Нижегородский государственный университет им. Н.И Лобачевского
Факультет вычислительной математики кибернетики ННГУ

Библиотека параллельных алгоритмов ParaLib v.1.0
Руководство пользователя

Нижний Новгород
2015

В начале руководства пользователя библиотеки параллельных алгоритмов ParaLib приводятся описания назначения и функциональности библиотеки и визуального приложения, а также краткое руководство пользователя по визуальному приложению библиотеки. Далее рассматриваются архитектура библиотеки и механизм подключения к ней дополнительных вычислительных задач, технологий и языков параллельного программирования и реализаций алгоритмов. Затем описываются требования к программному обеспечению, необходимому для запуска визуального приложения ParaLib. В заключении для всех используемых языков и технологий параллельного программирования описываются требования к установленному программному обеспечению, при необходимости описываются последовательности установки ПО, необходимого для компиляции и запуска использующих соответствующий язык или технологию программ, а также приводятся примеры программ и командных строк их компиляции и запуска.

Оглавление

1. Введение 5
2. Используемые методы и алгоритмы 6
3. Визуальное приложение ParaLib 6
4. Архитектура библиотеки ParaLib 9
4.1. Принцип интеграции различных задач, алгоритмов и технологий 10
4.2. Подключение дополнительных задач 10
4.3. Подключение дополнительных технологий 10
4.4. Подключение дополнительных реализаций алгоритмов 11
5. Запуск визуального приложения ParaLib 12
5.1. Требования к программному обеспечению 12
6. Язык параллельного программирования Chapel 12
6.1. Средства компиляции и сборки 12
6.1.1. Сборка компилятора 12
6.1.2. Установка переменных окружения 12
6.1.3. Сборка приложений 13
6.2. Разработка приложений для среды ParaLib на языке программирования Chaple 13
6.2.1. Передача параметров 13
6.2.2. Пример программы для системы ParaLib 14
6.2.3. Интеграция примера в библиотеку Paralib 14
7. Язык параллельного программирования CoArray Fortran 15
7.1. Средства компиляции и сборки 15
7.1.1. Компиляторы 15
7.1.2. Установка переменных окружений 15
7.1.3. Сборка приложений 15
7.2. Разработка приложений для среды ParaLib с применением технологии CoArray Fortran 16
7.2.1. Пример программы для системы ParaLib 16
7.2.2. Интеграция в систему Paralib 18
8. Технология MPI 19
8.1. Средства компиляции и сборки 19
8.1.1. Библиотеки MPI 19
8.1.2. Сборка приложений 19
8.2. Разработка приложений для среды ParaLib с применением технологии MPI 20
8.2.1. Пример программы для системы ParaLib 20
8.2.2. Интеграция в систему Paralib 20
9. Технология OpenMP 21
9.1. Средства компиляции и сборки 21

9.1.1. Компиляторы 21
9.1.2. Установка переменных окружений 21
9.1.3. Сборка приложений 22
9.2. Разработка приложений для среды ParaLib с применением технологии OpenMP.22
9.2.1. Пример программы для системы ParaLib....22
9.2.2. Интеграция в систему Paralib 23
10. Литература 24

1. Введение

Библиотека параллельных алгоритмов ParaLib содержит краткие описания последовательных и параллельных алгоритмов решения ряда типовых задач вычислительной математики, а также примеры их реализаций с использованием различных технологий параллельного программирования. Для демонстрации работоспособности алгоритмов разработано визуальное приложение, используя которое можно выполнить эксперименты для оценки производительности имеющихся реализаций.
В текущей версии библиотеки рассмотрены следующие задачи:

- умножение матриц;
- умножение матрицы на вектор;
- сортировка массива данных;
- решение системы линейных уравнений;
- поиск всех кратчайших путей в графе;
- решение задачи Дирихле для дифференциального уравнения в частных производных.

Для всех задач представлены реализации последовательных алгоритмов и по крайней мере одна реализация параллельного алгоритма. При подготовке параллельных реализаций использовались различные языки и технологии параллельного программирования:

- технология параллельного программирования для систем с общей памятью OpenMP;
- технология параллельного программирования для систем с распределенной памятью MPI;
- язык параллельного программирования Chapel (Cascade High Productivity Language);
- язык параллельного программирования CoArray Fortran.

