Разработка параллельных алгоритмов в прикладной гидрофизической задаче с использованием современных средств профилирования кода

**Человек:** В данной статье автор рассматривает разработку параллельного алгоритма для ускорения вычислений в прикладной научной задаче. Особое внимание уделяется программным инструментам для профилирования кода. Применение таких инструментов позволяет быстро выделить участки кода для распараллеливания. Рассматриваются библиотеки и технологии реализации параллельного кода для многопроцессорных систем с общей памятью. Обсуждаются подходы к написанию программ на основе потоков и на основе задач. Рассматриваются этапы разработки параллельного кода путем изменения последовательного кода. Приводится анализ времени выполнения функций приложения при помощи инструмента VTune. Предлагается способ распараллеливания по задачам на основе библиотеки TBB. Показана реализация кода позволяющая выполнять параллельные расчеты. В результате данной работы был разработан новый параллельный алгоритм для решения задачи нахождения характеристик внутренних волн в океане в рамках слабонелинейной теории. Полученный алгоритм проанализирован, получены метрики эффективности. Достигнуто ускорение позволяющее производить расчеты карт двумерных распределений в 39 раз быстрее последовательного алгоритма на 56 ядерном вычислительном сервере. Полученные результаты будут применены в дальнейшем при изучении внутренних волн в океане, а также позволят увеличить эффективность исследований при расчете карт параметров большего масштаба.

**Key words:** численная модель, уравнение Гарднера, численное моделирование, внутренние волны, многопроцессорные системы, профилировщик кода, параллельные технологии, анализ производительности, параллельные алгоритмы, гидрологические данные

=================================

**FastText\_KMeans\_Clean:** Такой подход позволяет ускорить вычисления путем деления расчетного алгоритма на подзадачи, выполняемые параллельно, что позволяет задействовать все аппаратные ресурсы в многоядерных процессорах и сократить время расчета. Где n является смещением изопикнической поверхности в максимуме вертикальной моды; c – фазовая скорость внутренних волн; a – коэффициент квадратичной нелинейности; a1 – коэффициент кубической нелинейности; b – коэффициент дисперсии. igwdatamodel::IDataProfile::Ptr salProfile = salProfileIt->get();. tbb::mutex readMutex;.

**Key words part:** 0.4137931034482759

=================================

**FastText\_KMeans\_Raw/:** igwdatamodel::IDataProfile::Ptr salProfile = salProfileIt->get();. Анализ стека этой функции показывает, что основное время состоит из вызовов методов интегрирования системы дифференциальных уравнений, реализованной в библиотеке boost - integrate\_adaptive (61.6%) и (22.9%). CalcTask(DataHolder\* dataHolder):. Profile data = m\_holder-> getNextProfile();.

**Key words part:** 0.4137931034482759

=================================

**FastText\_PageRank\_Clean/:** 1.1.Загрузить данные солености, температуры, глубины. 1.2.Рассчитать параметры плотности. 1.3.Рассчитать коэффициенты. DataHolder\* m\_holder;. m\_holder(dataHolder){}. tbb::task::recycle\_as\_continuation();. tbb::task\_list list;. tbb::task::spawn\_root\_and\_wait(list);.

**Key words part:** 0.3448275862068966

=================================

**FastText\_PageRank\_Raw/:** 1.3.Рассчитать коэффициенты. tbb::mutex writeMutex;. DataHolder\* m\_holder;. m\_holder(dataHolder){}. tbb::task\* execute() {. tbb::task::recycle\_as\_continuation();. tbb::task\_scheduler\_init init;. tbb::task\_list list;.

