Программные системы и вычислительные методы
Правильная ссылка на статью:
Викторов И.В., Гибадуллин Р.Ф. — Разработка синтаксического дерева для автоматизированного транслятора последовательного программного кода в параллельный код для многоядерных процессоров // Программные системы и вычислительные методы. – 2023. – № 1. DOI: 10.7256/2454-0714.2023.1.38483 EDN: ANMSZI URL: https://nbpublish.com/library_read_article.php?id=38483

Разработка синтаксического дерева для автоматизированного транслятора последовательного программного кода в параллельный код для многоядерных процессоров

Викторов Иван Владимирович

аспирант кафедры компьютерных систем Казанского национального исследовательского технического университета им. А.Н. Туполева - КАИ (КНИТУ-КАИ)

420015, Россия, республика Татарстан, г. Казань, ул. Большая Красная, 55, оф. 432

victorov.i.vl@yandex.ru

Гибадуллин Руслан Фаршатович

ORCID: 0000-0001-9359-911X

кандидат технических наук

доцент кафедры компьютерных систем Казанского национального исследовательского технического университета им. А.Н. Туполева-КАИ (КНИТУ-КАИ)

420015, Россия, республика Татарстан, г. Казань, ул. Большая Красная, 55, каб. 432

CSharpCooking@gmail.com

Статья из рубрики "Языки программирования"

DOI:

10.7256/2454-0714.2023.1.38483

EDN:

ANMSZI

Дата направления статьи в редакцию:

19-07-2022

Дата публикации:

28-01-2023

Аннотация: Появление многоядерных архитектур чрезвычайно стимулировало область параллельных вычислений. Однако разработка параллельной программы и ручное распараллеливание унаследованных последовательных программных кодов являются трудоемкой работой, программист должен обладать хорошими навыками применения

методов параллельного программирования. Данное обстоятельство определяет актуальность предмета исследования работы – разработка транслятора последовательного кода в параллельный. В статье приводится обзор существующих решений в рамках выбранного направления исследований, рассматриваются их преимущества и недостатки. Предлагается принцип формирования синтаксического дерева, который основан на JSON формате (текстовый формат обмена данными, основанный на JavaScript), и разбирается пример формирования синтаксического дерева на основе данного принципа. Результатом работы является подход к построению программной платформы трансляции последовательного кода в параллельный. Отличительной особенностью разработанной платформы является web-сервис, который потенциально позволяет расширить транслятор другими языками программирования. Взаимодействие с программной средой осуществляется посредством REST-запросов (HTTP-запросов, предназначенных для вызова удаленных процедур). Разработанная программная платформа состоит из трёх модулей: модуль обработки запросов, обеспечивающий взаимодействие с внешними системами посредством REST-запросов; модуль построения дерева, служащий для формирования синтаксического дерева на основе исходного программного кода; модуль преобразования кода, получающий параллельный программный код на основе синтаксического дерева.

Ключевые слова:

многоядерные процессоры, параллельные вычисления, параллельное программирование, многопоточное программирование, автоматизированный транслятор, JSON формат, языки программирования, синтаксическое дерево, веб-сервис, REST-запросы

Введение

В последнее время в целях повышения производительности вычислительных систем наблюдается увеличение числа мультипроцессорных комплексов, позволяющих оперировать большим объёмом вычислений. Для решения вычислительно сложных задач за приемлемое время требуется разработка программных модулей с применением многопоточного программирования. Для написания параллельных программ нужны высококвалифицированные программисты: в работе требуется участие аналитика, программиста, специалиста по тестированию. Компании несут существенные финансовые затраты, оплачивая работу IT-специалистов подобного класса.

Подспорьем для параллельного программирования являются специализированные языки, например, Cilk. С применением данного языка можно обеспечить вызовы функций асинхронно до тех пор, пока в родительской функции не будет достигнута точка синхронизации. Этот стиль параллелизма называется "разделяй и властвуй" (divide and conquer), или fork-join parallelism. В нём программа подразделяется на более мелкие задачи, которые планируются к параллельному выполнению. Помимо Cilk существуют специализированные библиотеки, например, Java Fork Join.

