SAMARA UNIVERSITY

Востокин С.В.

Самарский национальный исследовательский университет
имени академика С.П. Королева, Самара

ИСПОЛЬЗОВАНИЕ КОМПЛЕКТА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ TEMPLET ДЛЯ ИНТЕГРАЦИИ ОБЛАЧНЫХ РЕСУРСОВ

Национальный Суперкомпьютерный Форум (НСКФ-2022)
ИПС имени А.К. Айламазяна РАН, 29 ноября – 02 декабря 2022 года

1. **Мотивация разработки Templet SDK.**

2. **Особенности комплекта разработки, их роль в интеграции ресурсов:**
   - модель вычислений;
   - генерация кода по модели;
   - библиотека среды выполнения.

3. **Опыт использования комплекта разработки для интеграции ресурсов:**
   - инфраструктура разработки приложений в облаке;
   - приложение А - анализ динамических систем;
   - приложение Б - обработка данных.

4. **Перспективы развития Templet SDK.**

Ресурсоемкое (по данным и/или вычислениям) **приложение должно уметь выполняться на произвольной совокупности доступных в данный момент** (как и в проекте BOINC, а не специализированных) **вычислительных ресурсов,**

но при этом возможности приложения не должны ограничиваться случаем чрезвычайно параллельных вычислений (можно программировать нетривиальные параллельные алгоритмы).

В ЧЕМ ВАЖНОСТЬ ЭТОГО СВОЙСТВА ПРИЛОЖЕНИЯ?

- Низкая себестоимость вычислений на простаивающих ресурсах.
- Возникает возможность получить (и использовать в одной сложной задаче) большое количество ресурсов за счет их интеграции.

- Персональные компьютеры добровольцев, как в проекте BOINC или в других проектах добровольных распределенных вычислений.
- Временно простаивающие корпоративные персональные компьютеры, которые потенциально доступны по сети для решения производственных вычислительных задач.
- Временно свободные вычислительные узлы суперкомпьютерных или кластерных систем большой производительности.
- Бесплатные или недорогие виртуальные машины, предоставляемые облачными провайдерами.

Конечно, программы должны эффективно выполняться и на «обычных» многопроцессорных системах с общей и распределенной памятью.

- Проблема **не сколько в получении доступа** к перечисленной аппаратуре, а в умении эффективно ее использовать в приложениях.

- Не преуменьшая важности промежуточного ПО в решении данной проблемы, мы делаем акцент на том, **что само приложение должно быть устроено определенным образом,** в том числе, чтобы работать с используемым для интеграции ресурсов ППО.

Общий подход к построению приложений, умеющих выполняться на произвольной совокупности ресурсов, известен: **нужно разбить вычисления на большое число автономно выполняющихся задач** (это позволяет реализовывать механизмы отказоустойчивости и балансировки нагрузки).

Как описывать взаимосвязи задач?

Традиционные подходы:

- задачи не имеют взаимосвязей;
- используются языки описания потоков работ (заданные через API или специальный язык типа CWL);
- не используется систематический подход описания, главное – ППО управления запуском задач;
- решения на основе алгоритмических скелетов с заданными в них моделями исполнения, похоже на MapReduce и др.;
- ..

Наш подход – **использовать модель акторов Хьюитта** в новой интерпретации, адаптированной **для управления задачами**:

- система акторов занимается синхронизацией задач;
- задача - часть изолированного шага актора.

Семантика параллельного протекания вычислительного процесса описывается в терминах **распараллеливания последовательных недетерминированных алгоритмов специальной структуры**:

- не требуется концепция локального состояния актора и можно считать, что вычисления выполняются в разделяемой (shared) памяти;
- акторы и сообщения – сущности, моделируемые “внутренними” переменными модели и специальными процедурами доступа к ним;
- так как модель – это алгоритм (с некоторыми допущениями), не требуется специального синтаксиса для выражения ее семантики;
- семантика модели – это соглашение о расширенной интерпретации последовательного алгоритма, при этом обычная интерпретация включается в расширенную.

Примечание: такой же принцип используется в описаниях итеративного и рекурсивного параллелизма.

