УДК 681.325.518.5

ВЕКТОРИЗАЦИЯ ПРОЦЕССОВ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ СЕЙСМИЧЕСКИХ СИГНАЛОВ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ «PARALLEL STUDIO»

Зайнидинов Х.Н., Маллаев О.У.

Сейсмик ва геофизик сигналларни таҳлил қилишда ва қайта ишлашда сплайн усуллари кенг оммалашган. Сплайн усуллар ушбу сигналларга универсал ёндашув воситаси бўлиб ҳизмат қилади ҳамда бошқа математик усуллар билан таққослаганда ахборотларнинг теңглиги ва аппарат харажатларига эга бўлиш билан кўпроқ аниқликка эга. Бошқа томондан, бундай тизимларда ишлатиладиган аппарат воситалар ҳам юқори ишлаш тезлиги талабларига жавоб бериши керак. Юқори тезликка эришиш учун «**Parallel Studio**» технологияларни ишлатиш асосида параллел алгоритмларни ишлаб чиқилади ва уларни кўп ядроли протсессорларнинг архитектурасига тадбиқ қилинади. Ушбу мақола кўп ядроли протсессорлар архитектураси асосида «**Parallel Studio**» технологиясини ишлатган ҳолатда сигналларни рақамли ишлашнинг сплайн-усулларининг дастурий тадбиқини тавсия қилади. Бу ўрнатилган аниқлик кўрсаткичларида маълумотларни қайта ишлаш тезлигини ошириш ҳисобига тизимнинг ишлаш самарадорлигини кўтаришга имкон беради.

Таянч иборалар: Параллел алгоритм, оқим, кўп ядроли процессор, сплайн, B-сплайн, сейсмик сигнал, векторлаштириш.

Широкая популярность сплайн-методов в задачах анализа и обработки сейсмических и геофизических сигналов объясняется тем, что они служат универсальным инструментом приближения и по сравнению с другими математическими методами при равных с ними информационных и аппаратных затратах обеспечивают большую точность. С другой стороны, применяемые в таких системах аппаратные средства также должны отвечать требованиям высокой скорости обработки. Для достижения высокой скорости обработки необходимо разработать параллельные алгоритмы с использованием технологии «**Parallel Studio**» и реализовать их на многоядерных архитектурах процессоров. Статья предлагает программную реализацию сплайн-методов цифровой обработки сигналов с использованием технологии «**Parallel Studio**» на основе многоядерной архитектуры процессоров. Это позволяет в целом повысить эффективность функционирования систем за счет увеличения скорости обработки данных при установленных показателях точности.

Ключовые словы: Параллельный алгоритм, поток, многоядерный процессор, сплайн, B-сплайн, сейсмический сигнал, векторизация.

The wide popularity of spline methods in the problems of analysis and processing of seismic and geophysical signals is explained by the fact that they serve as a universal tool of approximation and in comparison with other mathematical methods at equal information and hardware costs provide greater accuracy[7-9]. On the other hand, the hardware used in such systems must also meet the requirements of high processing speed. In order to achieve high processing speed, it is necessary to develop parallel algorithms using "Parallel Studio" technology and implement them on multi-core processor architectures. The article offers software implementation of spline methods of digital signal processing using technology "Parallel Studio" based on multi-core architecture of processors. This allows to improve the overall efficiency of the system by increasing processing speed when the established indicators of accuracy. All the functionality of the tools Intel Parallel Studio seems to be a natural extension of Visual Studio and the expansion of its capabilities, which previously lacked to develop good parallel programs. There are two approaches to writing parallel programs. The first is parallelization, partially or completely, of already ready serial applications to speed up the work of some algorithms. In this case, the developer simply analyzes the application and determines the parts of the program that consume the maximum amount of processor resources. The second approach assumes the original design, taking into account the requirements of parallel execution of the load. And if in essence the project can be divided into sections that must be executed simultaneously, then starting its implementation in the form of a program is often a difficult task for beginners. It is especially difficult to write a project so that you do not have to resort to the first approach. Here the Parallel Advisor comes to the rescue. This is a class of tools that bears the methodology of creating parallel programs from scratch using the correct approaches to their implementation, including using parallel libraries. Intel Parallel Composer is not only Intel's C ++ compiler. It is integrated in Visual Studio with the Integrated Performance Primitives (IPP) and Parallel TBB (Threading Building Blocks), which greatly facilitates the development of parallel code. Normal engineering practice involves testing the program for errors and the developer himself, at least at the level of unit tests. Parallel Inspector helps to detect two classes of errors:

errors of multithreading and errors of work with memory, and the analysis for each class is launched separately. The last class of errors is well known to programmers who until recently used various tools to find memory leaks, a stack integrity violation or access to non-existent addresses. The second class of errors is related to the smoothed nature of the programs. The mechanism for detecting memory errors is based on the analysis of absolutely all read / write instructions and their addresses at the binary code level using binary tools.

The Performance Profiler is designed to find out how efficiently the application uses the multiprocessor platform, and where are the bottlenecks in the program that prevent it from scaling and increase performance with the growth of compute cores in the system.

Keywords: Parallel algorithm, flow, multi-core processor, spline, B-spline, seismic signal, vectorization.

I. ВВЕДЕНИЕ

В настоящее время разработчикам приложений необходимы мощные и удобные инструменты для адаптации существующих или написания новых приложений, максимально использующих производительность многоядерных процессоров. Понятно, что часто графический интерфейс приложений написан с использованием Java или .Net языков. Тем не менее, те части приложений, которые требовательны к производительности (фильтры, кодеки, и т.д.), в большинстве случаев реализованы именно на C/C++, и именно в них важно добиться задействования всех возможностей процессора[7-10]. Сейчас уже нет сомнений, что дальнейшее увеличение производительности приложений будет достигаться за счет того, насколько хорошо эти приложения распараллелены и как хорошо они масштабируются с ростом процессоров в системе. Очевидно, что фактическим стандартным набором C/C++ разработчика является Microsoft Visual Studio. Intel Parallel Studio – это набор из нескольких инструментов, который является расширением Microsoft Visual Studio, и позволяющий за счет удобства использования и понятного интерфейса добиваться хорошей эффективности параллельных программ на многоядерных системах [5,6].

II. ОСНОВНАЯ ЧАСТЬ

Пакет параллельной обработки Intel Parallel Studio.

Несмотря на то, что этот набор является plug-in'ом к Visual Studio, он полностью покрывает все этапы разработки приложения программистом, от создания скелета будущей параллельной программы до оптимизации релизной версии проекта (Рис.1). В состав этого набора входят четыре отдельных продукта, каждый их которых используется на определенном этапе разработки, и каждый может быть установлен и интегрирован в Visual Studio как отдельно, так и всем пакетом сразу.

Рис.1. Цикл разработки параллельной программы

В состав пакета входят:

Intel Parallel Advisor - помогает найти возможности распараллеливания кода с самого начала разработки приложения;

Intel Parallel Composer - предназначен для генерирования параллельного кода, т.е. создания программ с помощью компилятора и широкого набора библиотек для многопоточных алгоритмов;

Intel Parallel Inspector - проводит проверку параллельного приложения на корректность и находит ошибки работы с памятью;

Intel Parallel Amplifier - находит «узкие места» в программе, которые мешают масштабируемости и увеличению производительности на многоядерных платформах.

Профилировщик производительности предназначен для того, чтобы выяснить, насколько эффективно используется приложением многопроцессорная платформа, и где находятся те узкие места в программе, которые мешают ей масштабироваться и увеличивать производительность с ростом вычислительных ядер в системе. Методология профилировки приложения предельно проста: необходимо ответить себе на три основных вопроса, каждый из которых соответствует своему типу анализа и отражает суть, место и причины проблем с производительностью.

Hotspot-анализ. «На что программа тратит вычислительное время процессора?» Необходимо знать те места в программе, Hotspot-функции, где больше всего тратится вычислительных ресурсов при исполнении, а также стек вызовов (тот путь, по которому мы в эти места попали).