Зачастую программисту из-за сложности отладки изначально приходится писать последовательную (однопоточную) программу, а далее адаптировать решение в параллельный (многопоточный) эквивалент. Поэтому актуален вопрос по автоматизированному (или автоматическому) преобразованию последовательного кода в параллельный. Автоматическое преобразование не требует участия человека, но

сопровождается рядом трудностей по достижению однозначного решения (последовательный код может иметь различные параллельные эквиваленты). Автоматизированный транслятор использует диалог с программистом для уточнения свойств последовательной программы в целях получения наилучшего параллельного эквивалента, который потокобезопасен и оптимизирован по использованию аппаратных ресурсов и по критерию быстродействия.

В настоящее время отсутствуют программы, позволяющие успешно справляться с автоматической параллелизацией. Это связано, например, с наличием в программах вложенных функций, сложных выражений формирования циклов при которых усложняется анализ зависимостей по данным. Наиболее известными программными продуктами, реализующими автоматизированный подход, являются [1,2]: Parawise, Polaris, Cetus, Pluto, Polly, Par4All, CAPO, WPP, SUIF, VAST/PARALLEL, OSCAR, САПФОР.

Проведем сравнительный анализ на примере Parawise. Рассмотрим шаги алгоритма, отвечающего за преобразования последовательного в параллельный код в программном продукте Parawise (см. рис. 1):

1. Получение последовательного кода.
2. Анализ зависимостей последовательного кода на основе пользовательских данных и запись информации в БД.
3. Разделение входного кода на токены (объекты, создающиеся из лексемы в процессе лексического анализа, например, “{”, “}”, “case”, “do”, “else”, “for”, “if”, “sizeof” и т.д.) и запись информации в БД при участии пользователя.
4. Контроль масок (контроль максимальных и минимальных значений для условий) и запись информации в БД при участии пользователя.
5. Вычисление и генерация структуры синтаксического дерева и запись информации в БД.
6. Оптимизация кода (снижение использования памяти) и запись информации в БД.
7. Генерация кода и переход к следующему разделу кода в п.3, либо формирование результата преобразования последовательного кода в параллельный код.

