Разработка и реализация системных алгоритмов эффективного исполнения фрагментированных программ на мультикомпьютере

Докладчик: Беляев Н. А., аспирант 1 курса ИВМиМГ СО РАН

Научный руководитель: Малышкин В. Э., д. т. н., проф.

Введение

- Решение задач численного моделирования с использованием суперкомпьютеров требует разработки соответствующих параллельных программ
- Разработка параллельных программ для решения задач численного моделирования зачастую требует от разработчика навыков системного параллельного программирования

Введение

- Системное параллельное программирование не является частью предметной области
- Абстрагирование разработчика параллельной программы для решения задач численного моделирования от решения задач системного программирования позволит сократить время разработки параллельных программ

Требования к системам параллельного программирования

- Абстрагирование разработчика параллельной программы от задач системного параллельного программирования
- Обеспечение уровня производительности параллельных сравнимого с уровнем, достигаемым при ручной реализации программ

Обзор существующих средств и систем параллельного программирования

MPI

- Стандарт и библиотека ПП, предоставляющая возможность организовывать взаимодействие параллельных посредством передачи сообщений
- Является низкоуровневым средством ПП

OpenMP (OpenACC)

- Стандарт, описывающий набор директив компилятора, позволяющих “распараллеливатть” циклы
- Работает в общей памяти
- OpenACC поддерживает работу с GPU
- Отсутсвует поддержка динамических свойств программы

DVM-H

- Стандарт, описывающий набор директив компилятора, позволяющих “распараллеливатть” циклы
- Работает в распределенной памяти, поддерживаются GPU, Xeon PHI
- Возможно обеспечение динамической балансировки вычислительной нагрузки на узлы мыльтикомпьютера

OpenCL

- Открытый стандарт параллельного программирования
- Программа представляется в виде множества задач, записанных на встроенном си-подобном языке (kernel)
- Взаимодействие между задачами, а также, управление памятью ложится на плечи разработчика ПП
- Программирование коммуникаций в распределенной памяти ложится на плечи разработчика ПП

Charm++

- Программа представляется в виде множества взаимодействующих процессов
- Динамические свойства программы обеспечиваются системой
- Структура программы не учитывается, используются универсальные системные алгоритмы
- Представление численного алгоритма в виде множества взаимодействующих объектов не всегда удобно и очевидно

Система LuNA

Общие сведения

- Система LuNA – это система автоматического конструирования параллельных программ, ориентированная на решение задач численного моделирования на суперкомпьютерах

Параллельная программа в системе LuNA

- ПП в системе LuNA представляется я виде множества фрагментов вычисления (ФВ) и фрагментов данных (ФД)
- Также ПП в системе LuNA может быть представлена в виде двудольного ориентированного графа, вершинами которого являются ФВ и ФД, а дуги обозначают информационные зависимости

Термины и определения

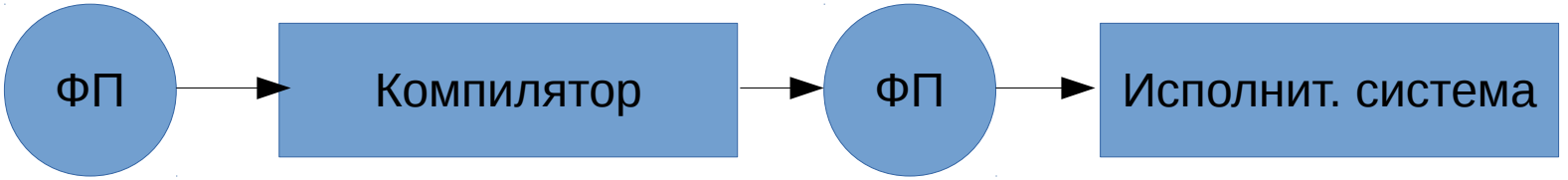
- Управление (на множестве ФВ) – отношение частичного порядка (на множестве ФВ), накладывающее ограничения на порядок исполнения ФВ

Исполнение фрагментированной программы

- Исполнение ФВ в системе LuNA происходит по т. н. базовому алгоритму, ФВ исполняется по готовности значений всех своих входных ФД
- В языке LuNA присутствуют ФВ двух видов:
  - Атомарные ФВ, представляющие собой вызовы процедур языка Си
  - Структурированные ФВ (if, for, while, sub) – представляют собой ФВ, исполнение которых заключается в поступлении на исполнение некоторого множества ФВ, называемого телом структурированного ФВ. Исполнение структурированного ФВ в дальнейшем условно будем называть “раскруткой” ФВ