Concurrency-анализ. «Почему программа плохо параллелится?» Бывает так, что независимо от того, насколько продвинута параллельная

структура приложения, ожидаемый прирост производительности при переходе, например, от 4-ядерной системе к 8-ядерной так и не достигается. Поэтому тут нужна оценка эффективности параллельного кода, которая дала бы представление о том, насколько полно используются ресурсы процессора.

Lock & Wait - анализ. «Где программа простаивает в ожидании синхронизации или операции ввода-вывода?» Поняв, что программа плохо масштабируется, хочется найти, где и какие именно объекты синхронизации встали на пути к хорошей параллельности. Возможно, необходимо пересмотреть реализацию алгоритмов, а может, и всю параллельную структуру приложения[4,10].

Каждый из этих видов анализа запускается по отдельности и имеет собственное окно представления результатов. Наличие встроенного функционала сравнения результатов позволяет отслеживать влияние изменения кода программы на ее производительность.

Параллельные алгоритмы обработки сейсмических сигналов на основе сплайнов. В технических приложениях наиболее употребительными являются сплайны невысокой степени, в частности параболические и кубические [1–4]. Процесс построения таких сплайнов значительно проще, чем процесс построения сплайнов более высокой степени. Значительно упрощаются вычислительные проблемы при обращении к локальной сплайн - аппроксимации, в которых значения приближающей сплайн - функции на каждом отрезке зависят только от значений аппроксимируемой функции из некоторой окрестности этого отрезка. Другой особенностью таких методов является то, что они не требуют решения систем уравнений при нахождении параметров сплайна.

Любой сплайн Sm(x) степени m дефекта 1, интерполирующий заданную функцию f(x), может быть единственным образом представлен B-сплайнами в виде суммы [2, 5, 10]:

$$f(x) \cong S_m(x) = \sum_{i=-1}^{m+1} b_i \cdot B_i(x), \quad a \le x \le b \qquad (1)$$

где bi – коэффициенты, $B_i(x)$ - базисный сплайн

В случае, когда используется кубический базисный сплайн, то его значения вычисляются по формуле:

$$B_3 = \begin{cases} 0, x \ge 2, \\ (2-x)^3/6, 1 \le x < 2, \\ 1/6(1+3(1-x)+3(1-x)^2-3(1-x)^3), 0 \le x < 1, \end{cases} \qquad (2)$$

а коэффициенты можно определять по следующей формуле:

$$b_i = (1/6)(-f_{i-1} + 8f_i - f_{i+1}); \qquad (3)$$

Согласно формуле (1) значение интерполируемой функции в произвольной точке заданного интервала определяется значениями лишь m+1 слагаемых – парных произведений базисных функций на постоянные коэффициенты. Например, кубические B-сплайны требуют четырех базисных слагаемых. Значение функции вычисляется по формуле:

$$f(x) \cong S_3(x) = b_{-1}B_{-1}(x) + b_0B_0(x) + b_1B_1(x) + b_2B_2(x) \text{ при } x \in [0,1] \quad (4)$$

Остальные базисные сплайны на этом подинтервале равны нулю и, следовательно, в образовании суммы не участвуют.

Если использовать один основной базисный сплайн и с помощью переменной j задать адреса разных участков основного сплайна, то уравнение (4) принимает вид:

$$S_3[i] = (b[i-1]\,B[j+30]) + b[i]\,B[j+20] + b[i+1]\,B[j+10]) + b[i+2]\,B[j]) \quad (5)$$

Для разработки параллельного алгоритма выделим на потоки [4, 5, 6,]. Для выполнения четырех параллельных умножений выделим m=4 потока. В результате получим формулу (6):

$$S_j(L_j : K_j : P_j : T_j) = \begin{cases} L_j = \sum_{i=-1}^{m+1} b_{i-1}B_{i+30}(x); \\ K_j = \sum_{i=-1}^{m+1} b_iB_{i+20}(x); \\ P_j = \sum_{i=-1}^{m+1} b_{i+1}B_{i+10}(x); \\ T_j = \sum_{i=-1}^{m+1} b_{i+2}B_{i+2}(x); \end{cases} \quad (6)$$

где Lj, Kj, Pj и Tj - потоки умножения матрицы на вектор.