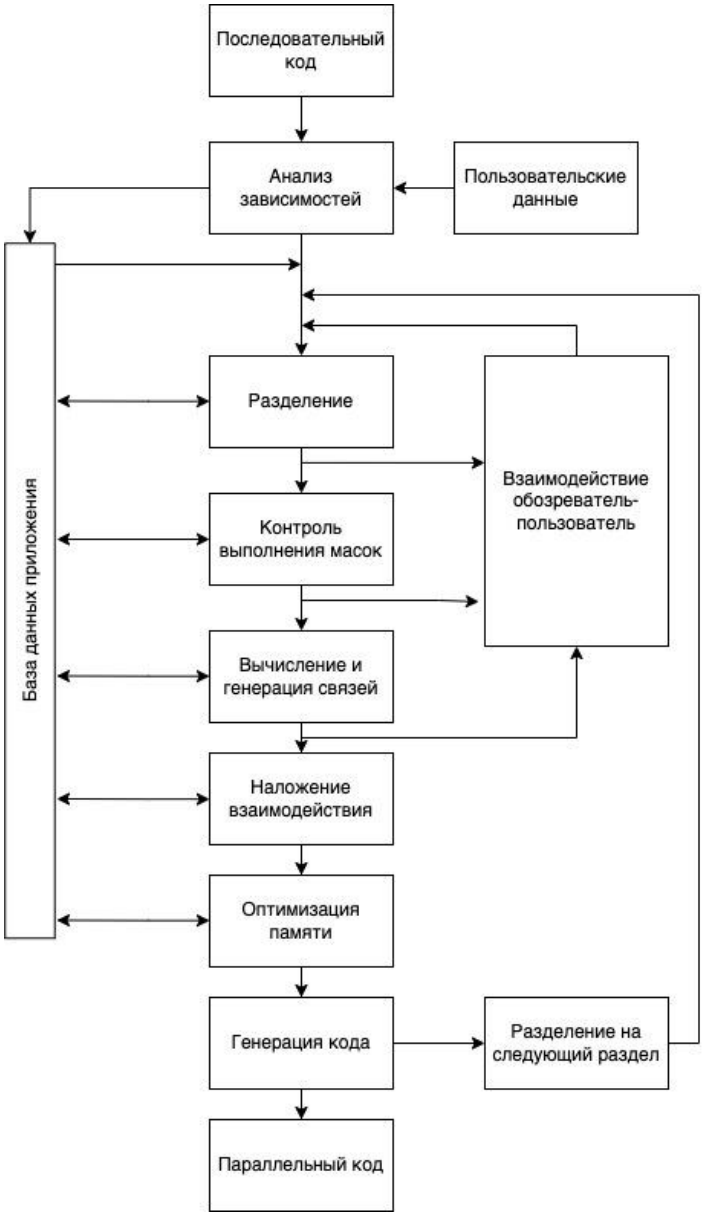

Рисунок 1. Алгоритм работы Parawise

На рисунке 2 представлены шаги преобразования последовательного кода в параллельный для программного продукта CAPO.

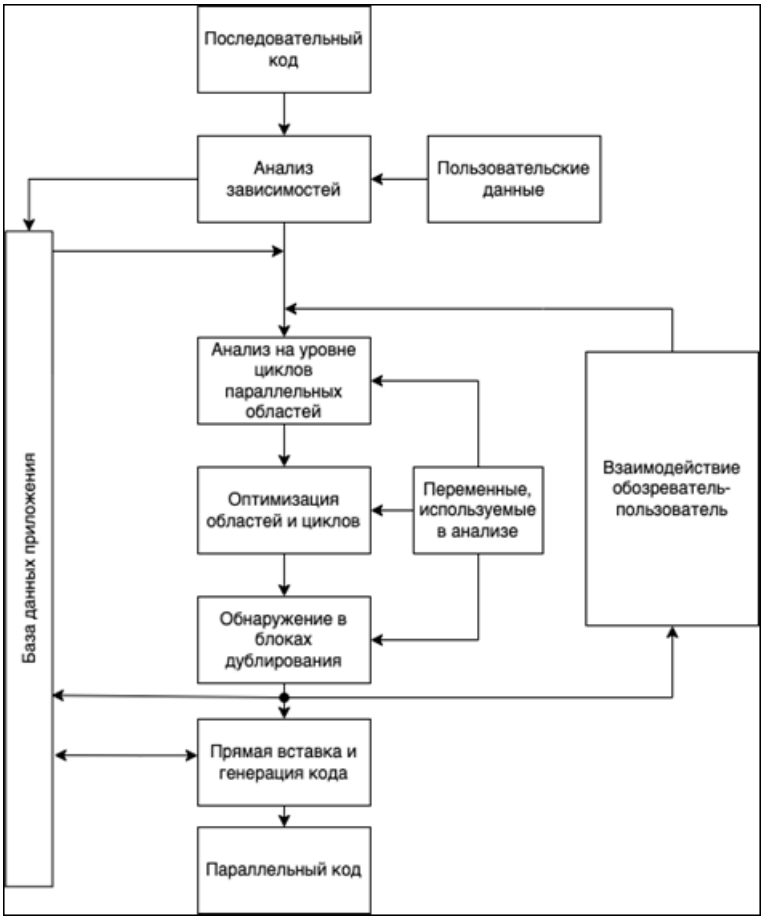

Рисунок 2. Алгоритм работы CAPO

1. Получение последовательного кода, который требуется распараллеливать.
2. Анализ зависимостей в полученном последовательном коде с обрабатываемыми пользовательскими данными и запись данных в БД.
3. На основе входных зависимостей, БД, данных взаимодействия с пользователем и переменных, используемых в анализе, формирование анализа на уровне циклов параллельных областей.
4. Оптимизация областей и циклов на основе анализа на уровне циклов параллельных областей и переменных, используемых в анализе, формирование оптимизации областей и циклов.
5. На основе полученных данных производится обнаружение дублирования в блоках. В БД вносятся промежуточные данные и происходит опрос пользователя на корректировку данных.
6. Выполнение прямой вставки полученными данными, генерация кода, обмен данными с БД.