Архитектура системы LuNA

- Система состоит из компилятора языка LuNA и исполнительной системы



Недостатки (проблемы) системы LuNA

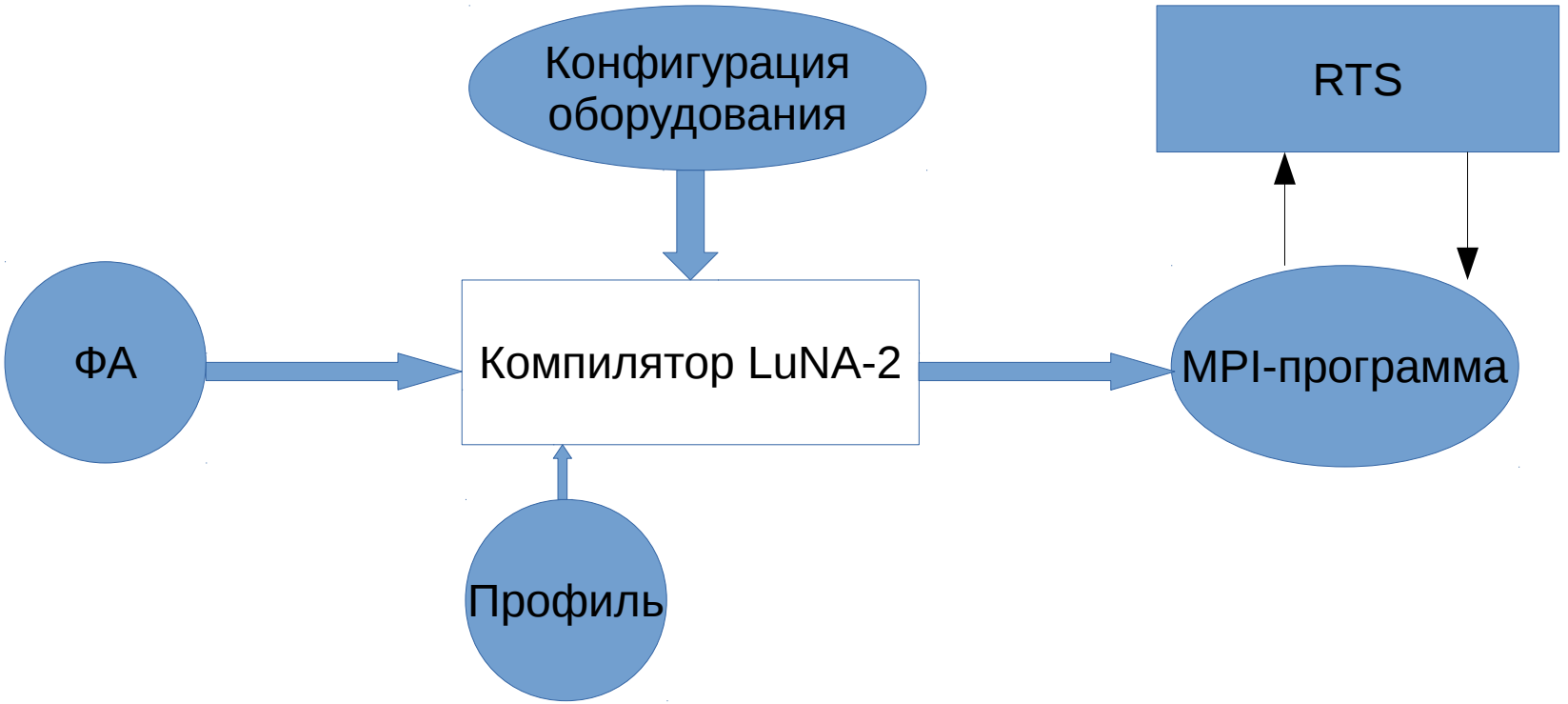
- Неконтролируемая “раскрутка” структурированных ФВ во время исполнения ФП
- Неэффективные алгоритмы распределения фрагментов по узлам мультикомпьютера.
- Проблема «сборки мусора»
- Использование универсальных алгоритмов динамической балансировки вычислительной нагрузки на узлы мультикомпьютера
- Решения о выборе управления на подмножествах ФВ принимается динамически, что снижает производительность системы