Блок-схема алгоритма параллельного вычисления с выделением потоков с применением пакета OpenMP приведена на рис. 2.

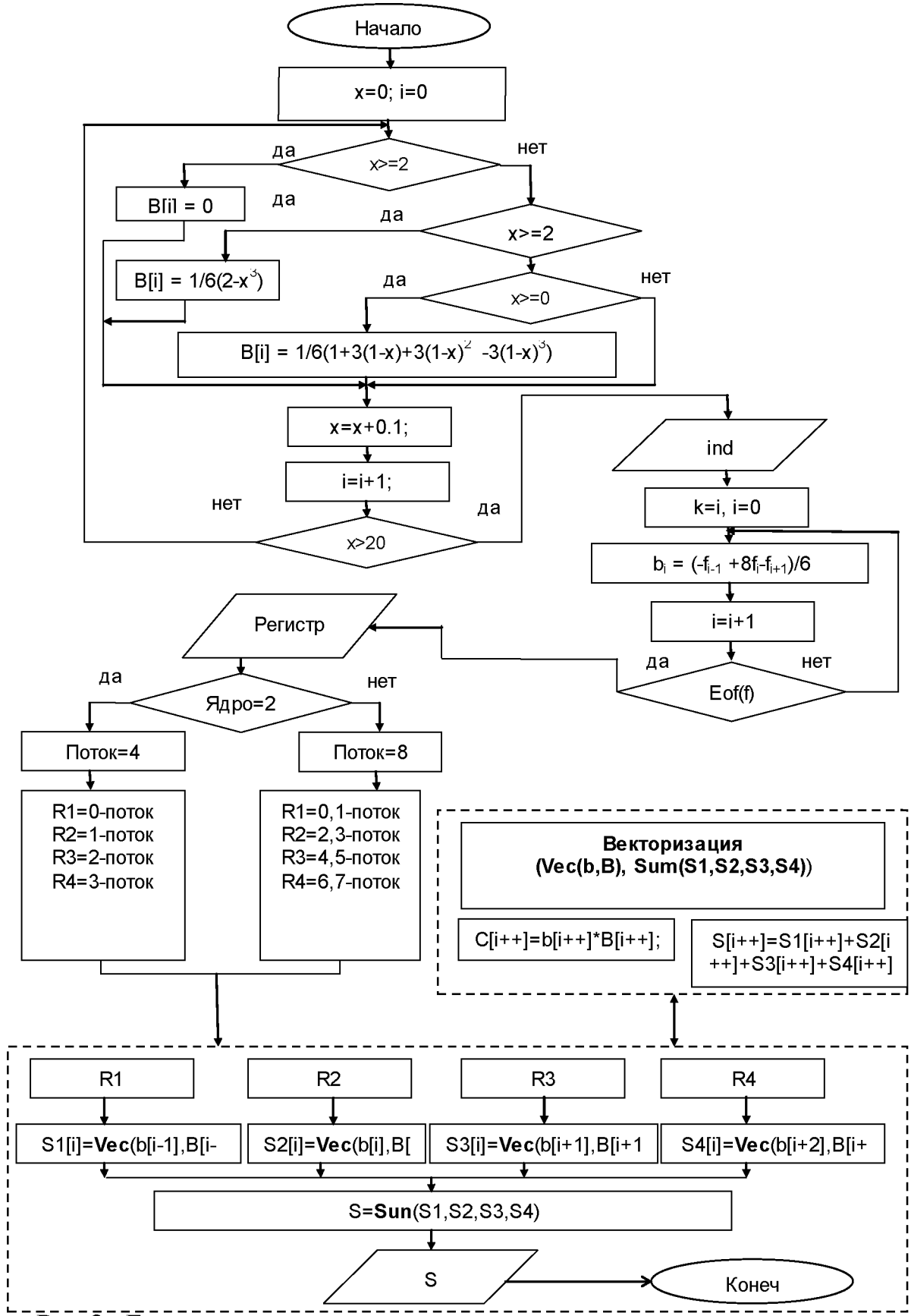

Рис.2. Блок-схема алгоритма векторизации параллельного вычисления с выделением потоков.

