МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

Разработка подсистемы прямого управления исполнением фрагментированных программ в системе LuNA

Подготовил: студент 4 курса ФИТ НГУ Пирожков Андрей Константинович

Руководитель: к.т.н., доцент каф. ПВ ФИТ НГУ Маркова Валентина Петровна

Соруководитель: ст. преп. каф. ПВ ФИТ НГУ Перепёлкин Владислав Александрович

Рецензент: к.ф.-м.н., зав.каф. информационной безопасности КемГУ Карабцев Сергей Николаевич

Дата защиты:
23.06.23

Актуальность

- Автоматическое конструирование параллельных программ является трудно решаемой задачей. Выбор эффективного варианта исполнения параллельной программы важен, чтобы избежать дополнительных затрат на энергию, память и время на разработку.
- Автоматическое конструирование осуществляется двумя основными способами:
  - Первый способ заключается в использовании исполнительной системы при конструировании параллельных программ.
  - Второй способ заключается в генерировании параллельного кода без использования исполнительной системы.

Актуальность

- Проблема конструирования параллельной программы – проблема определения прямого управления.
- Средства прямого управления — элементы конструкций, которые являются некоторой частью фрагментированной программы, отображенный на конкретный ресурс с конкретным управлением.
- Осуществление прямого управления – подсистема прямого управления.
- Система фрагментированного программирования LuNA является примером, на которой можно рассмотреть проблему эффективного конструирования параллельных программ.

Цель и задачи

Цель: разработать подсистему автоматического конструирования традиционных императивных параллельных LuNA-программ без использования исполнительной системы.

Задачи:

- Разработка языковых средств описания параллельной программы.
- Разработка структур данных и алгоритмов, обеспечивающих конструирование параллельных программ.
- Экспериментальное исследование эффективности конструируемых параллельных программ.

Обзор

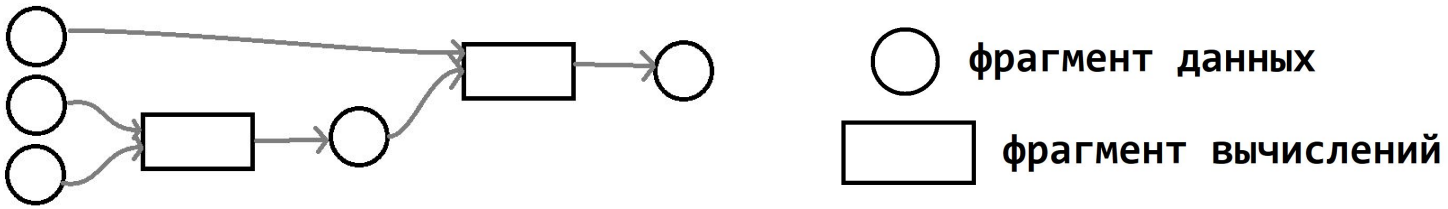
Существует 2 основных подхода решения проблемы:

1. Создание параллельной программы из последовательной. Примеры инструментов: ParaWise, Polaris, Cetus, Par4All, CAPO.
2. Использование систем автоматического конструирования. Примеры инструментов: Charm++, Chapel, X10, Titanium, Fortress, LuNA.

Были рассмотрены технологии с помощью которых работают все вышеперечисленные инструменты.

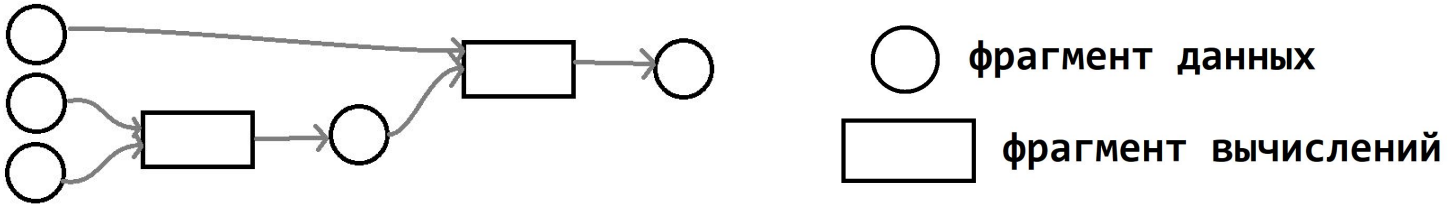
Фрагментированное программирование

- Во фрагментированной LuNA-программе вычисления рассматриваются как совокупность фрагментов данных и вычислений.
- Фрагменты данных — это блок данных, который используется в фрагментированной программе как входной или выходной параметр для фрагмента вычислений.