Наличие конкретных реализаций в текущей версии ParaLib описывается в Таблице 1, но библиотека позволяет легко добавлять дополнительные задачи и пользовательские реализации.

Таблица 1. Реализации методов и алгоритмов, содержащиеся в ParaLib.

Задача	**Метод**	**Технология**			
		Chapel	OpenMP	CoArray Fortran	MPI
Умножение матриц	последовательный	+	+	+	-
	параллельный	+	+	+	+
Умножение матрицы на вектор	базовый параллельный алгоритм	-	+	-	+
	последовательный	-	+	-	-
Сортировка массива данных	последовательный	-	+	-	-
	базовый параллельный алгоритм	-	+	-	+
Решение системы линейных уравнений	базовый параллельный алгоритм	-	+	-	+
	последовательный	-	+	-	-
Поиск всех кратчайших путей в графе	последовательный алгоритм Флойда	-	+	-	-
	базовый параллельный алгоритм Флойда	-	+	-	+
Решение задачи	базовый параллельный	-	+	-	+

Дирихле для СДУ	алгоритм Гаусса-Зейделя				
	последовательный алгоритм Гаусса-Зейделя	-	+	-	-

Визуальное приложение, входящее в состав библиотеки позволяет выполнять эксперименты для сравнительной оценки производительности имеющихся реализаций. С его помощью можно:

- выбрать вычислительную задачу, для которой имеются реализованные алгоритмы решения,
- выбрать последовательный/параллельный метод для решения выбранной задачи;
- выбрать язык/технологию параллельного программирования для решения выбранной задачи;
- задать параметры задачи;
- определить число используемых процессоров;
- выполнить эксперимент, в ходе которого будет выполнен запуск решения выбранной задачи;
- накапливать и анализировать результаты выполненных экспериментов.

Выполнение экспериментов может осуществляться на локальном компьютере или на вычислительной системе с распределенной памятью.
Все реализации выполнены для операционной системы Windows 7/Windows 8.
Библиотека ParaLib носит учебно-исследовательский характер и может быть использована в ходе обучения специалистов различным аспектам параллельного программирования. Имеющиеся реализации могут использоваться в качестве образцов при разработке параллельных программ с использованием различных языков и технологий переллельного программирования.

2. Используемые методы и алгоритмы

Библиотека ParaLib включает краткие описания и реализации алгоритмов решения нескольких вычислительных задач. Подробное описание используемых алгоритмов можно прочитать в [1-2].

3. Визуальное приложение ParaLib

Визуальное приложение ParaLib предназначено для демонстрации работоспособности алгоритмов. Также с его помощью можно выполнить эксперименты для оценки производительности имеющихся реализаций.
Запуск визуального приложения осуществляется вызовом файла ParaLib.jar (для запуска приложения необходимо, что бы были установлены все runtime библиотеки указанные в разделе 4). Для корректной работы приложения в каталоге с исполняемым файлом должен находиться конфигурационный файл tasks.xml и необходимый набор исполняемых и текстовых файлов.
В демонстрационной системе представлен пример организации набора исполняемых файлов и текстовых документов, необходимых для работы системы. Также показан способ написания конфигурационного файла.
Рассмотрим визуальную составляющую системы. Ниже представлен общий вид визуального приложения:

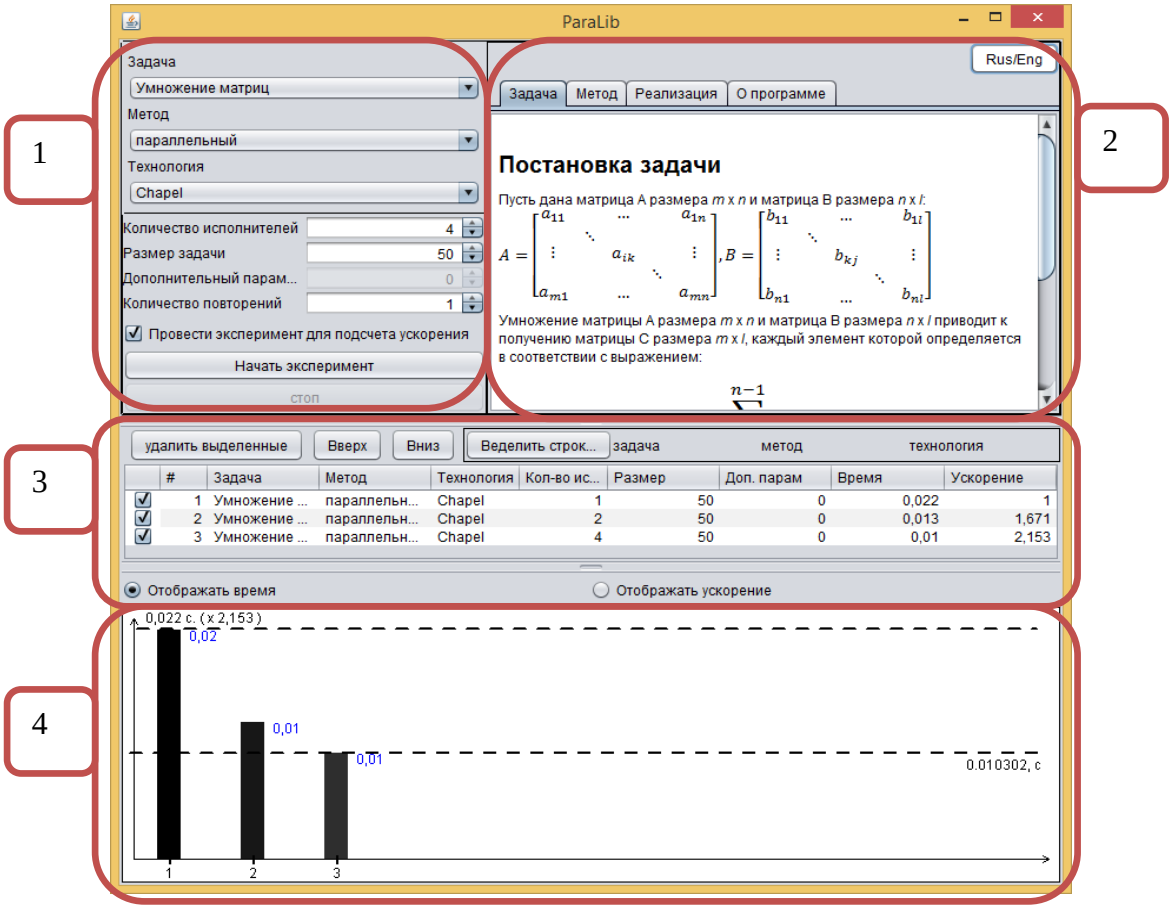

Рис. 1. Внешний вид приложения.

В приложении можно выделить ряд областей. Каждая часть системы используется для задания условий экспериментов, получения сведений о проводимом эксперименте и сравнении численных результатов вычислений. Рассмотрим каждую из областей подробнее.

1. Выбор задачи и метода решения

На рисунке в левом верхнем углу приложения (см. Рис. 1 область 1) находится меню задания условий численного эксперимента.

Для проведения эксперимента необходимо вначале выбрать решаемую задачу. Выбрав задачу можно указать метод ее решения. Определившись с алгоритмом необходимо указать технологию, используемую для реализации.

В демонстрационной системе рассматривается ряд классических задач, используемых для обучения основам параллельного программирования. Для задач представлены два метода реализации – последовательный алгоритм и базовый параллельный. Все задачи содержат реализации с использованием технологий OpenMP и MPI. На примере задачи умножения матриц демонстрируется способ реализации и интеграции в библиотеку решений задач с применением языков параллельного программирования CoArray Fortran и Chapel.

2. Информационная панель

Получить необходимую информацию о выбранном алгоритме и его реализации можно в правом верхнем углу приложения (см. Рис. 1 область 2). Выбор отображаемой информации осуществляется переключением вкладок.

Вкладка «Задача» используется для отображения постановки решаемой задачи. У каждой задачи может быть несколько методов реализации. Описание выбранного метода находиться в соответствующей вкладке. Выбрав задачу, метод реализации и технологию можно изучить пример реализации алгоритма во вкладке «Реализация».

Последняя вкладка используется для отображения информации о библиотеке и реализованных в ней на текущий момент задачах и методах.