Векторизация параллельных процессов на основе Parallel Studio. Векторизация — вид распараллеливания программы, при котором однопоточные приложения, выполняющие одну операцию в каждый момент времени, модифицируются для выполнения нескольких однотипных операций одновременно, векторных операций. Методы векторизации выбираются на основе типа микропроцессора.

Например, Векторный процессор **(Vector processor)** –это процессор, поддерживающий на уровне системы команд операции для работы с одномерными массивами – векторами (vector).

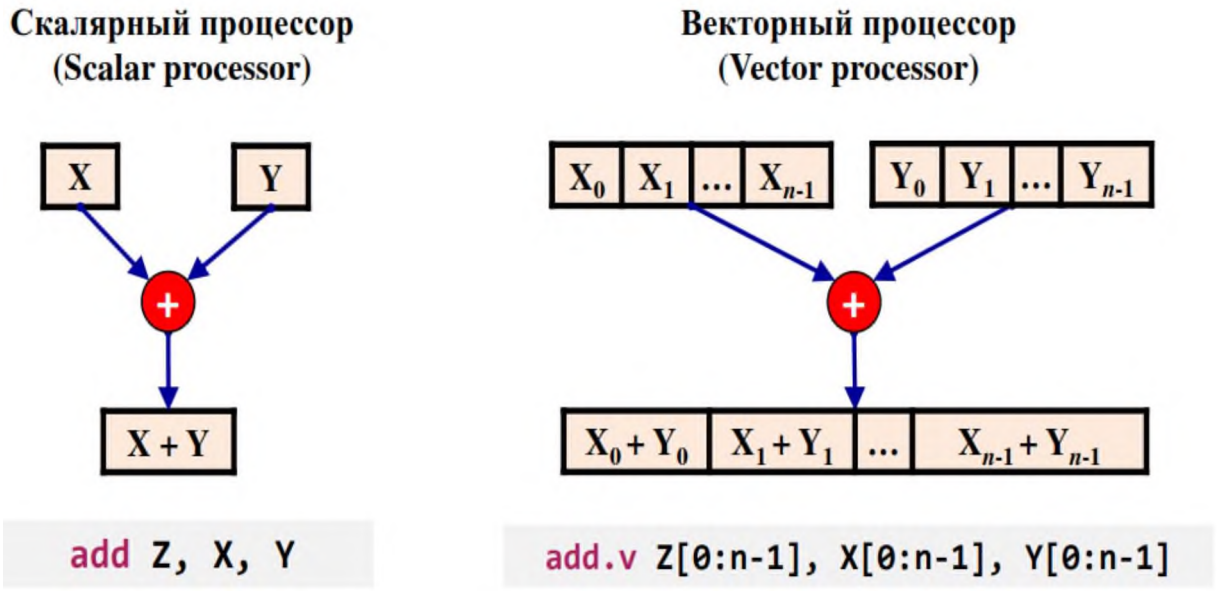

Рис.3. Режим работы скалярных и векторных процессоров.

Во многих вычислительных программах одна и та же операция применяется к целому набору данных: к массиву, ну либо к его какому-то кусочку. В этом случае наблюдается повторяющаяся модель доступа к данным, а также повторяющаяся операция, которую можно выполнять параллельно. Например, в следующем фрагменте кода показан пример такой повторяющейся модели. В данном примере происходит сложение двух векторов.

Обычный скалярный процессор будет выполнять данную программу следующим образом: сначала загружаются в регистр числа A(0) и B(0), затем они складываются и результат записывается в C(0), далее A увеличивается на 1, в регистры загружаются элементы A1 и B1, и выполняется операция сложения и так далее.

