DYNAMIC LOAD BALANCING ALGORITHMS APPLICATION AUTOMATION SUBSYSTEM DEVELOPMENT FOR LUNA SYSTEM

V. E. Malyshkin*,**,***, V. A. Perepelkin*,**,***, A. V. Chmil**

*Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russia
**Novosibirsk State University,
630090, Novosibirsk, Russia
***Novosibirsk State Technical University,
630073, Novosibirsk, Russia

DOI: 10.24412/2073-0667-2022-4-107-119
EDN: SKJWTU

The imbalance of computational load is one of the key problems of parallel programming. The imbalance of the computational load occurs due to various factors. The causes of the imbalance can be both hardware and software. Hardware imbalance occurs due to heterogeneity of computing system resources. The program imbalance is associated with such factors as the dynamics of the simulated phenomenon and an inefficiently parallelized program containing excessive communications, an unsuccessful distribution of calculations between nodes, etc.

Parallel implementation on supercomputers of large numerical models which requires dynamic load balancing on computing nodes is a complex task of system parallel programming. Solving this problem requires certain qualifications. Ordinary users of supercomputers in the field of scientific numerical modeling usually do not have such qualifications, which makes it difficult to use supercomputers in the relevant tasks. The problem is partially solved by using specialized software, where dynamic load balancing has already been implemented. However, the use of such software is not always possible, especially for new numerical models.

Dynamic load balancing is a relatively well-developed topic. There are many publications on this topic. There are algorithms, methods, software implementations and studies of their properties. However, the task of ensuring efficient and balanced performance of supercomputer computing resources is still time-consuming and requires relevant qualifications. Even a "simple" adjustment of the parameters of the load balancing algorithm can become an insurmountable problem in practice. The problem is especially relevant for modern supercomputers of the peta and exaflops ranges, since it is not trivial to provide a sufficiently full load of computing resources of such supercomputers even for simple tasks.

The elimination of the imbalance is a non-trivial task, for which there is no single method. There are many algorithms aimed at eliminating the imbalance, but none of them is universal.

The principal solution is automation of dynamic balancing of the computing load. Automation in this case refers to a situation when various methods, algorithms and programs that perform dynamic load balancing accumulate in some database or library in a form that allows their automatic application. User creates his program in such a way that the corresponding methods, algorithms, and programs are applied automatically, without the need for the user to deeply understand the problems of dynamic

load balancing. A specific case is the support for dynamic balancing of computational load in software such as Charm++ or PICADOR. A general case would be a situation where the programming system is not specialized, and all the knowledge about dynamic load balancing accumulated by researchers is available and automatically applied in the library.

It is significant that there are no universal dynamic balancing algorithms due to the algorithmic complexity of this problem in the general formulation. Therefore, various particular and heuristic methods and algorithms used in various practical tasks are being researched in the relevant field. Accordingly, the automation of dynamic load balancing is based on the accumulation of these particular and heuristic algorithms, as well as on the information about their appropriate usage, and on how to determine which case is more suitable in a particular situation.

The LuNA system of automatic design of parallel programs develops with an understanding of this circumstance. One of the tasks of the system is to ensure the accumulation and automatic application of knowledge about dynamic balancing of computing load. The paper reveals the question of how fundamentally the LuNA system is suited to provide this accumulation and automatic application, and also provides information about the extent at which this approach is currently implemented. In particular, the results of an experimental study of the performance characteristics of programs on LuNA systems using various dynamic load balancing algorithms are presented.

Key words: load balancing automation, fragmented programming technology, LuNA system.

References

1. Victor Malyshkin. Active Knowledge, LuNA and Literacy for Oncoming Centuries // LNCS, 2015. V. 9465. P. 292–303. DOI: 10.1007/978-3-319-25527-9-19.

2. Kale L. V., Krishnan S. Charm++. A portable concurrent object oriented system based on C++ // Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications, 1993. P. 91–108.

3. Bastrakov S. et al. Particle-in-cell plasma simulation on heterogeneous cluster systems // Journal of Computational Science, 2012. N 3 (6). P. 474–479.

4. Malyshkin V., Perepelkin V. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. // Parallel Computing Technologies, 2011. LNCS 6873. P. 53–61.

5. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed algorithm of data allocation in the fragmented programming system LuNA // International Conference on Parallel Computing Technologies. Springer, Cham, 2015. P. 80–85.

6. Acar U. A., Charguéraud A., Rainey M. Scheduling parallel programs by work stealing with private deques // Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming, 2013. P. 219–228.