Фрагментированное программирование

- Фрагменты вычислений — это объект времени исполнения, который описан в фрагментированной программе, который содержит описание входных и выходных фрагментов данных. Фрагменты вычислений выполняются по готовности данных.
- В общем случае фрагментированная программа – это сложный объект.
- В рамках этой работы фрагментированная программа состоит из конечного количества объектов.



Постановка задачи

- Разработать входное представление фрагментированной программы.
- Разработать языковые средства описания программы.
- Разработать конструкции для реализации прямого управления.
- Алгоритм конструирования должен построить из конструкций параллельную программу.
- Рассмотреть вопрос эффективности генерируемых программ.

Предлагаемая структура входного языка

- Входные и выходные фрагменты данных, количество используемых процессов.
- Перечисление фрагментов вычислений с подробным их описанием.
- Перечисление всех фрагментов данных и переменных с указанием типов.
- Указание переменных по умолчанию (define).

```
1   {
2     "using": [ "funcs.json" ],
3     "interface": {
4       "input": [ "size_vector", "value" ],
5       "output": [ "c_0", "c_1", "c_2", "c_3", "size_vector" ],
6       "ranks": "4"
7     },
8     "operations": {
9       "c_initVector_a_0": [...],
20      "c_initVector_a_1": [...],
31      "c_initVector_a_2": [...],
42      "c_initVector_a_3": [...],
53      "c_initVector_b_0": [...],
64      "c_initVector_b_1": [...],
75      "c_initVector_b_2": [...],
86      "c_initVector_b_3": [...],
97      "c_sumVectors_0": {
98        "function_type": "intermediate",
99        "code": "c_sumVectors",
100       "inputs": [ "a_0", "b_0", "size_vector" ],
101       "outputs": [ "c_0" ],
102       "arguments": {
103         "vector1": "a_0",
104         "vector2": "b_0",
105         "result": "c_0",
106         "size": "size_vector"
107       }
108     },
109     "c_sumVectors_1": [...],
121     "c_sumVectors_2": [...],
133     "c_sumVectors_3": [...],
145     "init_args": [...],
155     "finish_program": [...]
168   },
169   "variables": {
170     "a_0": "df",
171     "a_1": "df",
172     "a_2": "df",
173     "a_3": "df",
174     "b_0": "df",
175     "b_1": "df",
176     "b_2": "df",
177     "b_3": "df",
178     "c_0": "df",
179     "c_1": "df",
180     "c_2": "df",
181     "c_3": "df",
182     "size_vector": "int",
183     "value": "real"
184   }
185 }
186
```

Предлагаемые конструкции

- ProgSpec – описание множества фрагментов вычислений и фрагментов данных.
- Mcell – ячейка памяти, описывающая используемые фрагменты данных.
- Port – описывает аргументы, применяемые операцией.
- Sniplet – это конструкция, предназначенная для выполнения операций:
  - ArgvSniplet – сниплет ввода из переданных аргументов в программе.
  - VarToStdoutSnip – сниплет вывода в консоль.
  - InvokeOperationSnip – сниплет подстановки операции.
  - SendRecvSniplet – коммуникационный сниплет.
- PartialProgram – частично-определенная программа.

Алгоритм конструирования программ

Генерируемая программа конструируется в несколько этапов:

- Построение программы без коммуникаций:
  - Критерий завершенности конструирования программы.
  - Шаг 1 – в конструкции сниплетов порты строят связи с ячейками памяти. Конструкции ячеек памяти создаются, если они ещё не были созданы.
  - Шаг 2 – если существует неинициализированная ячейка памяти, то создаётся сниплет, в которой эта ячейка памяти инициализируется.
- Добавление коммуникаций в построенной программе.
- Генерация кода для построенной программы.

Эффективность конструируемых программ

- Пользователь может самостоятельно управлять распределением фрагментов вычислений по процессам с помощью входного языка.
- Предложенная оценка качества параллельного кода может быть основана на 2 критериях:
  - Количество коммуникаций между узлами – чем меньше коммуникаций, тем быстрее работает программа и меньше расходуется памяти.
  - Распределение фрагментов вычислений между узлами – равномерное распределение снижает вероятность простоев и обеспечивает более равномерную нагрузку на оборудование.
- Применение метода градиентного спуска.