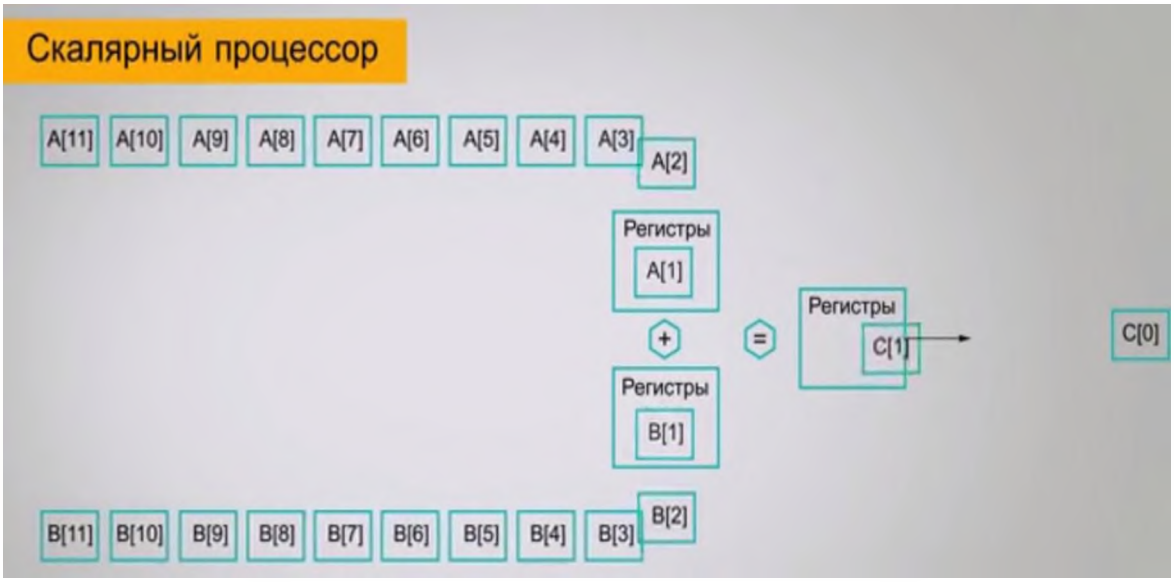

Рис.4. Режим программирования обычных скалярных процессоров

Рис.5. Режим программирования векторных процессоров.

Если же процессор векторный, то тогда в нём есть векторные регистры, которые позволяют хранить вектор значений, и в процессоре реализованы векторные инструкции. Если воспользоваться векторными инструкциями, то процессор в регистры загрузит сразу несколько значений вектора A и B. Предположим 4 элемента. Тогда в регистрах будут элементы: A(0), A(1), A(2), A(3); и соответственно для вектора B: B(0), B(1), B(2) и B(3). И для всех 4 пар одновременно выполнится операция сложения, а результат запишется в C(0), C(1), C(2) и C(3).

Преимущество векторизации в том, что время выполнения векторной операции такое же, как и скалярной, но полезной работы выполняется больше. Одна векторная команда распознаётся, декодируется и выполняется быстрее нескольких скалярных, выполняющих тоже действий.

На самом деле, все современные процессоры содержат в себе векторные инструкции, поэтому векторную обработку данных нужно применять как на суперкомпьютерах, так и на обычных компьютерах. С самого начала развития компьютерной техники велись попытки архитектурно решить задачу ускорения вычислений, то есть создания параллельной обработки данных.

Для ручной векторизации потребовалось приложить достаточное количество усилий, а для автовекторизации достаточно воспользоваться опцией компилятора. Каким бы умным ни был компилятор, — а с каждым годом они становятся все умней и умней, — существует множество случаев, когда компилятор не в состоянии векторизовать код. В этом случае мы должны помочь компилятору с помощью определенных подсказок. Мы должны сообщить ему дополнительную информацию об алгоритме или о данных, и потом он сможет векторизовать этот код уже сам. Чаще такими подсказками являются специальные директивы. Например, директива #pragma simd входит в набор инструментов Cilk Plus. Она отключает все проверки компилятора о возможности векторизовать и сообщает компилятору, чтобы тот целиком полагался на то, что говорит разработчик, и просто векторизует вычисления. Ответственность в этом случае перекладывается на разработчика. Если вдруг будет зависимость по данным или еще какие-то условия не будут выполнены, не позволяющие векторизовать код, то программа будет работать неправильно.

