного в работе примера время работы программы было сокращено на 43 % при сравнении со стандартным подходом, когда используется одна технология программирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Шаповалов О.В., Андреев А.Е., Фоменков С.А.* Применение библиотеки параллельных шаблонов в разработке геометрического ядра // Вестник компьютерных и информационных технологий. – 2014. – № 11. – С. 8-12.
2. ParallelPatternsLibrary (PPL) [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/dd492418.aspx (дата обращения: 03.05.2015).
3. *Айвазян С.А., Бухштабер В.М., Енюков И.С., Мешалкин Л.Д.* Прикладная статистика: классификация и снижение размерности. – М.: Финансы и статистика, 1989.
4. *Клевцов С.И.* Прогнозирование изменений физической величины в реальном времени с использованием линейного адаптивного фильтра // Известия ЮФУ. Технические науки. – 2013. – № 5 (142). – С. 180-185.
5. *Shakhnarovish G., Darrel T., Indyk P.* Nearest-Neighbor Methods in Learning and Vision // Pattern Analysis and Applications. – 2008. – Vol. 11, Issue 2. – P. 221-222.
6. Freedman, D. Statistical Models: Theory and Practice // Cambridge University Press. – 2009. – p. 128.
7. *Ромм Я.Е., Стаховская И.И.* Временная сложность параллельного кусочно-интерполяционного вычисления функций // Известия ЮФУ. Технические науки. – 2013. – № 5 (142). – С. 160-165.
8. Центр разработки NVIDIA CUDA [Электронный ресурс]. – Режим доступа: https://developer.nvidia.com/cuda-zone (дата обращения: 03.05.2015).
9. *Jackson Q., Landgrebe D.A.* An adaptive classifier design forhigh-dimensional data analysis with a limited training data set // IEEETrans. Geosci. Remote Sens. – 2001. – Vol. 39. – P. 2664-2679.
10. *Антонов А.С.* Параллельное программирование с использованием технологии OpenMP: Учеб. пособие. – М.: Изд-во МГУ, 2009. – 77 с.
11. *Бочаров Н.В.* Технологии и техника параллельного программирования [Электронный ресурс]. – Режим доступа: http://dks.invitation.ru (дата обращения: 03.05.2015).
12. *Карпов А.Н.* Тестирование параллельных программ [Электронный ресурс]. – Режим доступа: http://www.viva64.com/ru/a/0031 (дата обращения: 03.05.2015).
13. *Жариков Д.Н., Лукьянов В.С., Шамов Е.А., Шаповалов О.В., Попов Д.С.* Моделирование динамики электронного потока в скрещенных полях на кластере из центральных и графических процессорных устройств / // Современные наукоёмкие технологии. – 2010. – № 4. – С. 58-60.
14. *Немнюгин С.А., Стесик О.Л.* Параллельное программирование для многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 2002. – 400 с.
15. *Кузьмина К.С., Марчевский И.К.* Об ускорении вычислений при решении двумерных сопряженных задач гидроупругости вихревыми методами // Вестник Пермского Национального исследовательского политехнического университета. Аэрокосмическая техника. – 2014. – № 39. – С. 145-163.
16. *Yokota R., Barba L.A.* Hierarchical N-body simulations with auto-tuning for heterogeneous systems // Computing in Science and Engineering. – 2012. – Vol. 14. – P. 30-39.
17. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем: Пер с англ. Дж. Ортега. – М.: Мир, 1991. – 367 с.
18. Сообщество разработчиков программного обеспечения [Электронный ресурс]. – Режим доступа: http://software.intel.com/ru-ru/?iid=gg_work-RU+home_software (дата обращения: 03.05.2015).
19. *Шамов Е.А., Шаповалов О.В., Стуров Д.А., Горобцов А.С.* Анализ эффективности применения технологии Nvidia CUDA для задач физической электроники // Известия Волгоградского государственного технического университета: Межвуз. сб. науч. ст. – Волгоград: ВолгГТУ, 2009. – № 12 (60). – 128 с. – (Сер. Актуальные проблемы управления, вычислительной техники и информатики в технических системах. Вып. 7).
20. *Шаповалов О.В*. Параллельные методы численного решения жестких СОДУ // XV Региональная конференция молодых исследователей Волгоградской области. – Волгоград: ВолгГТУ, 2010.
21. *Шаповалов О.В., Юдкин А.А., Андреев А.Е.* Разработка параллельных решателей для систем многотельного моделирования // Информационные технологии в образовании, технике и медицине: Материалы Междунар. конф. – Волгоград: ВолгГТУ, 2009.