**Key words part:** 0.3103448275862069

=================================

**Mixed\_ML\_TR/:** Другой подход к распараллеливанию заключается в разделении алгоритма на множество логических подзадач, которые выполняются в потоках, но количество потоков и количество задач регулируется планировщиком. На основе карт двумерных распределений параметров уравнения Гарднера можно проводить моделирование распространения волн. igwdatamodel::IDataProfileIterator::Ptr tempProfileIt,salProfileIt;. tbb::task::recycle\_as\_continuation();. На рис. 3 приведен анализ ускорения параллельного алгоритма при расчете на 2348 точках исходного набора данных. Выполнен анализ последовательного кода с помощью программного продукта VTune. Выполнена реализация параллельного алгоритма, с использованием библиотек TBB, на основе разделения алгоритма на задачи.

**Key words part:** 0.6551724137931034

=================================

**MultiLingual\_KMeans/:** Другой подход к распараллеливанию заключается в разделении алгоритма на множество логических подзадач, которые выполняются в потоках, но количество потоков и количество задач регулируется планировщиком. На основе карт двумерных распределений параметров уравнения Гарднера можно проводить моделирование распространения волн. igwdatamodel::IDataProfileIterator::Ptr tempProfileIt,salProfileIt;. На рис. 3 приведен анализ ускорения параллельного алгоритма при расчете на 2348 точках исходного набора данных. Выполнен анализ последовательного кода с помощью программного продукта VTune.

**Key words part:** 0.6551724137931034

=================================

**Multilingual\_PageRank/:** Рассмотрим задачу численного моделирования внутренних гравитационных волн в океане в рамках слабонелинейной теории. Данный тип волн возникает в океане вследствие вертикальной неоднородности воды по температуре и солёности. В частности, такие данные могут быть получены путем натурных измерений температуры и солёности в океане с помощью контактных датчиков подводного зондирования, опускаемых с корабля, либо могут быть рассчитаны на основе математических моделей. Эти данные распространяются на свободной основе организациями, которые занимаются исследованием климата планеты. Поэтому увеличение задач не приведет к более эффективному распараллеливанию. Полученное ускорение приведено на рис. 3. В реальности (синяя кривая) практическое ускорение почти никогда не совпадает с идеальным, так как расчеты зачастую связаны с сохранением и загрузкой данных, а также синхронизацией, которые вносят накладные расходы процессорного времени. Время расчета сократилось в 39 раз и составило около 165 секунд против 6437 секунд последовательного алгоритма.

**Key words part:** 0.5862068965517241

=================================

**RuBERT\_KMeans\_Without\_ST/:** Выбор связан с тем, что в библиотеке реализована эффективная поддержка планировщика и распараллеливание на основе задач. 1.3.1. Методом пристрелки найти решение краевой задачи Штурма-Лиувилля. igwdatamodel::IDataProfile::Ptr tempProfile = tempProfileIt->get();. Анализ стека этой функции показывает, что основное время состоит из вызовов методов интегрирования системы дифференциальных уравнений, реализованной в библиотеке boost - integrate\_adaptive (61.6%) и (22.9%).

**Key words part:** 0.4137931034482759

=================================

**RuBERT\_KMeans\_With\_ST/:** Он содержит набор приложений для профилировки, имеет пользовательский интерфейс и не требует вставки директив в исходный код для анализа времени выполнения участков кода. tempProfileIt->next();. Анализ стека этой функции показывает, что основное время состоит из вызовов методов интегрирования системы дифференциальных уравнений, реализованной в библиотеке boost - integrate\_adaptive (61.6%) и (22.9%). Поэтому распараллеливание целесообразно провести по данным на основе независимости расчетов коэффициентов в каждой отдельно взятой географической точке. tbb::mutex::scoped\_lock lock(writeMutex);.

**Key words part:** 0.4482758620689655

=================================

**RUBERT\_page\_rank\_Without\_ST/:** Следует также отметить наличие выстроенных средств параллелизма в новых стандартах языков программирования. Это приводит к образованию в толще жидкости слоев различной плотности, на границах которых и образуются данные волны. DataHolder, листинг 2. while(!tempProfileIt->atEnd() && !salProfileIt->atEnd()). DataHolder holder(temperature,.

