ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий
Кафедра параллельных вычислений

Разработка эффективных модулей исполнения фрагментированных программ частного вида для run-time системы фрагментированного программирования

Выполнил: Чмиль Александр Владимирович ФИТ НГУ

Научный руководитель: д.т.н., проф. Малышкин В. Э. зав. каф. ПВ ФИТ НГУ

Соруководитель: Перепёлкин В.А. научный сотрудник ИВМиМГ СО РАН

Новосибирск 2020

Необходимые определения

- Фрагмент данных – это переменная алгоритма. Он представлен в виде блока памяти.
- Фрагмент вычислений (ФВ) – исполняемая единица, имеющая в качестве входных и выходных значений заданные фрагменты данных.
- Балансировка нагрузки - метод распределения заданий между несколькими вычислительными узлами с целью оптимизации использования ресурсов, сокращения времени работы программы, а также горизонтального масштабирования программы.

Проблема

- При исполнении параллельных программ нередко возникает дисбаланс вычислительной нагрузки. Это приводит к тому, что программа неэффективно использует свои аппаратные ресурсы;
- для борьбы с дисбалансом используют балансировку вычислительной нагрузки;
- существует большое множество алгоритмов балансировки, имеющих свои достоинства и недостатки. Проблема заключается в том что ни один алгоритм балансировки не является универсальным.
- важно уметь автоматически применять алгоритм, который является удовлетворительным в конкретной ситуации.

LuNA

- LuNA - Экспериментальная система программирования, разрабатываемая в ИВМиМГ СО РАН и поддерживающая инструментально технологию фрагментированного программирования;
- LuNA предусматривает автоматический выбор частных алгоритмов балансировки. Но в системе нет алгоритма, который позволяет реагировать на точечный дисбаланс;
- существующий в LuNA алгоритм балансировки - Rope of beads [1] ориентирован на дисбаланс сразу на многих узлах. Но плохо подходит для устранения точечного дисбаланса, когда значительный дисбаланс возникает на отдельных узлах.

Цель работы

- Цель работы - разработка и реализация алгоритма динамической балансировки нагрузки в системе LuNA;
- задачи, которые нужно выполнить для достижения поставленной цели:
  - Исследование преимуществ и недостатков существующих алгоритмов;
  - поддержка динамической балансировки в языке LuNA;
  - разработка и реализация алгоритма балансировки в run-time системе LuNA;
  - экспериментальное исследование производительности доработанной системы LuNA.

Исследование алгоритмов

- Реализуемая мной динамическая балансировка нагрузки будет использоваться для точечной балансировки, в дополнении к алгоритму балансировки Rope-of-Beads;
- для точечной балансировки алгоритм должен удовлетворять следующим требованиям:
  - Низкое число коммуникаций;
  - низкое время реакции на дисбаланс.

Исследование алгоритмов

Исходя из требований было исследовано 3 наиболее подходящих алгоритма:

Алгоритм балансировки	+	-
Work requesting. Статья Scheduling parallel programs by work stealing with private deques [2].	Возможность передачи задач группами, коммуникация происходит только когда случается дисбаланс.	Алгоритм в чистом виде не учитывает близость данных.
Work dealing. Статья Work dealing (extended abstract) [3].	Возможность учитывать близость данных при балансировке, возможность передачи задач группами.	В зависимости от политики коммуникации загрузка на узлы либо не учитывается, либо для того чтобы актуализировать состояние загруженности узлов, требуются постоянные фоновые коммуникация.
Work stealing Статья Dynamic Load Balancing Using Work-Stealing [4].	Использование разделяемой памяти.	Большое количество синхронизаций при выполнении небольших задач.

Особенности LuNA

- Программа на языке LuNA состоит из фрагментов данных, фрагментов вычислений и их описаний. В описании фрагмента расположены различные рекомендации по обработке фрагментов - аннотации.
- Компилятор переводит программный код с языка LuNA в код на языке C++, в котором каждый ФВ представлен в виде отдельной функции - блока. Внутри этого блока могут создаваться другие блоки.
- В ходе работы программы фрагменты динамически порождаются и мигрируют. Выравнивание нагрузки осуществляется путем перераспределения фрагментов вычислений по узлам.