Результаты экспериментов. Для проверки разработанного алгоритма были обработаны сейсмические сигналы разной длины N. Алгоритм был реализован в двухядерном процессоре Intel(R) Core(TM) i5-2410M CPU @ 2.30 GHz. Полученные результаты приведены в Таблице 1.

Таблица 1. Сравнительные данные, полученные в результате обработки сейсмических сигналов в двухядерном процессоре

Количество отсчетов входного сигнала N	**Parallel Studio (сек.)**	**Visual Studio (сек.)**	**Коэффициент ускорения**
1024	0,00001	0,00002	2
2048	0,000011	0,00003	2,7
4096	0,000073	0,000191	2,6
16000	0,000023	0,000078	3,4

В соответствии с данными таблицы 1 для векторизация процессов параллельной обработки сейсмического сигнала при количестве отсчетов N=1024 требовалось $0,02x10^{-4}$ секунд в обычном (Visual Studio) компиляторе, $0,01x10^{-4}$ секунд (после векторизации) в Parallel Studio.

С увеличением количества входных отсчетов N параллельная часть обработки увеличивается и растёт коэффициент ускорения. Так при N=16000 для векторизация процессов параллельной обработки сейсмического сигнала требовалось $0,078x10^{-4}$ секунд в обычном (Visual Studio) компилятор, $0,023x10^{-4}$ секунд в Parallel Studio.

ЗАКЛЮЧЕНИЕ

Таким образом, требования высокой производительности вычислительных систем, применяемых в задачах обработки и восстановления сейсмических данных, могут быть удовлетворены как за счет разработки новых методов и параллельных алгоритмов и программ обработки, так и с помощью многоядерных средств параллельных вычислений. Практическая реализация сплайн-методов в виде **векторизации процессов** параллельной обработки сейсмических данных **с использованием технологии «parallel studio»** и многоядерной архитектуры процессоров в процедурах прогнозирования, интерполирования, сглаживания и идентификации, восстановления и сокращения избыточности позволяет в целом повысить эффективность систем за счет увеличения скорости обработки данных при установленных показателях точности.

ЛИТЕРАТУРА

[1] Завьялов Ю.С. Леус В.А., Скороспелов В.А. Сплайны в инженерной геометрии. - М.: Машиностр., 1985.- 224 с.

[2] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб: «БХВ-Петербург», 2012. – 608 с.

[3] Богачев К.Ю. Основы параллельного программирования. – М.: «БИНОМ. Лаборатория знаний», 2013. – 342 с.

[4] Х.Н. Зайнидинов, Г.О. Тожибоев, О.У. Маллаев. Параллельные алгоритмы обработки сейсмических сигналов на многоядерных процессорах. Автоматика и программная инженерия. 2018. № 1 (23). С. 89–95.

[5] Герберт Р., БикА., Смит К., ТианК. **Оптимизация ПО. Сборник рецептов**. -СПб: Питер, 2010. -352 с.

[6] Randal E. Bryant, David R. O'Hallaron. **Computer Systems: A Programmer's Perspective.**-Addison-Wesley, 2010

[7] AgnerFog. **The microarchitecture of Intel, AMD and VIA CPUs** (an optimization guide for assembly programmers and compiler makers) // http://www.agner.org/optimize/microarchitecture.pdf

[8] Michael E. Thomadakis. **The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms**// http://sc.tamu.edu/systems/eos/nehalem.pdf

[9] **Intel® 64 and IA-32 Architectures Optimization Reference Manual**// http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf

[10] Intel Parallel Studio home page http://software.intel.com/en-us/forums/intel-parallel-studio