Структура алгоритмического скелета акторных алгоритмов (АМ)

```
Algorithm( run() ) ->

AM<actor,message>(
    init(){set(message,actor,boolean)}
    recv(message,actor){
        send(message,actor)
        access(message)
    }
)
```

init – инициализация вычислений

recv – обозначение действий, выполняемых при поступлении сообщения в актор

set – активация акторов при помощи сообщений (запуск исполнения)

send – отправка сообщения в произвольный актор

acccess – определение доступности сообщения в контексте данного вызова (доступ только к неактивному сообщению)

run

```
начало
    выполнить init()
    пока есть такие m, что m.active = true выполнять
        m.active := false
        m.act.active := true
        выполнить recv(m, m.act)
        m.act.active := false
конец
```

set

```
вход: mes, act, activity

начало
    mes.act := act
    mes.active := activity
конец
```

send

```
вход: mes, act

начало
    mes.active := true
    mes.act := act
конец
```

access

```
вход: mes

начало
    вернуть mes.active = false ∧ mes.act.active = true
конец
```

Источник: Бобылева И.В. Метод попарной обработки элементов информационных массивов для многозадачных вычислений в гибридном облаке, дис. на соискание уч. степ. канд. техн. наук по спец. 2.3.5 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, защищена 25.05.2022.

Проблема выбранного нами подхода – **требуется специальный метод программирования**: код в `recv(_,_)` внутри цикла «пока» на слайде 8 желательно структурировать.

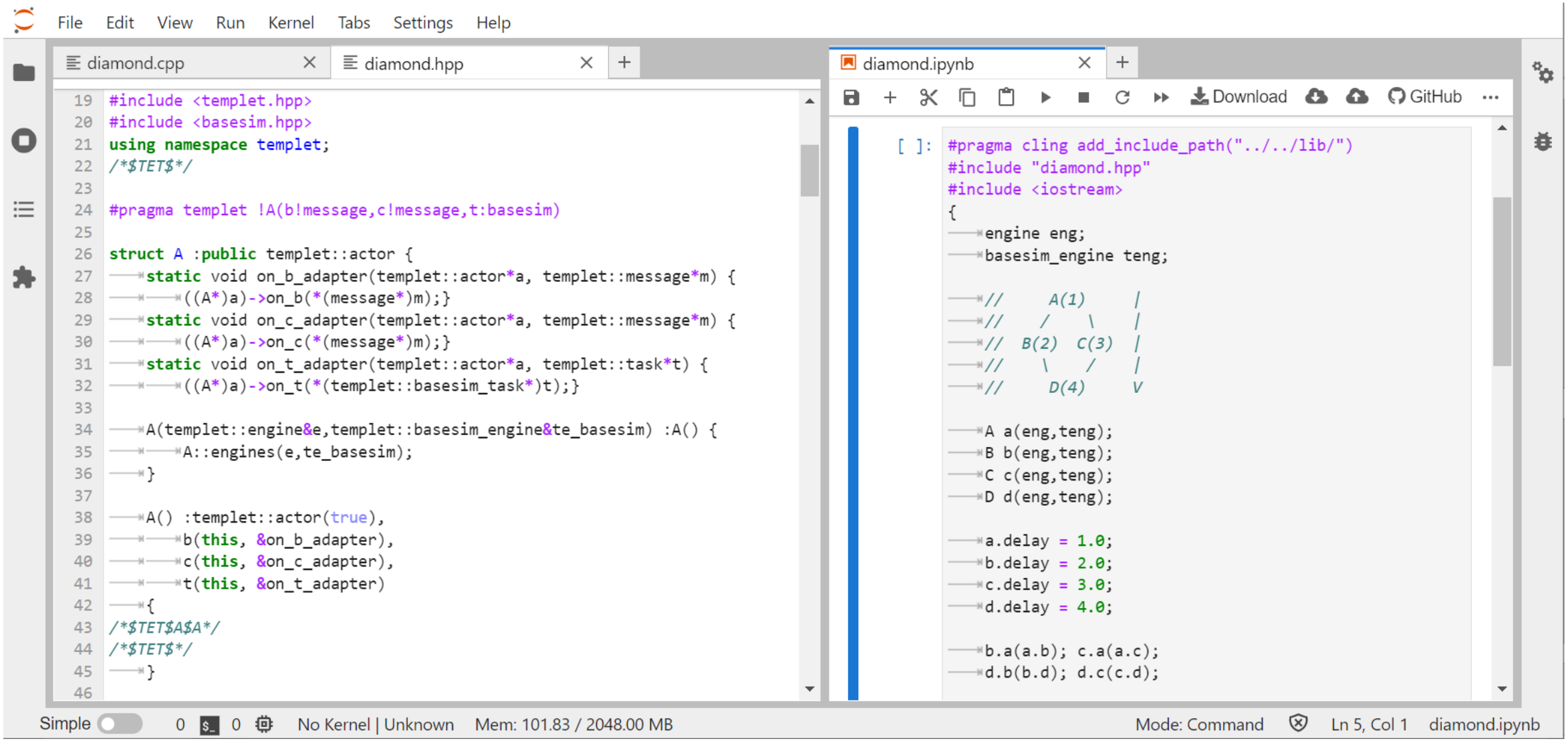
Решение:

- методология программирования – на основе алгоритмических скелетов Коула;
- инструментальная поддержка – **метапрограммирование на основе препроцессора**, автоматическая генерация и поддержание требуемой моделью вычислений структуры кода приложения.