РАЗРАБОТКА ПОДСИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРИМЕНЕНИЯ АЛГОРИТМОВ ДИНАМИЧЕСКОЙ БАЛАНСИРОВКИ НАГРУЗКИ ДЛЯ СИСТЕМЫ LUNA

В. Э. Малышкин*,**,***, В. А. Перепелкин*,**,***, А. В. Чмиль**

*Институт вычислительной математики и математической геофизики СО РАН,
630090, Новосибирск, Россия
**Новосибирский государственный университет,
630090, Новосибирск, Россия
**Новосибирский государственный технический университет,
630073, Новосибирск, Россия

УДК 004.4'242
DOI: 10.24412/2073-0667-2022-4-107-119
EDN: SKJWTU

В научном численном моделировании на суперЭВМ часто возникает проблема статического или динамического обеспечения баланса вычислительной нагрузки. Эта проблема не имеет эффективного универсального решения, вследствие чего на практике используются различные частные и эвристические алгоритмы балансировки нагрузки на вычислительные узлы. Несмотря на то, что эта тема хорошо разработана в литературе и имеется большое количество методов, алгоритмов и программ балансировки нагрузки, их применение в каждом конкретном случае представляет собой проблему. Даже настройка параметров подходящего алгоритма балансировки нагрузки может стать непреодолимым препятствием для пользователя суперЭВМ. Это обуславливает актуальность автоматического обеспечения балансировки нагрузки на узлы как подзадачи автоматического конструирования параллельных программ. Если в системе программирования имеется набор алгоритмов балансировки в виде, допускающем их автоматическое применение, то обозначенная проблема снимается с пользователя. В системе автоматического конструирования параллельных программ LuNA имеются средства для накопления и автоматического применения алгоритмов статической и динамической балансировки вычислительной нагрузки на узлы. В статье рассматривается подход, на основе которого такое накопление и применение возможно в системе LuNA.

Ключевые слова: автоматическое конструирование параллельных программ, динамическая балансировка нагрузки, технология фрагментированного программирования, система LuNA.

Введение. Параллельная реализация на суперкомпьютерах больших численных моделей, требующих динамической балансировки нагрузки на вычислительные узлы, является сложной задачей системного параллельного программирования, решение которой требует

Разработка алгоритмов метабалансировки выполнена в рамках государственного задания ИВМ и МГ СО РАН 0251-2021-0005. Программная реализация алгоритмов выполнена при частичной поддержке гранта МОН РК «ИРН AP09058423».

соответствующей квалификации и трудозатрат. Большинство пользователей суперкомпьютеров, специализирующихся в своей области научного численного моделирования, такой квалификацией не обладают. Это затрудняет применение ими суперкомпьютеров в соответствующих задачах. Частично проблема решается применением специализированного программного обеспечения (например, пакет моделирования), где динамическая балансировка нагрузки уже реализована, но применение такого ПО не всегда возможно, особенно для новых и малораспространенных численных моделей.

Динамическая балансировка нагрузки является относительно хорошо проработанной в литературе темой. Имеются алгоритмы, методы, программные реализации и исследования их свойств. Тем не менее, задача обеспечения эффективной и сбалансированной работы вычислительных ресурсов суперкомпьютера все еще трудоемка и требует соответствующей квалификации. Даже «простая» настройка параметров уже реализованного алгоритма балансировки нагрузки может стать непреодолимой проблемой на практике. Проблема особенно актуальна для современных суперкомпьютеров пета- и экзафлопсных диапазонов производительности, т. к. обеспечить равномерную и полную загрузку вычислительных ресурсов таких суперкомпьютеров нетривиально даже для самых простых задач.

Одним из важнейших направлений исследований, направленных на преодоление этой проблемы, является автоматизация обеспечения динамической балансировки вычислительной нагрузки. Под автоматизацией в данном случае понимается ситуация, когда различные методы, алгоритмы и программы, осуществляющие динамическую балансировку нагрузки, накапливаются в некоторой базе или библиотеке в виде, допускающем их автоматическое применение [1], а пользователь представляет свою программу в таком виде, при котором соответствующие методы, алгоритмы и программы применяются автоматически, без необходимости пользователя глубоко понимать проблематику динамической балансировки нагрузки. Вырожденным случаем этого подхода является поддержка динамической балансировки вычислительной нагрузки в ПО, таком как Charm++ [2] или PICADOR [3]. Более общим случаем является ситуация, когда система программирования не является специализированной, и в ней накапливаются и автоматически применяются знания о динамической балансировке нагрузки.