Особенности LuNA

Пример программы на языке LuNA

```
1.  /*
2.   Hello world example.
3.  */
4.
5.  import c_helloworld() as hello_world;
6.
7.  sub main()
8.  {
9.      hello_world() @ {
10.         locator_cyclic: 0;
11.     };
12. }
```

Пример блока скомпилированной программы

```
1.  // BI_EXEC: cf _l7: hello_world();
2.  BlockRetStatus block_2(CF &self)
3.  {
4.      if (self.migrate(CyclicLocator(0))) {
5.          return MIGRATE;
6.  }
7.  {
8.      // EXEC_EXTERN cf _l7: hello_world();
9.      c_helloworld();
10. }
11.     return EXIT;
12. }
```

Разработка

- Work-dealing алгоритм в зависимости от политики распределения задач либо не учитывает загруженность узлов, что, по сути, делает его аналогичным статической балансировке, либо учитывает, но для этого требуются постоянные коммуникации между процессами.
- Если рассматривать work-stealing алгоритм, то можно столкнуться с проблемой, что в системе LuNA узел имеет доступ только к своей очереди задач. Поэтому для реализации такого алгоритма потребуется использовать дополнительные механизмы коммуникации и синхронизации, как например разделяемая память. Что может негативно сказаться на производительность системы.
- Исходя из вышеперечисленных особенностей, мною был выбран алгоритм work-requesting. Этот алгоритм был выбран по причине того, что он лучше всего подходит под особенности LuNA и лучше других удовлетворяет требованиям к низкому числу коммуникаций и низкому времени реакции на дисбаланс.
- Для реализации алгоритма в runtime системе LuNA требуется:
  - Расширение языка LuNA аннотацией, позволяющей указывать фрагменты, участвующие в балансировке;
  - добавление в компилятор языка LuNA генерации кода внутри фрагмента вычислений, в котором происходит проверка наличия запросов ФВ от других узлов;
  - поддержка в runtime системе LuNA механизма для порождения и отправки запросов на балансировку.

Добавление специальной аннотации

Пример программы на языке LuNA

```
1.  /*
2.   Hello world example.
3.  */
4.
5.  import c_helloworld() as hello_world;
6.
7.  sub main()
8.  {
9.      hello_world() @ {
10.         locator_cyclic: 0;
11.         balance;
12.     };
13. }
```

Пример блока скомпилированной программы

```
1.  // BI_EXEC: cf _l7: hello_world();
2.  BlockRetStatus block_2(CF &self)
3.  {
4.      if (self.migrate(CyclicLocator(0))) {
5.          return MIGRATE;
6.      }
7.      if  (self.need_to_balance()) {
8.          return BALANCE;
9.      }
10. }
11. {
12.     // EXEC_EXTERN cf _l7: hello_world();
13.     c_helloworld();
14. }
15.     return EXIT;
16. }
```

Реализация – запрос фрагментов

- В случае отсутствия ФВ в очереди задач на исполнения производится запрос ФВ на некоторое количество узлов, заданное внутри runtime системы;
- при получении ФВ происходит отправка отмены запроса.

Реализация – отправка фрагментов

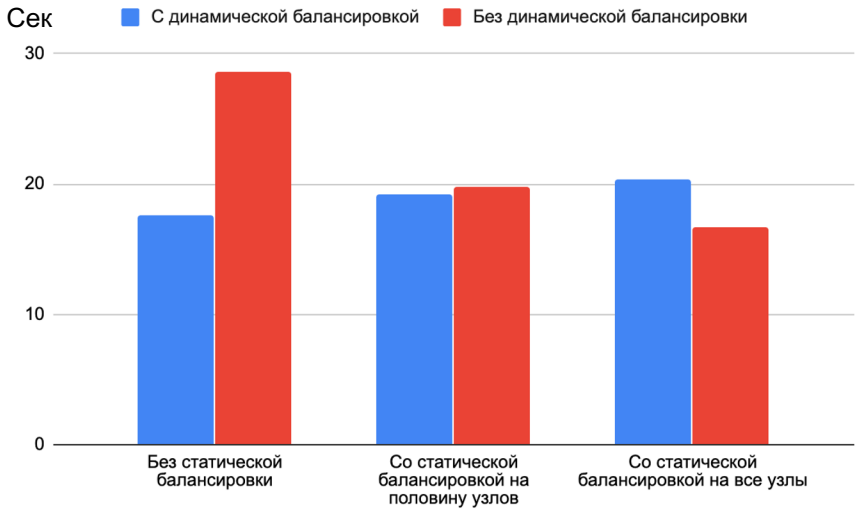
- При исполнении ФВ проверять, помечен ли он специальной аннотацией, есть ли запросы на передачу фрагментов и наличие задач на исполнение. В случае если условия выполняются – передавать фрагмент запросившему узлу.