Заметим, что Parawise и CAPO работают по схожему алгоритму, как и программы Cetus, Pluto, Polly, Par4All, CAPO, WPP, SUIF, VAST/PARALLEL, OSCAR, САПФОР. Одним из минусов является то, что данные программные продукты десктопные, т.е. имеют зависимость от целевой платформы (нельзя запустить на разных операционных системах), также требуется отслеживать обновление программного продукта. Отдельно следует выделить блок взаимодействия с пользователем, при котором пользователь выбирает требуемое решение. Реализация данного блока, представленная в CAPO и Parawise, исключает возможность автоматического параллелизма. Это требует от программиста наличие знаний в области параллельного программирования [3,4].

Предлагаемый в статье подход позволяет перейти к автоматизации перевода параллельного программного кода с наименьшим участием оператора, так как блок взаимодействия с пользователем является само обучаемым, основанным на глубоком обучении (deep learning) [5,6].

Архитектура транслятора

При разработке транслятора позволяющего преобразовать последовательный код в параллельный было принято решение сделать веб-приложение. Данное решение вызвано следующими факторами:

- веб-приложение не требует установки;
- все обновления происходят на сервере, доставляются пользователю сразу (достаточно перезагрузить веб-страницу);
- веб-приложение доступно из любой точки мира, с любого устройства, а пользовательские файлы всегда будут под рукой при наличии Интернет-соединения.

Проект транслятора имеет микросервисную архитектуру (рис. 3). Приложение состоит из нескольких программ (сервисов):

- сервис обработки запросов (обработчик запросов),
- сервис построения синтаксического дерева,
- сервис преобразования кода,
- база данных на базе СУБД PostgreSQL.

Сервис обработки запросов позволяет принимать REST-запросы пользователя, а именно запрос на преобразование последовательного кода в параллельный [7,8]. В запросе указывается последовательный код, требуемый для распараллеливания, а также язык программирования, на котором он написан. Также сервис позволяет делать запросы к базе данных для получения данных о синтаксическом дереве и преобразованном коде.

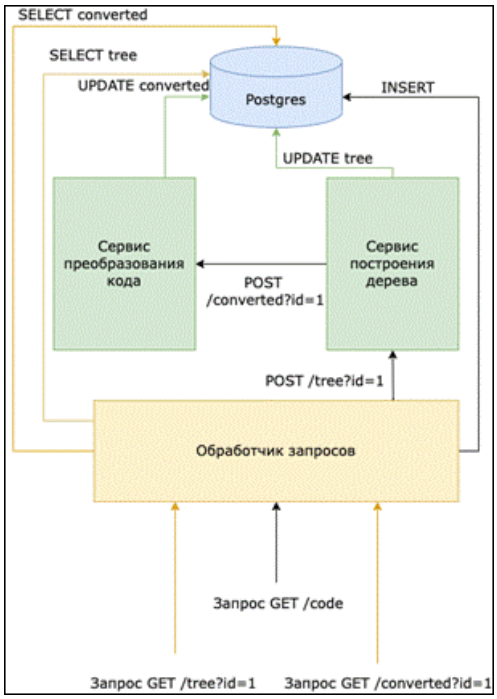

Рисунок 3. Архитектура транслятора

Сервис построения дерева преобразует последовательный код в синтаксическое дерево. Записывает данные о синтаксическом дереве в базу данных, после записи данных в БД отправляет запрос о завершении построения синтаксического дерева в сервис преобразования кода.