Существенно, что универсальных алгоритмов динамической балансировки нет ввиду высокой алгоритмической сложности этой задачи в общей постановке. Поэтому исследуются различные частные и эвристические методы и алгоритмы, применимые в тех или иных практических задачах. Соответственно, и автоматизация динамической балансировки нагрузки возможна на основе накопления этих частных и эвристических алгоритмов, а также информации о том, в каких случаях и с какими параметрами их уместно применять, и как эти случаи выявлять автоматически.

Система автоматического конструирования параллельных программ LuNA [4] развивается с пониманием этого обстоятельства, и одна из задач системы состоит в том, чтобы обеспечить накопление и автоматическое применение знаний о динамической балансировке вычислительной нагрузки. В статье раскрывается вопрос о том, каким образом принципиально система LuNA подходит к тому, чтобы обеспечить это накопление и автоматическое применение, а также приводятся сведения о том, в какой мере этот подход реализован на текущий момент. В частности, приводятся результаты экспериментального исследования характеристик производительности программ на системе LuNA с применением различных алгоритмов динамической балансировки нагрузки.

Остальная часть статьи организована следующим образом. В разделе 2 описывается, как в системе LuNA рассматривается задача конструирования параллельной программы и как при этом ставится задача динамической балансировки вычислительной нагрузки. В разделе 3 излагаются основные идеи предлагаемого подхода к решению проблемы, в разделе 4 представлены результаты экспериментального исследования.

1. Постановка задачи. Программа на языке LuNA представляет собой описание множества вычислительных процессов — фрагментов вычислений (ФВ). Каждый ФВ определяется триплетом <in, mod, out>, где mod — это некоторый вычислительный модуль (например, последовательная процедура) без побочных эффектов, a in и out — имена, соответственно, входных и выходных параметров, к которым модуль применяется. Эти параметры иммутабельны, сериализуемы и называются фрагментами данных (ФД). Если некоторый ФД является выходным для одного ФВ и выходным для другого ФВ, то второй ФВ информационно зависит от первого.

Множества ФВ и ФД существуют явно в процессе выполнения программы, что вместе с отсутствием побочных эффектов в ФВ и сериализуемостью и иммутабельностью ФД позволяет исполнительной подсистеме динамически распределять и перераспределять ФВ и ФД по узлам мультикомпьютера. Исполнение ФВ осуществляется на одном из узлов при условии, что все его входные ФД находятся в памяти этого узла. Результатом исполнения является вычисление значений выходных ФД, которые в дальнейшем могут храниться на этом же узле или быть переданы на другие узлы мультикомпьютера для хранения или потребления другими ФВ.

Система LuNA обеспечивает распределенное исполнение множества ФВ, доставку по сети входных ФД на узлы их потребления фрагментами вычислений, выбор узлов хранения ФД и исполнения ФВ, выбор порядка выполнения ФВ и т. п. Задача динамической балансировки вычислительной нагрузки на узлы мультикомпьютера понимается как перераспределение ФВ и ФД по его узлам с целью выравнивания нагрузки на узлы и с учетом накладных расходов, необходимых для мониторинга и прогнозирования загрузки и перераспределения ФВ и ФД.

На текущий момент в системе LuNA имеется 3 варианта балансировки нагрузки: статическая балансировка, rope of beads [5] и work requesting [6].

Для статической балансировки разработчик или транслятор определяют, на какой узел должны отправиться ФВ и ФД.

В алгоритме rope of beads ФВ и ФД отображаются на отрезок фиксированной длины. Этот отрезок разделен на подотрезки, которые соответствуют узлам вычислительной системы. В случае возникновения дисбаланса загрузки разбиение отрезка на подотрезок изменяется динамически. Как следствие, фрагменты перераспределяются по узлам в соответствии с новым разбиением. Этот механизм направляется на устранение дисбаланса загрузки. При этом соседство (взаимное расположение) фрагментов на отрезке (а значит, и в подотрезках) не изменяется. Это позволяет экономить на коммуникациях путем размещения информационно связанных ФВ и ФД в окрестности друг друга на отрезке.

В алгоритме work requesting недогруженный узел запрашивает избыточные задачи у соседних узлов. Данный алгоритм может использоваться совместно с другими и нацелен в основном на устранение «точечного» дисбаланса загрузки вычислительных узлов (когда в целом нагрузка равномерна, но встречаются отдельные недогруженные или перегруженные узлы).