**Key words part:** 0.3793103448275862

=================================

**RUBERT\_page\_rank\_With\_ST/:** 1. Для каждой географической точки карты. 1.3.Рассчитать коэффициенты. DataHolder, листинг 2. DataHolder\* m\_holder;. Анализ достигнутой производительности.

**Key words part:** 0.3793103448275862

=================================

**RUSBERT\_KMeans\_Without\_ST/:** Такой подход позволяет ускорить вычисления путем деления расчетного алгоритма на подзадачи, выполняемые параллельно, что позволяет задействовать все аппаратные ресурсы в многоядерных процессорах и сократить время расчета. igwdatamodel::IDataProfile::Ptr tempProfile = tempProfileIt->get();. m\_holder->setResult(result);. Число задач может быть и больше количества потоков, так как планировщик будет выполнять балансировку нагрузки автоматически. В статье были рассмотрены современные средства профилировки, анализа и распараллеливания кода.

**Key words part:** 0.4137931034482759

=================================

**RUSBERT\_KMeans\_With\_ST/:** В частности, такие данные могут быть получены путем натурных измерений температуры и солёности в океане с помощью контактных датчиков подводного зондирования, опускаемых с корабля, либо могут быть рассчитаны на основе математических моделей. igwdatamodel::IDataWriter::Ptr dst). CalcTask& t = \*new(tbb::task::allocate\_root()) CalcTask(&holder);. Запуск параллельных задач осуществляется в новой функции calcCoeffsParallel. Число задач может быть и больше количества потоков, так как планировщик будет выполнять балансировку нагрузки автоматически. Выполнена реализация параллельного алгоритма, с использованием библиотек TBB, на основе разделения алгоритма на задачи.

**Key words part:** 0.4827586206896552

=================================

**RUSBERT\_page\_rank\_Without\_ST/:** tempProfileIt->next();. salProfileIt->next();. DataHolder, листинг 2. tempProfileIt->next();. salProfileIt->next();.

**Key words part:** 0.3103448275862069

=================================

**RUSBERT\_page\_rank\_With\_ST/:** 1.2.Рассчитать параметры плотности. tempProfileIt->next();. DataHolder, листинг 2. tempProfileIt->next();. salProfileIt->next();.

**Key words part:** 0.3103448275862069

=================================

**Simple\_PageRank/:** После анализа кода распараллеливание может быть выполнено с применением различных библиотек и технологий: OpenMP [7,8], TBB [9], QtConcurrent [10]. Параметры c , a, a1 и b зависят от стратификации плотности и глубины океана. Как видно из анализа большая часть времени выполнения (99.8%) принадлежит функции deriveddatalib::coeffs. Анализ стека этой функции показывает, что основное время состоит из вызовов методов интегрирования системы дифференциальных уравнений, реализованной в библиотеке boost - integrate\_adaptive (61.6%) и (22.9%). Данный класс содержит в себе указатели на контейнеры входных данных температуры и солености (tempProfileIt, salProfileIt), и контейнер для записи данных (dst). Расчеты проводились на сервере со следующей конфигурацией: 2 процессора Intel Xeon CPU E5-2650 v4 2.20Ггц, 64 Гб оперативной памяти, 512 Гб SSD диск Samsung Evo 850.

**Key words part:** 0.5172413793103449

=================================

**TextRank/:** Другой подход к распараллеливанию заключается в разделении алгоритма на множество логических подзадач, которые выполняются в потоках, но количество потоков и количество задач регулируется планировщиком. Но в данном случае задачи при старте и окончании расчета выполняют доступ к данным и время доступа мало. В таком сценарии создание количества задач большее, чем число физических потоков целесообразно, так как заполненные данными для расчета задачи будут ожидать в очереди выполнения и начинать расчет сразу в то время пока данные из выполненных задач будут сохраняться. Анализ стека вызовов и времени выполнение показывает, что основное время работы приложения находится в расчетных задачах. На рис. 3 приведен анализ ускорения параллельного алгоритма при расчете на 2348 точках исходного набора данных. Выполнена реализация параллельного алгоритма, с использованием библиотек TBB, на основе разделения алгоритма на задачи.