Сервисы формирования параллельной программы

Сервис преобразования кода забирает данные о построенном синтаксическом дереве из БД, строит различные схемы распараллеливания для исходной программы. Далее производится оценка построенной схемы по набору правил полученных из БД

соответствующего синтаксису языка преобразования текста из синтаксического дерева, и строит текст программы с применением принципов параллельного программирования и записывает в БД [9,10]. Создание параллельной программы можно рассмотреть на диаграмме последовательностей (рис. 4). Дадим описание данной диаграммы.

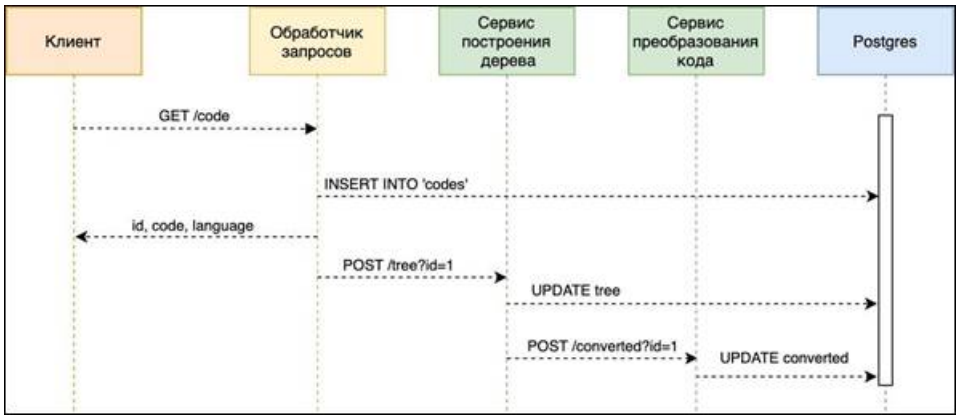

Рисунок 4. Диаграмма последовательностей создания параллельной программы

Клиент отправляет REST (GET '/code') запрос на преобразование последовательного кода в параллельный. Обработчик запросов вставляет новые данные в таблицу *code* базы данных и получает значение идентификатора нового поля. Следующим этапом обработчик запросов отправляет данные о записанном идентификаторе в сервис построения синтаксического дерева POST запросом. Сервис построения дерева получает данные о последовательном коде из БД и преобразует их в синтаксическое дерево. Результат преобразования записывается в БД и передается сервису преобразования кода POST запросом с идентификатором. Сервис преобразования кода преобразует синтаксическое дерево в параллельный код и записывает его в БД.

Построение программы выполняется в соответствии с микросервисной архитектурой. В качестве плюса микросервисов можно выделить наличие дополнительных возможностей по добавлению нового функционала программы для работы с другими языками: добавление синтаксического дерева или преобразование в параллельный код. В случае появления большей нагрузки со стороны пользователей добавляется брокер сообщений RabbitMQ или Kafka, позволяющий преобразовывать сообщение (для RabbitMQ – AMQP, Kafka – Kafka protocol) от приложения-источника в приложение-приемник, тем самым выступая между ними посредником. В этом случае клиент отправляет запрос на преобразование кода в сервис обработчика запросов. Обработчик запросов записывает данные в БД, отправляет код в брокер сообщений с идентификатором и языком программирования, который требуется преобразовать в синтаксическое дерево. Сервис построения дерева преобразовывает синтаксическое дерево в требуемый код для данного языка программирования. После этого происходит запись данных в БД и отправка идентификатора и названия языка программирования в брокер сообщений (система, преобразующая сообщение от источника данных "producer" в сообщение принимающей стороны "consumer"). Сервис преобразования кода через брокер сообщений преобразует код в параллельный эквивалент требуемого языка программирования.