Особенность: все алгоритмические скелеты объединены общей моделью выполнения.

Преимущество: компактный, понятный и надежный «движок» для модели, так как за привязку прикладного кода к «движку» отвечает генератор кода.

АНИМАЦИЯ РАБОТЫ ПРЕПРОЦЕССОРА



Библиотека среды выполнения Templet SDK **выполняет синхронизацию задач с использованием акторов**.

Из этого следует следующий дизайн «движка».

- Задачи - активные, акторы - пассивные, потоки управления связаны с задачами.
- Библиотека среды выполнения состоит из двух частей:
  - части, управляющей акторами, которая не зависит от используемого для интеграции ресурсов ППО;
  - части, управляющей задачами, которая зависит от используемого ППО.
- Можно иметь несколько типов задач различающихся способом выполнения и несколько типов ППО – это удобно для интеграции ресурсов.

Примеры реализованных в Templet SDK «движков» управления задачами:

- последовательное выполнение кода задач в общем с кодом акторов адресном пространстве для логической отладки (**base**);
- выполнение кода задач в общем с кодом акторов адресном пространстве с имитацией задержки по времени для отладки производительности параллельного алгоритма (**basesim**);
- параллельное выполнение кода задач в общем с кодом акторов адресном пространстве (**omptask**);
- параллельное выполнение кода задач в адресном пространстве другого компьютера, отличного от компьютера, где выполняются акторы (**everest**) – используется ППО платформы Everest, ИППИ РАН.

Стандартные применения Templet SDK - демонстрационное, учебное для практикумов, управление вычислительным экспериментом.

Поэтому основным выбран **метод развертывания на виртуальных машинах в публичном гибридном облаке** на комбинации платформ **MyBinder.org** и **everest.distcomp.org**. В качестве IDE используется **JupyterLab**.

Jupyter-ноутбуки используются для запуска команд: препроцессирования, компиляции, запуска приложений, управления сеансами Everest, получения ID приложений Everest, запуска агентов ресурса на виртуальных машинах и тд.

Язык реализации – C++. В примерах имеются файлы проектов sln/vcxproj для сборки и запуска кода в MS Visual Studio. Операционные системы: Linux, Windows.

Разработка и тестирование выполняется на объединенных посредством платформы Everest виртуальных машинах сервиса MyBinder.

Анализ динамических систем путем вычисления коэффициентов Ляпунова.

Особенности. Вычисления коэффициентов Ляпунова выполнялись в отдельных задачах, реализованных на языке Maple. Имелась группа компьютеров и виртуальных машин установленными лицензированными пакетами Maple. Требовалось интегрировать эти компьютеры для использования в параллельном вычислении коэффициентов Ляпунова и автоматизировать управление порядком выдачи задач.

Источник: Popov S. N. , Vostokin S.V., Doroshin A.V. Dynamical systems analysis using many-task interactive cloud computing // Journal of Physics: Conference Series. 2020. Vol. 1694. Issue 1.

Система частотного анализа текстов.

Особенности. Демонстрация возможности обработки данных в алгоритмах попарного сопоставления данных произвольной структуры. Демонстрация программирования алгоритмов со сложными зависимостями между задачами. Для хранения и обмена данными между виртуальными машинами в публичном облаке Самарского университета использовался доступ к распределенной файловой системе.

Источник: Vostokin S., Bobyleva I. V. Implementation of frequency analysis of Twitter microblogging in a hybrid cloud based on the Binder, Everest platform and the Samara University virtual desktop service // CEUR Workshop Proceedings. 2020. Vol. 2667. P. 162-165.

Направления разработки и исследований проекта Templet SDK:

семантика ; **метаязык** ; **управление акторами** ; **управление задачами** .

В каждом из направлений получены простые, пригодные для широкого круга задач решения, но остаются вопросы для углубленной проработки.

В настоящее время исследуется вопрос о возможности реализации управления задачами на основе концепции блокчейна.

http://templet.ssau.ru - главная страница

http://templet.ssau.ru/wiki - вики проекта Templet и образовательные ресурсы

https://github.com/the-templet-project - Templet SDK x3 - актуальная версия

Автор: Востокин Сергей Владимирович

д.т.н., зав. кафедрой программных систем, Самарский университет

easts@mail.ru

СПАСИБО ЗА ВНИМАНИЕ !