**Key words part:** 0.4827586206896552

=================================

**TF-IDF\_KMeans/:** Распараллеливание на основе потоков приводит к тому, что логические потоки отображаются на физические потоки вычислительного оборудования. Таким образом расчет карт данных параметров является необходимым этапом численного эксперимента. Рассмотрим пример анализа кода с помощью профилировщика VTune. igwdatamodel::IDataWriter::Ptr dst). salProfileIt->next();. DataHolder\* m\_holder;. tbb::task::recycle\_as\_continuation();. Реализация параллельного запуска задач представлена в листинге 4.

**Key words part:** 0.5862068965517241

=================================

**Текст:** При разработке прикладных программ для решения задач в области математического моделирования целесообразно применение параллельных алгоритмов. Такой подход позволяет ускорить вычисления путем деления расчетного алгоритма на подзадачи, выполняемые параллельно, что позволяет задействовать все аппаратные ресурсы в многоядерных процессорах и сократить время расчета. При проведении длительных численных экспериментов время расчета является важным фактором. Чем меньше время расчета, тем большее количество экспериментов можно провести на тех же аппаратных ресурсах.. В настоящее время для разработки параллельных алгоритмов существует множество программных продуктов, так называемых профилировщиков приложений. Они помогают ускорить процесс анализа «узких» мест в коде и выделить наиболее ресурсоемкие участки, а также провести анализ производительности распараллеленных программ. Такой подход позволяет в короткие сроки разработать параллельный алгоритм и проанализировать взаимодействие параллельных потоков.. Среди профилировщиков кода для многоядерных систем с общей памятью можно выделить следующие программные продукты: Valgrind [1], OpenSpeedShop [2], ompP [3], TAU [4], VTune [5], MAQAO [6] и др. Некоторые из них требуют предварительной подготовки кода для анализа приложений, путем вставки специальных директив или вызовов функций внешней библиотеки. Кроме того, не все из них имеют пользовательский интерфейс, а выполняют только профилировку, визуализация и анализ результатов может выполняться в другом приложении, поддерживающем формат вывода. После анализа доступных профилировщиков выбор был сделан в пользу программного продукта VTune. Он содержит набор приложений для профилировки, имеет пользовательский интерфейс и не требует вставки директив в исходный код для анализа времени выполнения участков кода.. После анализа кода распараллеливание может быть выполнено с применением различных библиотек и технологий: OpenMP [7,8], TBB [9], QtConcurrent [10]. Стандарт OpenMP базируется на включении директив препроцессора в исходный код, и поддерживается большинством компиляторов. Следует также отметить наличие выстроенных средств параллелизма в новых стандартах языков программирования. Например, в C++ версии 11 появилась поддержка потоков в библиотеке STL (std::threads), которую также поддерживает большинство современных компиляторов.. Распараллеливание на основе потоков приводит к тому, что логические потоки отображаются на физические потоки вычислительного оборудования. Наибольшая эффективность вычислений достигается, когда на один физический поток приходится один логический поток. Если на один физический поток приходится более одного логического, то происходит переключение контекста операционной системой, что приводит к накладным расходам. Затраты времени на переключение контекста связаны с тем, что поток имеет свою собственную копию ресурсов, таких как состояние регистра и стек. Создание и удаление потока также приводит к значительным накладным расходам. Другой подход к распараллеливанию заключается в разделении алгоритма на множество логических подзадач, которые выполняются в потоках, но количество потоков и количество задач регулируется планировщиком. В программировании на основе задач планировщик выполняет