Рисунок 5. Архитектура транслятора с использованием брокера сообщений

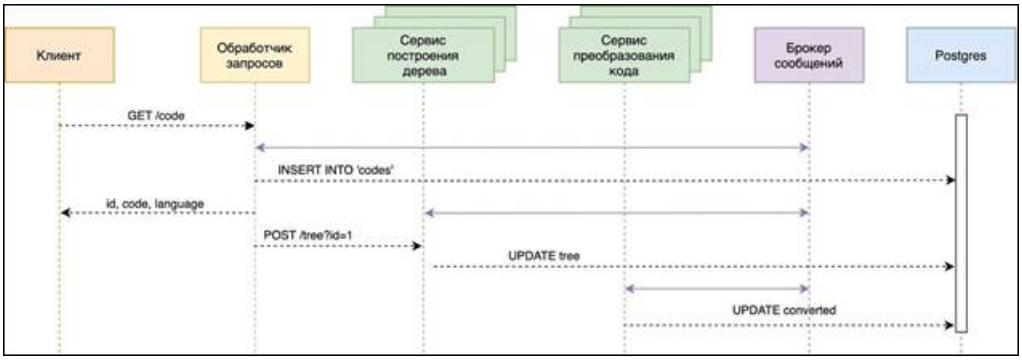

Рисунок 6. Диаграмма последовательностей создания параллельной программы с использованием брокера сообщений

Построение синтаксического дерева

В ходе проектирования были рассмотрены различные форматы синтаксического дерева.

Наиболее популярным является ANTLR4 это библиотека лексического анализа для Another Tool for Language Recognition (ANTLR). Принцип работы ANTLR основан на синтаксическом анализе входного потока данных, построения синтаксического дерева, обхода этого дерева и предоставления сгенерированного API для внедрения логики для каждого из событий обхода. API генерируется на основе *"грамматики "*, которая определяет структуру предложения, отдельных фраз/слов и конкретных символов. Однако данная библиотека написана давно, вследствие чего в ней отсутствует поддержка современных языков программирования и требуются дополнительные библиотеки во время выполнения.

Следующим распространенным форматом синтаксического дерева является Babel AST. К минусам данного формата можно отметить отсутствие поддержки языка программирования Си и большое количество передаваемых данных.

Исходя из выявленных недостатков рассмотренных решений принято решение разрабатывать синтаксическое дерево с собственным форматом.

Разработанное синтаксическое дерево характеризуется следующими этапами:

- Лексический анализ входной программы. Данный процесс характеризуется аналитическим разбором входной последовательности символов на распознанные группы – лексемы – с целью получения на выходе идентифицированных последовательностей, называемых «токенами» (подобно группировке букв в словах).
- Синтаксический анализ или синтаксический разбор – процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой.
- К полученной структуре применяется семантический анализ, процесс выявления «смысла» синтаксических конструкций. Примером является связывание идентификатора с его объявлением, типом данных, определение результирующего типа данных выражения.
- Далее производится преобразование в промежуточный код (в нашем случае это код в формате JSON).

Рассмотрим простой пример преобразования в синтаксическое дерево.

На вход отправляем JSON посылку с последовательным кодом:


```
{
"code": "int main()n {n for(int i=0; i<=4; i++)n{n printf("Текст"); n} n return 0;n}",
"language": "C"
}
```


Где code – последовательный код, предназначенный для параллелизации; language – язык программирования, на котором написан код.

Получаем сформированное синтаксическое дерево:


```
{
"tree": [
{
"type": "Program"
},
{
"type": "FunctionDeclaration",
"typeVar": "int",
"id": "main",
"params": [],
"body": [
{
"type": "ForStatement",
"init": [
{
"type": "VariableDeclaration",
"id": "i",
"value": "0",
"variable": "int"
},
{
"type": "CompareExpression",
"id": "i",
"mark": "<=",
"number": "4"
},
```

```
{
"type": "UpdateExpression",
"id": "i",
"operator": "++",
"prefix": "false"
}
],
"body": [
{
"type": "StringFunctionDeclared",
"value": "Текст"
}
]
},
{
"type": "ReturnStatement",
"argument": {
"type": "Literal",
"value": "0"
}
}
]
}
]
}
```


Описание используемых полей:

- tree – корневое дерево проекта;
- type – тип описываемого блока;
- typeVar – тип данных;
- id – название идентификатора;
- params – параметры описываемого объекта;
- body – тело описываемого блока;
- init – инициализируемые параметры;
- value – значение заданного параметра;
- variable – тип переменной;
- mark – значение знака;
- number – значение числа;
- operator – значение оператора;
- prefix – использование префикса;
- argument – значение аргумента.