Для осуществления динамической балансировки нагрузки на узлы требуется решать такие задачи как мониторинг загруженности узлов мультикомпьютера (напр., загрузка ядер, расход оперативной памяти) и коммуникационной сети, обнаружение и прогнозирование дисбаланса загрузки, обмен информацией, необходимой для работы балансировщика (напр., о загруженности узлов), принятие решения о проведении балансировки, выбор ФВ и ФД для перераспределения, осуществление перераспределения. Решение всех этих задач зависит от конкретного метода или алгоритма динамической балансировки нагрузки. Система LuNA спроектирована таким образом, чтобы все эти задачи могли решаться отдельными слабосвязанными модулями, и обеспечивает базовые низкоуровневые операции, такие как передача фрагментов по сети, облегчая разработку модулей. В частности, на этой основе может быть реализована и метабалансировка нагрузки на узлы. А именно, в системе могут присутствовать два и более различных модулей, осуществляющих динамическую балансировку вычислительной нагрузки, управление которыми осуществляет отдельный модуль — метабалансировщик (или оркестратор).

В функции метабалансировщика входит включение и выключение имеющихся модулей динамической балансировки нагрузки на узлы, а также настройка параметров этих модулей, таких как порог дисбаланса, доля пересылаемой нагрузки, период осреднения нагрузки при ее мониторинге и т. п. Разные алгоритмы балансировки нагрузки могут иметь различные параметры, поэтому так или иначе метабалансировщик подразумевает наличие некоторого общего интерфейса, с которым модули балансировки нагрузки должны быть совместимы. Через этот интерфейс метабалансировщик получает информацию о свойствах конкретного модуля, наличии у него параметров и способа настройки этих параметров на конкретную ситуацию.

Рассмотрим особенности вычислительной модели, используемой в системе LuNA с точки зрения динамической балансировки вычислительной нагрузки на узлы мультикомпьютера.

— Явно выделенные подзадачи — ФВ. Динамическая балансировка подразумевает, что балансировщик нагрузки способен выделить часть работы из общего объема вычислений и переместить ее на другой вычислительный узел. ФВ является такой единицей. Контекст исполнения ФВ — это его входные ФД, и он описан явно. Это позволяет системе автоматически выделять и передавать подзадачи с узла на узел, не влияя на вычисляемые значения, т. к. ФВ не имеет побочных эффектов, а ФД — иммутабельны. При этом информационные зависимости между ФВ выражены явно, что позволяет учитывать это при анализе и прогнозировании загрузки узлов.

— Гранулярность подзадач. Вообще говоря, чем крупнее единица балансировки, тем менее точного баланса возможно добиться. Размеры ФВ и ФД, как правило, являются параметром LuNA-программы, что позволяет сделать их достаточно маленькими, чтобы обеспечить требуемую гранулярность. Слишком маленькие ФВ и ФД, с другой стороны, означают увеличение накладных расходов на работу исполнительной системы, чего следует избегать.

— Информативность программы. Балансировка нагрузки может осуществляться более эффективно, если имеется дополнительная информация о самой программе, помимо информации о текущей загрузке оборудования. LuNA-программа имеет явную структуру информационных зависимостей (выраженную в информационных зависимостях между фрагментами), что позволяет анализировать, как будет изменяться нагрузка со временем, и какие потребуются коммуникации для передачи ФД по сети. В ряде случаев может

иметься информация об оценочных временах выполнения ФВ и о прогнозируемых размерах ФД. Эта информация также может быть использована при балансировке нагрузки. В качестве источников этой информации могут выступать программист, профили предыдущих запусков программы, статистика текущего исполнения и т. п.

Для фиксации частичных решений о способе исполнения LuNA-программы в системе используются аннотации кода или внутреннего представления программы во время трансляции или исполнения. Эти аннотации называются рекомендациями и могут частично определяться разработчиком LuNA-программы, частично — транслятором и исполнительной системой. Т. о., процесс конструирования и исполнения LuNA-программы можно рассматривать как поэтапное принятие решений о поведении программы (распределение фрагментов по узлам, порядок выполнения ФВ и т. п.), которые в итоге реализуются при исполнении программы.

2. Организация модулей балансировки нагрузки. Задачу накопления и применения алгоритмов динамической балансировки нагрузки на узлы отделим от задачи автоматического выбора и настройки параметров алгоритма. Будем рассматривать только первую задачу.

Идея заключается в том, чтобы алгоритмы балансировки в системе представлялись как унифицированные заменяемые модули (плагины), и их можно было бы переключать и настраивать их параметры при использовании системы. Для реализации этой идеи требуется выделить общий интерфейс для модулей балансировки нагрузки, предоставляющий инструментарий для взаимодействия с системой, а также разработать интерфейс для модулей метабалансировки.