LuNA-2

LuNA-2

- Для решения описанных проблем системы LuNA разрабатывается система LuNA-2
- Система LuNA-2 должна автоматически конструировать параллельные программы, обладающие свойством настраиваться на конкретный вычислитель (в т. ч., свойством обеспечения при необходимости динамической балансировки вычислительной нагрузки) для заданного класса численных алгоритмов
- Производительность сконструированных программ не должна быть хуже таковой у программ, разработанных “вручную” более чем в 2 раза

Ожидаемый результат



Цель работы

- Разработка системных алгоритмов эффективного исполнения фрагментированных программ на мультикомпьютере для заданного класса задач численного моделирования и реализация алгоритмов в виде средства параллельного программирования в рамках системы LuNA

Задачи

Разработать алгоритмы распределения фрагментов данных и фрагментов вычислений на узлы мультикомпьютера

Разработать алгоритмы анализа ФА для выбора походящего алгоритма распределения фрагментов на узлы мультикомпьютера

Разработать алгоритмы выявления в ФА подмножеств ФВ, описывающих “примитивы” параллельного программирования (редукция, барьер, и т. д.)

Разработать алгоритм анализа результатов измерения производительности системы во время предыдущих запусков фрагментированной программы (ФП) (т. н. профиля исполнения ФП)

Разработать алгоритмы динамического перераспределения фрагментов на узлы мультикомпьютера (алгоритмы динамической балансировки вычислительной нагрузки)

Задачи

Разработать алгоритмы выбора алгоритма динамической балансировки для множеств фрагментов, при исполнении которых наблюдается дисбаланс вычислительной нагрузки

Разработка алгоритмов своевременного освобождения памяти, занимаемой значениями ФД, уже потребленными всеми ФВ, для которых данное значение было входным (алгоритмы «сборки мусора»)

Разработка алгоритмов выбора управления, согласно которому будет осуществлено исполнение фрагментов на узлах мультикомпьютера. При выборе управление алгоритмы должны по возможности сокращать общее время исполнения ФП и потребление ресурсов мультикомпьютера во время исполнения. В частности, необходимо, по возможности, выбрать управление, позволяющее максимально быстро освобождать память, занимаемую значениями ФД

Текущий результат

- Язык LuNA расширен для повышения уровня программирования и облегчения задачи распознавания подмножеств ФВ, для которых возможна генерация эффективной программы

Новые конструкции языка LuNA

- Обмен границами: “<borders_exchange N>(…)
- Редукция : <reduction op>(…)
- Распределенный массив ФД: “DFArray name[Dim1]...[DimN]