Заключение

В данной статье проанализированы существующие реализации трансляторов последовательного кода в параллельный код, представлена архитектура принципиально нового транслятора, дано описание синтаксического дерева. Также продемонстрирован пример формирования синтаксического дерева, построение которого обеспечивается выполнением лексического, синтаксического и семантического анализов.

В будущем планируется расширить структурное представление синтаксического дерева,

дополнить сервис преобразования кода модулями преобразования последовательного кода в параллельный код с применением нейросетевого подхода [11,12].

Библиография

1. P. Czarnul, J. Proficz and K. Drypczewski, "Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems," Scientific Programming, vol. 2020, pp. 1058–9244, 2020.
2. D. B. Changdao, I. Firmansyah and Y. Yamaguchi, "FPGA-based computational fluid dynamics simulation architecture via high-level synthesis design method," in Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th Int. Symp., ARC 2020, Toledo, Spain, Springer Nature. vol. 12083, pp. 232, 2020.
3. D. Wang and F. Yuan, "High-performance computing for earth system modeling," High Performance Computing for Geospatial Applications, vol. 23, pp. 175–184, 2020.
4. M. U. Ashraf, F. A. Eassa, A. Ahmad and A. Algarni, "Empirical investigation: Performance and power-consumption based dual-level model for exascale computing systems," IET Software, vol. 14, no. 4, pp. 319–327, 2020.
5. B. Brandon, "Message passing interface (mpi)," in Workshop: High Performance Computing on Stampede, Cornell University Center for Advanced Computing (CAC), vol. 262, 2015.
6. A. Podobas and S. Karlsson, "Towards unifying openmp under the task-parallel paradigm," in Int. Workshop on OpenMP. Springer International Publishing, Springer International Publishing, Nara, Japan, vol. 9903, pp. 116–129, 2016.
7. M. U. Ashraf, F. Fouz and F. A. Eassa, "Empirical analysis of hpc using different programming models," International Journal of Modern Education & Computer Science, vol. 8, no. 6, pp. 27–34, 2016.
8. J. A. Herdman, W. P. Gaudin, S. Smith, M. Boulton, D. A. Beckingsale et al., "Accelerating hydrocodes with openacc, opencl and cuda," High Performance Computing, Networking, Storage and Analysis (SCC), vol. 66, pp. 465–471, 2012.
9. H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang et al., "High performance computing using mpi and openmp on multi-core parallel systems," Parallel Computing, vol. 37, no. 9, pp. 562–575, 2011.
10. M. Marangoni and T. Wischgoll, "Togpu: Automatic source transformation from c++ to cuda using clang/llvm," Electronic Imaging, vol. 1, pp. 1–9, 2016.
11. X. Xie, B. Chen, L. Zou, Y. Liu, W. Le et al., "Automatic loop summarization via path dependency analysis," IEEE Transactions on Software Engineering, vol. 45, no. 6, pp. 537–557, 2017.
12. M. S. Ahmed, M. A. Belarbi, S. Mahmoudi, G. Belalem and P. Manneback, "Multimedia processing using deep learning technologies, high-performance computing cloud resources, and big data volumes," Concurrency and Computation: Practice and Experience, vol. 32, no. 17, pp. 56–99, 2020

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Предмет исследования – разработка синтаксического дерева для автоматизированного транслятора последовательного программного кода в параллельный код для

многоядерных процессоров.

Методология исследования основана на теоретическом подходе с применением методов анализа, моделирования, алгоритмизации, схематизации, обобщения, сравнения, синтеза.

Актуальность исследования определяется широким распространением технологий параллельных вычислений, необходимостью разработки соответствующих программных систем и вычислительных методов, включая разработку синтаксического дерева для автоматизированного транслятора для многоядерных процессоров.