балансировку нагрузки. Помимо использования нужного количества потоков, важно равномерно распределять работу между этими потоками. При программировании на основе потоков часто возникает проблема балансировки нагрузки.. Для решения задачи распараллеливания была выбрана библиотека TBB. Выбор связан с тем, что в библиотеке реализована эффективная поддержка планировщика и распараллеливание на основе задач.. Постановка задачи. Рассмотрим задачу численного моделирования внутренних гравитационных волн в океане в рамках слабонелинейной теории. Данный тип волн возникает в океане вследствие вертикальной неоднородности воды по температуре и солёности. Это приводит к образованию в толще жидкости слоев различной плотности, на границах которых и образуются данные волны. Рассмотрим классическое уравнение для описания распространения внутренних волн – уравнение Гарднера [11]. . Где n является смещением изопикнической поверхности в максимуме вертикальной моды; c – фазовая скорость внутренних волн; a – коэффициент квадратичной нелинейности; a1 – коэффициент кубической нелинейности; b – коэффициент дисперсии. Параметры c , a, a1 и b зависят от стратификации плотности и глубины океана.. В частности, такие данные могут быть получены путем натурных измерений температуры и солёности в океане с помощью контактных датчиков подводного зондирования, опускаемых с корабля, либо могут быть рассчитаны на основе математических моделей. Эти данные распространяются на свободной основе организациями, которые занимаются исследованием климата планеты. Для расчета будем использовать набор данных RCO полученный на базе модели глобальной циркуляции [12].. На основе карт двумерных распределений параметров уравнения Гарднера можно проводить моделирование распространения волн. Таким образом расчет карт данных параметров является необходимым этапом численного эксперимента. Подробное изложение математической теории расчета данных параметров представлено в работах [11, 13]. В статье ограничимся рассмотрением программной реализации алгоритмов расчета.. Основной алгоритм расчета карт параметров:. . 1. Для каждой географической точки карты. 1.1.Загрузить данные солености, температуры, глубины. 1.2.Рассчитать параметры плотности. 1.3.Рассчитать коэффициенты. 1.3.1. Методом пристрелки найти решение краевой задачи Штурма-Лиувилля. 1.4.Сохранить коэффициенты в контейнер данных. . Анализ кода и разработка параллельного алгоритма. . Рассмотрим пример анализа кода с помощью профилировщика VTune. Для этого необходимо собрать исследуемое приложение в конфигурации Release с включением символов отладки (параметры компилятора /Zi /DEBUG для компилятора MSVC). Затем необходимо выполнить HotSpot анализ путем запуска приложения через профилировщик. Данный тип анализа собирает информацию о времени выполнения кода предоставляя распределение в процентах по функциям и строкам кода, рис. 1.. . . Рис. 1 Анализ последовательного алгоритма профилировщиком VTune. . Интересуемый участок расчётного кода представлен в листинге 1.. Листинг 1.. void calcCoeffs(igwdatamodel::IDataProfileIterator::Ptr tempProfileIt,. igwdatamodel::IDataProfileIterator::Ptr salProfileIt,. igwdatamodel::IDataWriter::Ptr dst). {. while(!tempProfileIt->atEnd() && !salProfileIt->atEnd()). {. igwdatamodel::IDataProfile::Ptr tempProfile = tempProfileIt->get();. igwdatamodel::IDataProfile::Ptr salProfile = salProfileIt->get();. Coeffs result = coeffs(getZ(tempProfile),. getBvf(tempProfile, salProfile),. getDepth(tempProfileIt));. dst->write(result);. tempProfileIt->next();. salProfileIt->next();. }. }. . Как видно из анализа большая часть времени выполнения (99.8%) принадлежит функции deriveddatalib::coeffs. Анализ стека этой функции показывает, что основное время состоит из вызовов методов интегрирования системы дифференциальных уравнений, реализованной в библиотеке boost - integrate\_adaptive (61.6%) и (22.9%). Данные методы не могут быть распараллелены