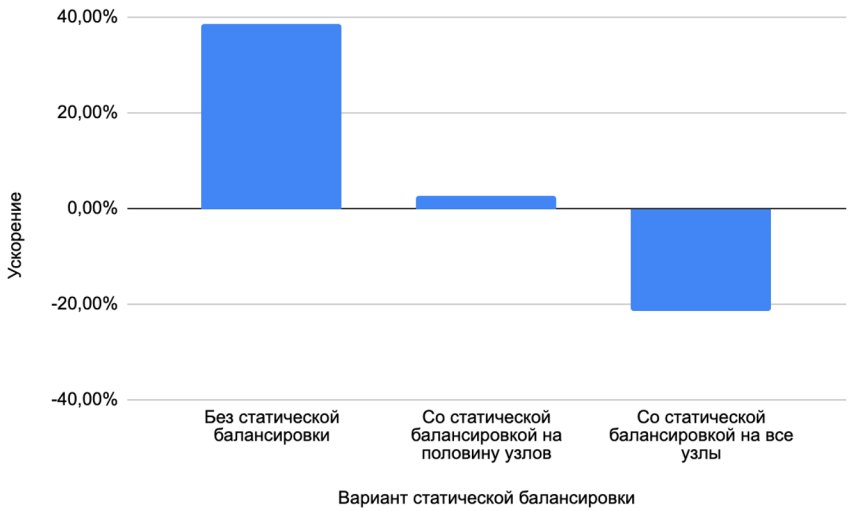
Тестирование

- Для тестирования используется программа умножения матриц;
- Цель тестирования - исследовать влияние динамической балансировки на скорость работы программы при различных условиях.
- Используются различные параметры программы:
  - Размер матрицы;
  - количество фрагментов, на которые разбивается матрица.
- Используются различные вариации программы:
  - Без статической балансировки;
  - с частичной статической балансировкой(задачи распределяются на половину узлов);
  - с полной статической балансировкой.

Результаты тестирования

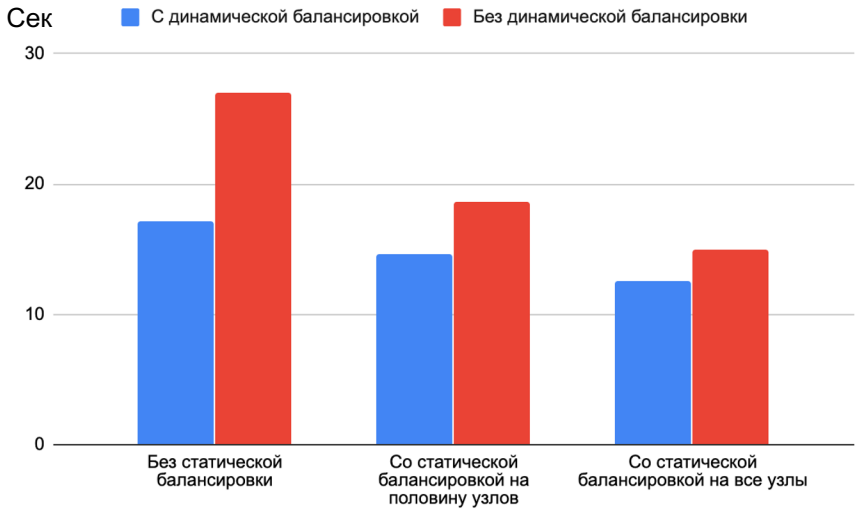
Параметры теста: Размер фрагмента: 100. Количество фрагментов: 40.