Основные требования к интерфейсу:

1) механизм коммуникации между узлами; 2) возможность определения нагруженности узла; 3) возможность задавать параметры работы модуля балансировки нагрузки.

Базовый алгоритм исполнения LuNA-программ подразумевает, что ФВ как вычислительный процесс в определенный момент порождается и может мигрировать с узла на узел. При этом система обеспечивает доставку по сети всех входных ФД к этому ФВ, и когда при ФВ окажется полный набор его входных ФД, то он может исполняться. В связи с этим можно выделить два этапа, когда миграцию ФВ осуществлять проще всего — это в начале, когда ФВ только создан, и перед исполнением, когда все входные ФД уже находятся при ФВ, и можно перемещать их с узла на узел как одно целое. Поэтому важно иметь возможность алгоритмам балансировки определять, в каком состоянии находится ФВ.

Еще одной особенностью системы является то, что балансировке подвержены не только ФВ, но и ФД. В частности, в алгоритме rope of beads при изменении разбиения отрезка на подотрезки изменяется распределение по узлам не только ФВ, но и ФД. Соответственно, необходим механизм, позволяющий модулю балансировки нагрузки перемещать ФД.

Был разработан интерфейс модуля динамической балансировки нагрузки на узлы, соответствующий предъявляемым требованиям и, в частности, достаточный для включения через этот интерфейс трех имеющихся в системе LuNA алгоритмов балансировки нагрузки (в данном случае статический алгоритм можно рассматривать как вырожденный случай динамического, когда заданное распределение не изменяется по ходу выполнения программы). Приведем этот интерфейс в синтаксисе языка C++, т. к. это язык исполнительной системы LuNA (листинг 1).

Листинг 1. Интерфейс модуля динамической балансировки нагрузки

```
01: int calculate_rank(bool with_data, int pos=0)=0;
02: std::set<int> get_msg_acceptable_tags();
03: void accept_msg(int src, int tag, void *buf,
size_t size);
04: void notify_pool_empty();
05: void notify_pool_submitted();
06: void send(int dest, int tag, const void *buf,
size_t size, std::function<void()> finisher=nullptr);
07: int size();
08: int rank();
09: unsigned int get_queue_size();
10: void relocate_DFs();
```

Метод *calculate_rank* является обязательным для реализации и реализует важнейшую логику модуля. Методы интерфейса балансировщика можно разделить на 3 группы:

1. Абстрактный метод, реализация обязательна (строка 01);

2. Абстрактный метод, реализация не обязательна (строки 02–05);

3. Метод реализован в абстрактном варианте балансировщика. Может использоваться внутри методов балансировщиков (строки 06–10).

Основным методом интерфейса балансировки является calculate_rank, рассчитывающий, на какой узел следует отправить фрагмент. Аргумент with_data сообщает о том, когда происходит балансировка — до или после получения всех входных фрагментов данных. Параметр pos является опциональным. Для некоторых алгоритмов балансировки он служит для того, чтобы сопоставить порядковый номер с фрагментом. Например, для статической балансировки узел, на который должен отправиться фрагмент, может определяется как pos%size, где size — это количество вычислительных узлов.

Также интерфейс балансировки должен позволять обмениваться сообщениями между балансировщиками на разных узлах. Для этого в интерфейс было введено 5 методов:

1. std::set<**int**> get_msg_acceptable_tags() — метод, возвращающий список идентификаторов сообщений, которые принимает балансировщик.

2. **void** accept_msg(**int** src, **int** tag, **void** *buf, size_t size) — метод, в который исполнительная система подает полученные сообщения, если tag этого сообщения входит в набор, возращаемый методом get_msg_acceptable_tags.

3. **void** send(**int** dest, **int** tag, **const void** *buf, size_t size, std::function<**void**()> finisher=nullptr) — метод, с помощью которого балансировщик может отправлять сообщения на другие узлы.

4. **int** size() — получить количество узлов в системе.

5. **int** rank() — получить идентификатор текущего узла.

Для некоторых алгоритмов балансировки требуется отслеживать состояние очереди задач, чтобы при отсутствии нагрузки на узле производить балансировку фрагментов между узлами. Было реализовано 3 метода, позволяющих следить за состоянием очереди задач:

1. **void** notify_pool_empty() — метод оповещает балансировщик, когда задачи в очереди кончились.

2. **void** notify_pool_submitted() — метод оповещает балансировщик, когда задача была добавлена в очередь.

3. **unsigned int** get_queue_size() — метод возвращает текущее количество задач в очереди.