так как шаги интегрирования имеют зависимость между предыдущим и следующим. Поэтому распараллеливание целесообразно провести по данным на основе независимости расчетов коэффициентов в каждой отдельно взятой географической точке. Таким образом логично поместить множественные вызовы функции deriveddatalib::coeffs в отдельные задачи.. Для этого введем класс доступа к исходным данным и записи результатов расчета. DataHolder, листинг 2.. Листинг 2.. . class DataHolder{. igwdatamodel::IDataWriter::Ptr dst;. igwdatamodel::IDataProfileIterator::Ptr tempProfileIt,salProfileIt;. tbb::mutex readMutex;. tbb::mutex writeMutex;. . public:. Profile getNextProfile(){. tbb::mutex::scoped\_lock lock(readMutex);. Profile profile;. while(!tempProfileIt->atEnd() && !salProfileIt->atEnd()). {. profile = getProfile(tempProfileIt->get(),salProfileIt->get()). tempProfileIt->next();. salProfileIt->next();. return profile;. }. return Profile();. }. void setResult(const Coeffs& result){. tbb::mutex::scoped\_lock lock(writeMutex);. dst.write(result);. }. };. . Данный класс содержит в себе указатели на контейнеры входных данных температуры и солености (tempProfileIt, salProfileIt), и контейнер для записи данных (dst). Так как контейнеры не имеют атомарных средств доступа то необходимо синхронизировать вызов этих функций так чтобы две задачи не могли одновременно выполнить их вызов. В функциях чтения (getNextProfile) и записи (setResult) реализован вызов синхронизационных примитивов типа мьютекс ( tbb::mutex::scoped\_lock lock(writeMutex)). Библиотека TBB содержит специальный класс обертку для реализации блокировки мьютекса (tbb::mutex::scoped\_lock), позволяющий заблокировать мьютекс при создании объекта обертки и освободить при разрушении.. Выполнение расчета характеристик в точке перенесено в класс CalcTask, наследуемый от tbb::task. Данный класс реализует разделение работ по задачам, листинг 3.. Листинг 3.. . class CalcTask: public tbb::task{. DataHolder\* m\_holder;. public:. CalcTask(DataHolder\* dataHolder):. m\_holder(dataHolder){}. . tbb::task\* execute() {. Profile data = m\_holder-> getNextProfile();. if(data.valid){. Coeffs result = coeffs(data.z, data.bvf, data.depth);. m\_holder->setResult(result);. tbb::task::recycle\_as\_continuation();. return this;. }. return nullptr;. }. };. . Класс CalcTask принимает на вход указатель на объект доступа к данным и переопределяет виртуальную функцию родительского класса execute, тело которой определяет код, выполняемый параллельно.. Вызов функции tbb::task::recycle\_as\_continuation() помечает задачу как задачу для повторного выполнения. Таким образом, задача будет перезапущена планировщиком повторно. Данный подход позволяет использовать пул задач и переиспользовать без удаления старых и создания новых задач, тем самым минимизируя накладные расходы, связанные с созданием и удалением объектов. Реализация параллельного запуска задач представлена в листинге 4.. Листинг 4.. . void calcCoeffsParallel(igwdatamodel::IDataProfileIterator::Ptr temperature,. igwdatamodel::IDataProfileIterator::Ptr salinity,. igwdatamodel::IDataWriter::Ptr dst). {. tbb::task\_scheduler\_init init;. DataHolder holder(temperature,. salinity,. dst);. . tbb::task\_list list;. for(int i=0;i<init.default\_num\_threads();++i){. CalcTask& t = \*new(tbb::task::allocate\_root()) CalcTask(&holder);. list.push\_back(t);. }. tbb::task::spawn\_root\_and\_wait(list);. }. . Запуск параллельных задач осуществляется в новой функции calcCoeffsParallel. В ней создается объект DataHolder, инициализируется данными. Далее создается объект tbb::task\_scheduler\_init, который позволяет получить доступное на системе число потоков, как правило равное числу физических потоков поддерживаемое аппаратным обеспечением. Список заполняется количеством задач равным числу потоков. Число задач может быть и больше количества потоков, так как планировщик будет выполнять балансировку