Научная новизна связана с предложенной автором архитектурой транслятора нового транслятора по переводу последовательного кода в параллельный, описанием синтаксического дерева, построение которого обеспечивается выполнением лексического, синтаксического и семантического анализов.

Статья написана русским литературным языком. Стиль изложения научный.

Структура рукописи включает следующие разделы: Введение (увеличение числа мультипроцессорных комплексов, решение вычислительно сложных задач за приемлемое время, разработка программных модулей с применением многопоточного программирования, участие аналитика, программиста, специалиста по тестированию, специализированный язык Cilk, точка синхронизации, стиль "разделяй и властвуй" (divide and conquer), или fork-join parallelism, специализированные библиотеки Java Fork Join, вопрос по автоматизированному (или автоматическому) преобразованию последовательного кода в параллельный, автоматизированный транслятор, диалог с программистом, оптимизация алгоритма по использованию аппаратных ресурсов и по критерию быстродействия, известные программные продукты Polaris, Cetus, Pluto, Polly, Par4All, CAPO, WPP, SUIF, VAST/PARALLEL, OSCAR, САПФОР, сравнительный анализ н примере Parawise, алгоритм работы CAPO), Архитектура транслятора (решение сделать веб-приложение, микросервисная архитектура транслятора, сервис обработки запросов, сервис построения дерева, сервисы формирования параллельной программы, сервис преобразования кода, диаграмма последовательностей создания параллельной программ, использованием брокера сообщений), Построение синтаксического дерева (форматы синтаксического дерева, ANTLR4, Babel AST, собственный формат, лексический анализ входной программы, синтаксический анализ, семантический анализ, преобразование в промежуточный код в формате JSON, пример преобразования в синтаксическое дерево, описание используемых полей), Заключение, Библиография.

Текст шесть рисунков. Для рисунков 5, 6 необходимо упоминание в предшествующем тексте.

Содержание в целом соответствует названию. Автором проанализированы известные реализации трансляторов последовательного кода в параллельный, представлена оригинальная архитектура транслятора, дано описание синтаксического дерева. Определены перспективы работы, связанные с применением нейросетевого подхода.

Библиография включает 12 источников зарубежных авторов – главы монографий, научные статьи, материалы научных мероприятий и пр. Библиографические описания некоторых источников требуют корректировки в соответствии с ГОСТ и требованиями

редакции, например:

1. Czarnul P. , Proficz J. , Drypczewski K. Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems // Scientific Programming. 2020. Vol. 2020. P. 1058–9244.

2. Changdao D. B., Firmansyah I., Yamaguchi Y. FPGA-based computational fluid dynamics simulation architecture via high-level synthesis design method // Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th Int. Symp. ARC 2020. Toledo, Spain : Springer Nature, 2020. Vol. 12083. P. 232.

4. Ashraf M. U., Eassa F. A, Ahmad A., Algarni A. Empirical investigation: Performance and power-consumption based dual-level model for exascale computing systems // IET Software, 2020. Vol. 14. № 4. P. 319–327.

Обращает внимание отсутствие ссылок на работы отечественных специалистов.

Апелляция к оппонентам (P. Czarnul, J. Proficz, K. Drypczewski, D. B. Changdao, I. Firmansyah, Y. Yamaguchi, D. Wang, F. Yuan, M. U. Ashraf, F. A. Eassa, A. Ahmad, A. Algarni, B. Brandon, A. Podobas, S. Karlsson, M. U. Ashraf, F. Fouz, F. A. Eassa, J. A. Herdman, W. P. Gaudin, S. Smith, M. Boulton, D. A. Beckingsale, H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, M. Marangoni, T. Wischgoll, X. Xie, B. Chen, L. Zou, Y. Liu, W. Le, M. S. Ahmed, M. A. Belarbi, S. Mahmoudi, G. Belalem, P. Manneback и др.) имеет место.

В целом материал представляет интерес для читательской аудитории и после доработки может быть опубликован в журнале «Программные системы и вычислительные методы».