Для динамических балансировок был добавлен метод **void** relocate_DFs(). Данный метод выполняет перемещение всех фрагментов данных, имеющихся на узле, на требуемый узел. Этот метод требуется вызывать после изменения конфигурации динамического балансировщика, чтобы фрагменты находились на актуальном для них узле.

В результате существующие в LuNA алгоритмы балансировки были адаптированы под общий интерфейс.

На текущем этапе разработки метабалансировщик является хранилищем балансировок и входной точкой управления ими. Метабалансировщик, как и другие алгоритмы балансировки, реализует общий интерфейс. Методы, которые он реализует:

1. **int** calculate_rank(bool with_data, **int** pos=0). Перенаправляет вызов во включенный алгоритм балансировки;

2. std :: set<**int**> get_msg_acceptable_tags(). Возвращает набор тегов всех хранящихся балансировщиков;

3. **void** accept_msg(**int** src, **int** tag, **void** *buf, size_t size). Перенаправляет сообщение в требуемый балансировщик;

4. **void** notify_pool_empty(). Нотифицирует балансировщики о том, что очередь задач пуста;

5. **void** notify_pool_submitted(). Нотифицирует балансировщики о том, что в очередь задач была добавлена задача.

В LuNA-программе алгоритм балансировки определяется с помощью рекомендаций use_for_balance — это алгоритм work requesting, locator_cyclic — это статическая балансировка и locator_rope — это алгоритм rope of beads. Эта рекомендация преобразуется в код по балансировке на этапе трансляции LuNA программы. Для того чтобы поддержать динамический выбор алгоритма балансировки, была добавлена рекомендация locator_abstract, которая транслируется в код, использующий метабалансировщик для выбора алгоритма балансировки фрагмента. Таким образом метабалансировщик решает потребность повторной трансляции LuNA программы при изменении алгоритма балансировки.

3. Тестирование. Целью тестирования была проверка работоспособности предложенной схемы на примере различных задач, для чего была исследована зависимость времени выполнения задач от алгоритма балансировки нагрузки и его параметров.

Тестирование производилось на кластере Информационно-вычислительного центра Новосибирского государственного университета. Характеристика каждого узла: 12 ядер и 24 ГБ ОЗУ.

Для тестирования была взята задача «Метод частиц в ячейках». Эта задача хорошо подходит для тестирования балансировки нагрузки из-за неравномерности нагрузки, требуемой для исполнения фрагментов.

В задаче вычислительная сетка разбивается по осям с помощью параметров NX, NY и NZ. Количество фрагментов в задаче, которые исполняются на каждом шаге, равняется NX?NY?NZ. Параметр Eps — это максимальная допустимая величина невязки на этапе решения уравнения Пуассона. Он влияет на объем вычислений на единицу данных (чем меньше Eps, тем больше вычислений).

Тестирование проводится на следующих алгоритмах балансировки нагрузки:

— без балансировки — все задачи исполняются на одном узле;

Таблица 1

Результаты для значений параметров NX = 8, NY = 8, NZ = 2, Eps = 0.01

Тип балансировки	Количество узлов	Параметры балансировки	Результат (сек)
Без балансировки	1		558.50
Work requesting	2	Количество соседей: 2	504.48
Work requesting	4	Количество соседей: 2	353.99
Work requesting	4	Количество соседей: 4	233.22
Статическая	2		471.41
Статическая	4		180.54
Статическая + work requesting	2	Количество соседей: 2	471.27
Статическая + work requesting	4	Количество соседей: 2	151.92
Статическая + work requesting	4	Количество соседей: 4	174.38
Rope of Beads	2	Подотрезков: 20	658.89
Rope of Beads	2	Подотрезков: 60	684.67
Rope of Beads	4	Подотрезков: 20	123.25
Rope of Beads	4	Подотрезков: 60	148.97
Rope of Beads + work requesting	2	Подотрезков: 20; Количество соседей: 2	1248.49
Rope of Beads + work requesting	2	Подотрезков: 60; Количество соседей: 2	1229.89
Rope of Beads + work requesting	4	Подотрезков: 20; Количество соседей: 2	133.38
Rope of Beads + work requesting	4	Подотрезков: 20; Количество соседей: 4	137.23
Rope of Beads + work requesting	4	Подотрезков: 60; Количество соседей: 2	152.76
Rope of Beads + work requesting	4	Подотрезков: 60; Количество соседей: 4	149.23

— статическая балансировка — балансировка выполняется статически вручную, силами разработчика, который задает фрагментам номер узла, на котором они должны исполняться (или храниться);

— Work requesting — динамический алгоритм балансировки work requesting; параметр «количество соседей» задает окрестность узла, в которой недогруженный узел запрашивает работу;

— Rope of beads — динамический алгоритм балансировки rope of beads. Алгоритм балансирует нагрузку исходя из загрузки текущего и соседних узлов, а разработчик статически присваивает фрагментам вычислений и данных координату на отрезке, на основе которой определяется, на каком узле должен исполняться фрагмент. Параметр «количество подотрезков» определяет количество подотрезков.