нагрузки автоматически. Но в данном случае задачи при старте и окончании расчета выполняют доступ к данным и время доступа мало. Поэтому увеличение задач не приведет к более эффективному распараллеливанию. В том случае если доступ к данным может занимать значительное время, то целесообразно создать отдельную параллельную задачу для наполнения данных. В таком сценарии создание количества задач большее, чем число физических потоков целесообразно, так как заполненные данными для расчета задачи будут ожидать в очереди выполнения и начинать расчет сразу в то время пока данные из выполненных задач будут сохраняться.. . Анализ достигнутой производительности. Расчеты проводились на сервере со следующей конфигурацией: 2 процессора Intel Xeon CPU E5-2650 v4 2.20Ггц, 64 Гб оперативной памяти, 512 Гб SSD диск Samsung Evo 850. Каждый процессор имеет 14 ядер и может выполнять до 28 потоков одновременно с помощью технологии Hyper Threading. Суммарно сервер может выполнять до 56 потоков одновременно.. Результаты профилировки распараллеленного приложения в профилировщике VTune представлено на рис. 2.. . . Рис.2 Результат профилировки параллельного алгоритма в профилировщике VTune. Анализ стека вызовов и времени выполнение показывает, что основное время работы приложения находится в расчетных задачах.. Задача расчета характеристик в каждой точке может быть неоднородна и неравномерна по вычислительной длительности между точками, вследствие физических характеристик и выбранных итеративных алгоритмов расчета. Как видно из скриншота загруженности потоков, балансировка нагрузки выполнена эффективно, все потоки загружены максимально однородно, что говорит о правильном делении алгоритма на подзадачи и эффективной работе планировщика задач. Полученное ускорение приведено на рис. 3.. . Рис.3 График ускорения параллельного алгоритма. На рис. 3 приведен анализ ускорения параллельного алгоритма при расчете на 2348 точках исходного набора данных. Пунктирной линией помечено идеальное ускорение, которое равно количеству запускаемых расчетных задач. В реальности (синяя кривая) практическое ускорение почти никогда не совпадает с идеальным, так как расчеты зачастую связаны с сохранением и загрузкой данных, а также синхронизацией, которые вносят накладные расходы процессорного времени. С увеличением количества потоков возрастает интенсивность вызова синхронизационных примитивов. Кроме того, дополнительные потоки в режиме Hyper Threading имеют более низкую производительность чем потоки запущенные в режиме равном количеству физических ядер процессора.. На рис. 4 представлена карта фазовой скорости, рассчитанная параллельным алгоритмом. Время расчета сократилось в 39 раз и составило около 165 секунд против 6437 секунд последовательного алгоритма. Данный расчетный блок входит в состав программного комплекса для исследования волн в океане [14] и позволит проводить вычисления значительно быстрее для новых наборов данных, охватывающих большие территории.. . Рис. 4 Карта распределения параметра фазовой скорости. . Заключение. В статье были рассмотрены современные средства профилировки, анализа и распараллеливания кода. Прикладная гидрофизическая задача расчета характеристик внутренних волн в стратифицированной жидкости сформулирована в математической постановке и реализована в программном коде. Выполнен анализ последовательного кода с помощью программного продукта VTune. Получены метрики времени выполнения участков кода. Предложен способ распараллеливания по данным, на основе независимости при расчете в разных географических локациях. Выполнена реализация параллельного алгоритма, с использованием библиотек TBB, на основе разделения алгоритма на задачи. Проведен анализ достигнутой производительности с помощью профилировщика VTune. В статье показано применение современных средств профилировки для быстрого и эффективного ускорения расчетов численных экспериментов в прикладной гидрофизической задаче.