22. *Эндрюс Г.Р.* Основы многопоточного, параллельного и распределенного программирования: Пер. с англ. Г.Р. Эндрюс. – М.: Издательский дом «Вильямс», 2003. – 505 с.
23. *Prasad S., Bruce L. M.* A robust multi-classifier decision fusion framework for hyperspectral, multi-temporal classification // Proc. IEEE IGARSS. – 2008. – Vol. 2. – P. 273-276.
24. *Ranawana R.* Intelligent multi-classifier design methods for the classification of imbalanced data sets: Ph.D. dissertation. – Univ. of Oxford, Oxford, U.K., 2007. – P. 33-39.
25. *Changhong F., Fernandez S., Antonio R., Miguel A., Pascual C.* Real-Time Adaptive Multi-Classifier Multi-Resolution Visual Tracking Framework for Unmanned Aerial Vehicles // Research, Education and Development of Unmanned Aerial Systems. – 2010. – Vol. 2, Part 1. – P. 99-106.
26. The OpenMP® API specification for parallel programming [Электронный ресурс].–Режим доступа: https://openmp.org (дата обращения: 03.05.2015).
27. ThreadingBuildingBlocks [Электронный ресурс]. – Режим доступа: https://www.threadingbuildingblocks.org (дата обращения: 03.05.2015).

Статью рекомендовал к опубликованию д.т.н., профессор А.М. Белевцев.

Шаповалов Олег Владимирович – ВолгГТУ; e-mail: oshapovalov@mail.com; 400123, г. Волгоград, ул. Триумфальная, 24, кв. 8; тел.: 89199805609; кафедра ЭВМиС; соискатель.

Андреев Андрей Евгеньевич – e-mail: andan2005@yandex.ru; 400131, г. Волгоград, ул. Краснознаменская, 8, кв. 43; тел.: 88442248494; кафедра ЭВМиС; и.о. зав. кафедрой; к.т.н.; доцент.

Фоменков Сергей Алексеевич – e-mail: saf550@yandex.ru; 400048, г. Волгоград, ул. Елецкая, 4, кв. 77; тел.: 89173376626; кафедра САПРиПК; д.ф-м.н.; профессор.

Shapovalov Oleg Vladimirovich – VSTU; e-mail: oshapovalov@mail.com; 400123, Volgograd, Triumfalnaya street, 24-8; phone: +79199805609; the department of EVM and systems; postgraduate.

Andreev Andrey Evgenyevich – e-mail: andan2005@yandex.ru; 400131, Volgograd, Krasnoznamenskaya street, 8-43; phone: +78442248494; the department of EVM and systems; head of department; cand. of Eng. sc.; associate professor.

Fomenkov Sergey Alekseevich – e-mail: saf550@yandex.ru; 400048, Volgograd, Eleckaya street, 4-77; phone: +79173376626; the department of CADVSTU; dr. of phys.-math. sc.; professor.

УДК 681.003

А.М. Белевцев, В.А. Балыбердин, Г.П. Бендерский, А.А. Белевцев

АНАЛИЗ НАПРАВЛЕНИЙ РАЗВИТИЯ НАНО- И IT-ТЕХНОЛОГИЙ ДЛЯ ПОСТРОЕНИЯ СПЕЦИАЛИЗИРОВАННЫХ СЕТЕВЫХ КОММУНИКАЦИОННЫХ СИСТЕМ НОВОГО ПОКОЛЕНИЯ

Анализируются на основе зарубежных исследований и разработок общие тенденции формирования коммуникационных систем сетевой информационной инфраструктуры, заключающиеся в стремлении к созданию единого информационно-коммуникационного пространства на принципах сетецентризма и сервис-ориентированной архитектуры, что обеспечивает гибкость применения различных сервисов в рамках данного пространства и доступ пользователей к необходимой информации в любое время и в любом месте. Отмечается общая тенденция формирования коммуникационных систем сетевой информационной инфраструктуры, заключающаяся в стремлении к созданию единого информационно-коммуникационного пространства на принципах сетецентризма и сервис-ориентированной архитектуры, что обеспечивает гибкость применения различных сервисов в рамках данного пространства и доступ пользователей к необходимой информации в любое время и в любом месте. Отмечается, что новые вызовы обусловили потребность в новых подходах и развитии новых направлений исследований, что