В табл. 1 представлены результаты тестирования для следующих значений параметров задачи: $NX = 8$, $NY = 8$, $NZ = 2$, $Eps = 0.01$.

Как видно из результатов, лучше всего в данном тесте себя показал алгоритм Rope of Beads с количеством подотрезков 20. Это связано с тем, что при статической балан-

Таблица 2

Результаты для значений параметров NX = 8, NY = 8, NZ = 1, Eps = 0.001

Тип балансировки	Количество узлов	Параметры балансировки	Результат (сек)
Без балансировки	1		128.88
Work requesting	2	Количество соседей: 2	129.02
Work requesting	4	Количество соседей: 2	97.94
Work requesting	4	Количество соседей: 4	93.54
Статическая	2		129.92
Статическая	4		88.47
Статическая + work requesting	2	Количество соседей: 2	130.19
Статическая + work requesting	4	Количество соседей: 2	89.32
Статическая + work requesting	4	Количество соседей: 4	117.53
Rope of Beads	2	Подотрезков: 20	130.59
Rope of Beads	2	Подотрезков: 60	137.63
Rope of Beads	4	Подотрезков: 20	91.57
Rope of Beads	4	Подотрезков: 60	93.40
Rope of Beads + work requesting	2	Подотрезков: 20; Количество соседей: 2	128.76
Rope of Beads + work requesting	2	Подотрезков: 60; Количество соседей: 2	130.23
Rope of Beads + work requesting	4	Подотрезков: 20; Количество соседей: 2	92.86
Rope of Beads + work requesting	4	Подотрезков: 20; Количество соседей: 4	95.27
Rope of Beads + work requesting	4	Подотрезков: 60; Количество соседей: 2	93.17
Rope of Beads + work requesting	4	Подотрезков: 60; Количество соседей: 4	97.73

сировке нагрузка между узлами разделена довольно неравномерно и использование work requesting для решения точечного дисбаланса не полностью устраняет дисбаланс. Также можно выделить то, что использование work requesting ускоряет работу программы почти во всех случаях. Так как work requesting является вспомогательным балансировщиком, который используется для решения точечного дисбаланса, его результаты отдельно от других алгоритмов оказались хуже, чем у других балансировщиков.

Параметры задачи: $NX = 8$, $NY = 8$, $NZ = 1$, $Eps = 0.001$.

В данном тесте было уменьшено количество «тяжелых» фрагментов, но увеличена точность Eps.

В данном тесте лучше всего показала себя статическая балансировка. Причина заключается в том, что из-за меньшего количества тяжелых частиц исполнение программы происходит более сбалансировано и динамическая балансировка не требуется, так как она содержит излишние коммуникации и поэтому проигрывает статической.

Заключение. Рассмотрена проблематика обеспечения динамической балансировки вычислительной нагрузки при автоматическом конструировании параллельных программ

численного моделирования в системе LuNA. Рассмотрены алгоритмы динамической балансировки нагрузки на узлы, применяемые в этой системе, а также представлены подход к метабалансировке нагрузки и ее поддержка. Результаты экспериментального исследования носят предварительный характер, но подтверждают, что в различных условиях предпочтительными оказываются различные алгоритмы динамической балансировки нагрузки или значения параметров этих алгоритмов. Система LuNA может выступать в качестве основы для накопления различных алгоритмов динамической балансировки нагрузки и экспериментального исследования подходов к автоматизации выбора этих алгоритмов из числа имеющихся и настройки их параметров.

Список литературы

1. Victor Malyshkin. Active Knowledge, LuNA and Literacy for Oncoming Centuries // LNCS, 2015. V. 9465. P. 292–303. DOI: 10.1007/978-3-319-25527-9-19.

2. Kale L. V., Krishnan S. Charm++. A portable concurrent object oriented system based on C++ // Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications, 1993. P. 91–108.

3. Bastrakov S. et al. Particle-in-cell plasma simulation on heterogeneous cluster systems // Journal of Computational Science, 2012. N 3 (6). P. 474–479.

4. Malyshkin V., Perepelkin V. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. // Parallel Computing Technologies, 2011. LNCS 6873. P. 53–61.

5. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed algorithm of data allocation in the fragmented programming system LuNA // International Conference on Parallel Computing Technologies. Springer, Cham, 2015. P. 80–85.

6. Acar U. A., Charguéraud A., Rainey M. Scheduling parallel programs by work stealing with private deques // Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming, 2013. P. 219–228.

Виктор Эммануилович Малышкин — получил степень магистра математики в Томском государственном университете (1970), степень доктора технических наук в Новосибирском государственном университете (1993). В настоящее время является заведующим лабораторией синтеза параллельных программ в Институте вычислительной математики и математической геофизики СО РАН. Он также основал и в настоящее время возглавляет кафедру параллельных вычислений в Национальном исследовательском университете Новосибирска. Является одним из организаторов международной конференции PaCT (Parallel Computing Technologies), проводимых каждый нечетный год в России.

Имеет более 110 публикаций по параллельным и распределенным вычислениям, синтезу параллельных программ, суперкомпьютерному программному обеспечению и приложениям, параллельной реализации крупномасштабных численных моделей.

В настоящее время область его интересов включает параллельные вычислительные технологии, языки и системы параллельного программирования, методы и средства параллельной реализации крупномасштабных численных моделей, технология активных знаний.

Victor Emmanuilovich Malyshkin graduated from Tomsk State University with the master of sciences degree in 1970, defended his candidate dissertation in physical and mathematical sciences in Computing Centre of SB RAS in 1984, and defended his doctoral dissertation in technical sciences in Novosibirsk

State University (1993). Currently is head of parallel programs synthesis laboratory in the Institute of computational mathematics and mathematical geophysics SB RAS.

Is also a founder and head of the chair of parallel computing in Novosibirsk State University. Is one of organizers of the PaCT (Parallel Computing Technologies) international conference, being held each odd year in Russia. Has more than 110 publications on parallel and distributed computing, parallel programs synthesis, supercomputing software and applications, parallel implementation of large-scale numerical models. Current scientific interests include parallel computing technologies, languages and systems of parallel computing, methods and tools of software implementation of large-scale numerical models, the active knowledge technology.

Перепелкин Владислав Александрович — научный сотрудник Института вычислительной математики и математической геофизики СО РАН; старший преподаватель каф. параллельных вычислений факультета информационных технологий Новосибирского национального исследовательского государственного университета. Тел.: (383) 330-89-94, e-mail: perepelkin@ssd.sscc.ru. В 2008 году окончил Новосибирский государственный университет с присуждением степени магистра техники и технологии по направлению «Информатика и вычислительная техника». Имеет более 20 опубликованных статей по теме автоматизации конструирования параллельных программ в области численного моделирования. Является одним из основных разработчиков экспериментальной системы автоматизации конструирования численных параллельных программ для мультикомпьютеров LuNA (от Language for Numerical Algorithms). Область профессиональных интересов включает автоматизацию конструирования параллельных программ, языки и системы параллельного программирования, высокопроизводительные вычисления.

Perepelkin Vladislav Aleksandrovich. Graduated from Novosibirsk State University in 2008 with the master degree in engineering and technology in computer science. Nowadays has the research position at the Institute of Computational Mathematics and Mathematical Geophysics (Siberian Branch of Russian Academy of Sciences), and also has a part time senior professor position in the Novosibirsk State University. He is an author of more than 20 papers on automation of numerical parallel programs construction. He is one of main developers of fragmented programming system LuNA (Language for Numerical Algorithms). Professional interests include automation of numerical parallel programs construction, languages and systems of parallel programming, high performance computing.

Чмиль Александр Владимирович — магистрант факультета информационных технологий, Новосибирский национальный исследовательский государственный университет, e-mail: chmil.alexander@gmail.com.

Чмиль Александр Владимирович получил степень бакалавра в 2020 году в Новосибирском национальном исследовательском государственном университете на кафедре параллельных вычислений факультета информационных технологий. Тема выпускной квалификационной работы — « Разработка эффективных модулей исполнения фрагментированных программ частного вида для run-time системы фрагментированного программирования». Научные интересы — разработка систем балансировки нагрузки.

Chmil Alexander Vladimirovich — Master's student of the Faculty of Information Technology, Novosibirsk State University, e-mail: chmil.alexander@gmail.com.

Chmil Alexander Vladimirovich received a bachelor's degree in 2020 at the Novosibirsk State University at the Department of Parallel Computing of the Faculty of Information Technology. The topic of the final qualifying work is «Development of effective modules for the execution of fragmented programs of a particular type for a run-time system of fragmented programming». Research interests — development of load balancing systems.

Дата поступления — 08.11.2022