Тесты на простых примерах

- Сгенерированный код программы должен компилироваться и выполняться без ошибок при верном формате входного представления.
- Проверено конструирование для разного количество процессов, а также работа всех разработанных конструкций.

```
41  int main(int argc, char **argv) {
42      int rank, size;
43      MPI_Init(&argc, &argv);
44      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
45      MPI_Comm_size(MPI_COMM_WORLD, &size);
46
47      DF c_0;
48      DF c_1;
49      DF c_2;
50      DF c_3;
51      int size_vector;
52      double value;
53      DF a_3;
54      DF b_3;
55      DF a_2;
56      DF b_2;
57      DF a_1;
58      DF b_1;
59      DF a_0;
60      DF b_0;
61      c_initVector_params(&size_vector, &value);
62      if (rank == 1) c_initVector(b_2, size_vector, value);
63      if (rank == 1) c_initVector(a_1, size_vector, value);
64      if (rank == 3) c_initVector(b_1, size_vector, value);
65      if (rank == 2) c_initVector(a_3, size_vector, value);
66      if (rank == 0) c_initVector(a_0, size_vector, value);
67      if (rank == 2) c_initVector(b_3, size_vector, value);
68      if (rank == 2) c_initVector(b_0, size_vector, value);
69      if (rank == 0) c_initVector(a_2, size_vector, value);
70      send_recv_df(a_0, 0, 2, rank, 584);
71      if (rank == 2) c_sumVectors(a_0, b_0, c_0, size_vector);
72      send_recv_df(c_0, 2, 0, rank, 817);
73      send_recv_df(a_1, 1, 2, rank, 392);
74      send_recv_df(b_1, 3, 2, rank, 643);
75      if (rank == 2) c_sumVectors(a_1, b_1, c_1, size_vector);
76      send_recv_df(c_1, 2, 0, rank, 675);
77      send_recv_df(a_2, 0, 2, rank, 22);
78      send_recv_df(b_2, 1, 2, rank, 892);
79      if (rank == 2) c_sumVectors(a_2, b_2, c_2, size_vector);
80      send_recv_df(c_2, 2, 0, rank, 333);
81      send_recv_df(a_3, 2, 1, rank, 142);
82      send_recv_df(b_3, 2, 1, rank, 702);
83      if (rank == 1) c_sumVectors(a_3, b_3, c_3, size_vector);
84      send_recv_df(c_3, 1, 0, rank, 174);
85      if (rank == 0) c_finilize(c_0, c_1, c_2, c_3, size_vector);
86
87      MPI_Finalize();
88  }
```

Тестирование на практически значимом примере

Практически значимый пример: решение уравнения Пуассона в трехмерной области методом Якоби с одномерной декомпозиции.

Результаты тестирования на 4 узлах по 8 процессов каждый (всего 32 MPI-процесса):

- Сгенерированная программа с неэффективным распределением — 57.158 с.
- Сгенерированная программа с эффективным распределением — 38.854 с.
- LuNA-программа эффективным распределением по узлам — 44.892 с.

Тестирование проводилось на ИВЦ НГУ.

Заключение

В результате работы была разработана подсистема прямого управления исполнением фрагментированных программ. В ходе работы были выполнены все поставленные задачи тем самым была достигнута цель работы.

Защищаемые положения:

- Разработаны языковые средства описания фрагментированной программы.
- Разработана структура данных и реализованы алгоритмы конструирования, обеспечивающие генерацию параллельного кода.
- Проведены экспериментальные исследования, в которых сравнивалась работоспособность и эффективность сгенерированных программ.

Заключение

Разработанное средство автоматического конструирования может иметь следующие направления развития:

- Автоматическая генерация входного представления для генератора параллельных фрагментированных программ.
- Увеличение количества конструкций для расширения класса поддерживаемых задач.
- Реализация в генераторе алгоритмов автоматического улучшения эффективности сконструированной параллельной программы.

Заключение

Материалы 61-ой международной научной студенческой конференции МНСК-2023: Информационные технологии / Новосибирск, 2023.

Тема доклада: “Разработка подсистемы прямого управления исполнением фрагментированных программ в системе LuNA”. (Пирожков А.К.)

Заключение

Спасибо за внимание!