Сравнение версий языка LuNA

```
sub update_shadow(name data_prev, name params, name data) {
    df x_left, x_right, y_bottom, y_top, stub, flag;

    for i = 0 .. $FX - 1
    for j = 0 .. $FY - 1 {
                    cf rs[i][j] : cf_read_shadow(data_prev[i][j], params[i][j],
                    x_left[i][j], x_right[i][j],
                    y_bottom[i][j], y_top[i][j],
                            flag[i][j]);

                    cf stb[i][j] : cf_set_empty(stub[i][j]);
            }

            // For inner fragments
    for i = 1 .. $FX - 2
    for j = 1 .. $FY - 2 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            x_right[i - 1][j], x_left[i + 1][j],
                            y_top[i][j - 1], y_bottom[i][j + 1], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_right[i - 1][j], x_left[i + 1][j],
                            y_top[i][j - 1], y_bottom[i][j + 1]);
            }

            // For j = 0 and j = $FY - 1
    for i = 1 .. $FX - 2 {
                    let j = 0 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            x_right[i - 1][j], x_left[i + 1][j],
                            stub[i][j], y_bottom[i][j + 1], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_right[i - 1][j], x_left[i + 1][j], y_bottom[i][j + 1]);
                    }

                    let j = $FY - 1 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            x_right[i - 1][j], x_left[i + 1][j],
                            y_top[i][j - 1], stub[i][j], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_right[i - 1][j], x_left[i + 1][j], y_top[i][j - 1]);
                    }
            }

            // For i = 0 and i = $FX - 1
    for j = 1 .. $FY - 2 {
                    let i = 0 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            stub[i][j], x_left[i + 1][j],
                            y_top[i][j - 1], y_bottom[i][j + 1], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_left[i + 1][j], y_top[i][j - 1], y_bottom[i][j + 1]);
                    }

                    let i = $FX - 1 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            x_right[i - 1][j], stub[i][j],
                            y_top[i][j - 1], y_bottom[i][j + 1], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_right[i - 1][j], y_top[i][j - 1], y_bottom[i][j + 1]);
                    }
            }

            // For corners
            let i = 0, j = 0 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            stub[i][j], x_left[i + 1][j],
                            stub[i][j], y_bottom[i][j + 1], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_left[i + 1][j], y_bottom[i][j + 1]);
            }

            let i = 0, j = $FY - 1 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            stub[i][j], x_left[i + 1][j],
                            y_top[i][j - 1], stub[i][j], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_left[i + 1][j], y_top[i][j - 1]);
            }

            let i = $FX - 1, j = 0 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            x_right[i - 1][j], stub[i][j],
                            stub[i][j], y_bottom[i][j + 1], flag[i][j], data[i][j])
                    -->
                    (data_prev[i][j], x_right[i - 1][j], y_bottom[i][j + 1]);
            }

            let i = $FX - 1, j = $FY - 1 {
                    cf ws[i][j] : cf_write_shadow(data_prev[i][j], params[i][j],
                            x_right[i - 1][j], stub[i][j],
```

```
sub main()
{
    df n, sz, iter, eps;
    DFArray A[n][n];
    init(n, sz);
    while (it < 10000 && eps >= 0.0001), it = 0 .. out iter : <A(it) --> A(it+1)>
    {
        DFArray D[n][n];
        for i = 0 .. n-1 : <A(it)[i][j] --> B(it)[i][j]>
        {
            for j = 0 .. n-1
                cf clc[i][j] : cf_calc(A(it)[i][j], B(it)[i][j], n, sz);
        }
        <borders_exchange 2>(B(it), A(it+1), cf_read(%in, n, sz, %1, %2, %3, %4),
            cf_write(%out, n, sz, %1, %2, %3, %4));
        for i = 0 .. n-1 : <A(it+1)[i][j] --> D[i][j]>, <B(it)[i][j] --> D[i][j]>
        {
            for j = 0 .. n-1
                cf diff[i][j] : cf_calc_diff(A(it+1)[i][j], B(it)[i][j], D[i][j], n, sz);
        }
        <reduce max>(D, eps, cf_reduce_max(%array, %out, n, sz));
    }
    for i = 0 .. n-1 : <A(iter)[i][j]>
    {
        for j = 0 .. n-1
            cf print[i][j] : cf_print(A(iter)[i][j], n, sz);
    }
}
```

Пример программы

```
sub main()
{
    df n, sz, iter, eps;
    DFArray A[n][n];
    init(n, sz);
……….
    while (it < 10000 && eps >= 0.0001), it = 0 .. out iter : <A(it) --> A(it+1)>
    {
        DFArray D[n][n];
        for i = 0 .. n-1 : <A(it)[i][j] --> B(it)[i][j]>
        {
            for j = 0 .. n-1
                cf clc[i][j] : cf_calc(A(it)[i][j], B(it)[i][j], n, sz);

        }
        <borders_exchange 2>(B(it), A(it+1), cf_read(%in, n, sz, %1, %2, %3, %4),
            cf_write(%out, n, sz, %1, %2, %3, %4));
        for i = 0 .. n-1 : <A(it+1)[i][j] --> D[i][j]>, <B(it)[i][j] --> D[i][j]>
        {
            for j = 0 .. n-1
                cf diff[i][j] : cf_calc_diff(A(it+1)[i][j], B(it)[i][j], D[i][j], n, sz);
        }
        <reduce max>(D, eps, cf_reduce_max(%array, %out, n, sz));
    }
    for i = 0 .. n-1 : <A(iter)[i][j]>
    {
        for j = 0 .. n-1
            cf print[i][j] : cf_print(A(iter)[i][j], n, sz);
    }
```

Спасибо за внимание