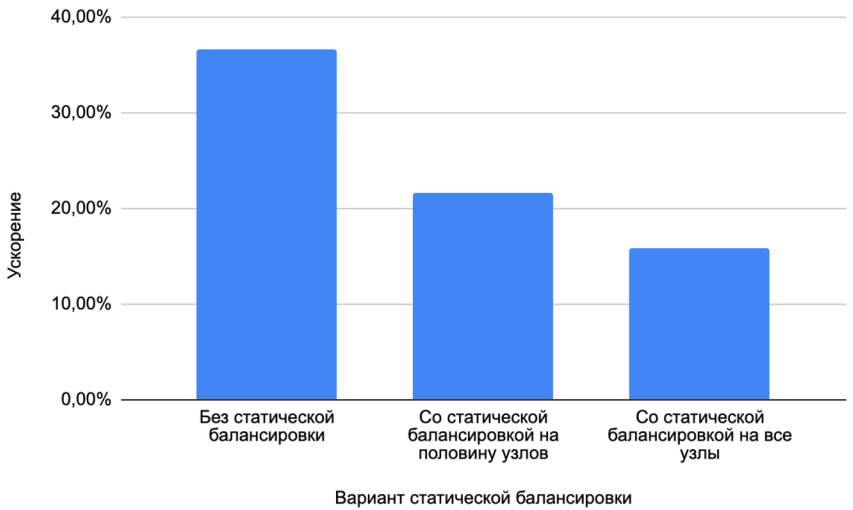




Результаты тестирования

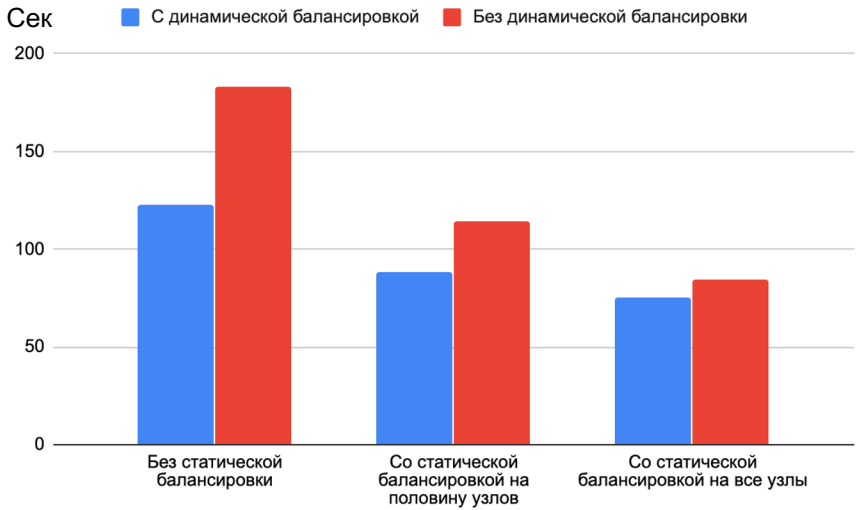
Параметры теста: Размер фрагмента: 200. Количество фрагментов: 20.

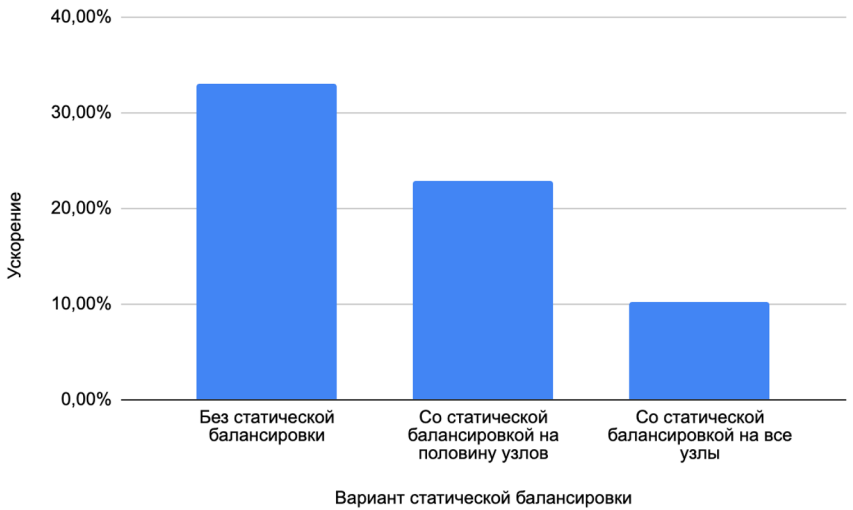




Результаты тестирования

Параметры теста: Размер фрагмента: 200. Количество фрагментов: 40.





Заключение

- В результате проведенной работы были достигнуты следующие результаты:
  - Алгоритм work requesting адаптирован для использования в runtime системе LuNA;
  - расширен язык LuNA для того, чтобы разработчики могли сами указывать, должен ли участвовать фрагмент в балансировке;
  - расширение было поддержано в runtime системе с использованием адаптированного алгоритма;
  - проведено исследование реализуемых алгоритмов.
- В дальнейшем планируется развитие динамического балансировщика нагрузки. Возможными направлениями для развития являются:
  - Настройка параметров балансировки, в том числе автоматическая;
  - реализация различных паттернов запроса ФВ.

Апробация работы

Данная работа была опубликована на 58-й Международной научно-студенческой конференции, г. Новосибирск, 2020 г. и была награждена дипломом второй степени.

Публикация по теме:

Чмиль А.В. Разработка динамического балансировщика нагрузки для runtime-системы LuNA // Информационные технологии : Материалы 58-й Междунар. науч. студ. конф. 10–13 апреля 2020 г. / Новосиб. гос. ун-т. — Новосибирск : ИПЦ НГУ, 2020. — 48

Список литературы

1. 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. – С. 80-85.
2. Acar U. A., Charguéraud A., Rainey M. Scheduling parallel programs by work stealing with private deques //ACM SIGPLAN Notices. – ACM, 2013. – Т. 48. – №. 8. – С. 219-228.
3. Hendler D., Shavit N. Work dealing //Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures. – ACM, 2002. – С. 164-172.
4. Cederman D., Tsigas P. Dynamic load balancing using work-stealing //GPU Computing Gems Jade Edition. – Morgan Kaufmann, 2012. – С. 485-499.