3. Запуск вычислительного эксперимента

Выбрав алгоритм и интересующую реализацию, необходимо задать условия вычислительного эксперимента. В системе ParaLib принято ограничение на использование следующих параметров:

1. Для эксперимента нужно указать количество исполнителей. В качестве исполнителей в зависимости от технологии могут быть вычислительные потоки или процессы.
2. Нужно также указать размер решаемой задачи. Размер интерпретируется в зависимости от задачи. Например, в задаче умножения матриц в качестве размера указывается размер матрицы, а задачах оптимизации на графах – количество вершин графа. Предполагается, что описание значения параметра указано в постановке задачи (см. раздел 2 "Информационная панель").
3. В ряде задач используется дополнительный параметр. Интерпретация значения зависит от постановки задачи или метода реализации. Подробную информацию о параметре так же можно узнать в постановке задачи или описании метода реализации. Параметр блокируется, если не используется.
4. Количество раз, которое эксперимент будет выполнен. Так как от запуска к запуску время выполнения может меняться визуальная система предлагает возможность уточнения времени исполнения путем повторного запуска эксперимента несколько раз.

Программный комплекс позволяет автоматически считать ускорение для проведенных экспериментов. Для подсчета ускорения необходим запуск в один поток или процесс в зависимости от технологии. Подобный эксперимент проводится автоматически, если установлена галочка в поле «Провести эксперимент для подсчета ускорения».

Для запуска вычислительного эксперимента необходимо нажать на кнопку «Начать эксперимент». В любой момент времени эксперимент можно остановить, нажав кнопку остановки. На кнопке отображается таймер, указывающий на текущее время работы заданного эксперимента.

4. Коллекция результатов вычислительных экспериментов

В середине окна визуального приложения (см. Рис. 1 область 3) отображаются результаты всех выполненных экспериментов. Для каждого проведенного эксперимента в таблице указываются параметры эксперимента и время работы. В таблице слева можно поставить галочку напротив тех запусков, которые хотелось бы отобразить на графике. После выполнения эксперимента, по умолчанию время работы или ускорение всегда отображается на графике.

Для удобства пользователя реализованы ряд дополнительных инструментов работы с результатами экспериментов. Для упорядочивания результатов, выбрав строку, есть возможность ее можно переместить вверх или вниз, нажав соответствующую кнопку. Строки с результатами выделенные галочками можно удалить. Если кликнуть на строчку в таблице, а затем на кнопку «Выделить строки с:», все результаты у которых совпадает решаемая задача, метод и технология выделятся галочками.

5. Визуальное сравнение времени выполнения экспериментов

Визуальное сравнение численных данных, как правило, позволяет быстрее сделать выводы относительно проведенных экспериментов. Для удобства пользователя внизу визуального приложения (см. Рис. 1 область 4) строится гистограмма сравнения результатов вычислительных экспериментов. На гистограмме по горизонтальной оси отложен номер выполненного эксперимента, а по вертикальной время работы или ускорение. На графике указывается максимальное и минимальное время (ускорение), а также отношение максимального времени к минимальному. Рядом с каждым столбиком в гистограмме указывается время или ускорении.

4. Архитектура библиотеки ParaLib

Программный комплекс ParaLib состоит из нескольких модулей. Общая архитектура приложения представлена на Рис. 2.

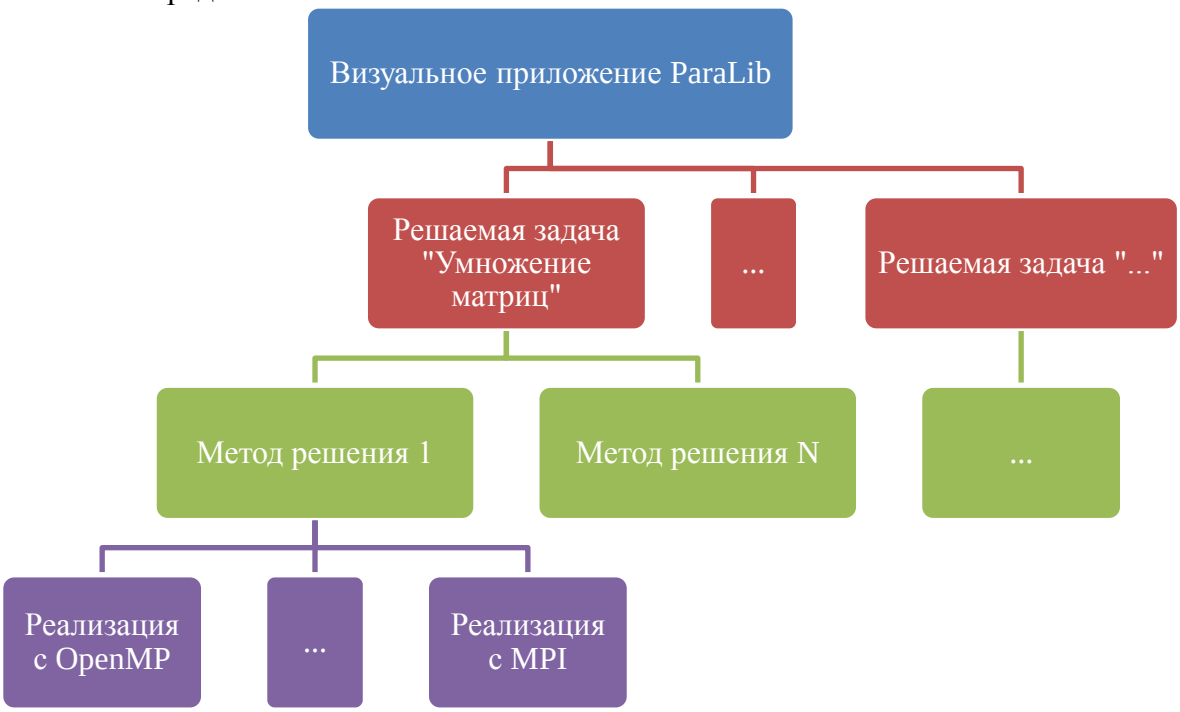

Рис. 2. Архитектура библиотеки ParaLib

Рассмотрим назначение каждого модуля.

В реализованном комплексе предполагается, что может существовать несколько задач. Для введения дополнительной задачи в программный комплекс необходимо разработать HTML документ с описание постановки задачи. Документ должен быть сохранен в кодировке UTF-8.

В рамках каждой задачи может существовать несколько методов решения. Текущей версии системы представлены два метода решения – последовательный и параллельный. Для добавления дополнительного метода также как и в случае с задачей необходимо разработать описание метода. Документ должен быть сохранен в кодировке UTF-8.

Для разработанного метода решения задачи в комплексе может существовать несколько реализаций. Каждая реализация характеризуется текстовых файлом, содержащим код, и исполнительным модулем. Модуль на вход должен принимать четыре параметра – размер задачи, количество повторных запусков алгоритма, количество исполнителей (количество процессов или потоков) и возможно дополнительный параметр. Результат работы модуля – время работы каждого запуска алгоритма записанное в отдельной строке.

Модуль «Визуальное приложение ParaLib» связывает все текстовые файлы и вычислительные исполняемы модули для выбора задач и их исполнения. Связь между модулями описывается по средствам XML файла tasks.xml. Данный файл должен находиться в одной директории с исполняемым модулем визуального приложения. Для запуска визуальной оболочки должны быть установлены все runtime библиотеки из пункта 5.

4.1. Принцип интеграции различных задач, алгоритмов и технологий

Для интеграции задач и реализаций алгоритмов используется XML файл tasks.xml. Общая структура документа представлена ниже:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <technology>
//описание технологий
    </technology>
//описание задач
    <task nameru="Умножение матриц" nameeng="Matrix multiplication"
          disc_text="tests\MM_task.html">
//описание используемых методов
        <method nameru="последовательный" nameeng="serial"
                text_src="tests\single.html">
//описание используемых технологий
            <use_tech id="1" >
                <run param2="false"
                     exe="tests\MM\OMP\OMP_Matrix_sing.exe"
                     src_code="tests\MM\OMP\OMP_Matrix_sing.cpp"
                />
            </use_tech>
        </method>
    </task>
</root>
```

В начале документа описываются используемые технологии в рамках программного комплекса (тег «technology»). Далее документ содержит описание задач. Описание каждой задачи содержится в теге «task». Каждая задача может содержать несколько методов решения (тег «method»). В методе может быть представлено несколько реализаций каждая заключена в тег «use_tech».

4.2. Подключение дополнительных технологий

Для подключения новой технологии в теге «technology» необходимо добавить запись с тегом «tech» следующего вида:

```
    <technology>
…
        <tech id="0">
            <name value="Chapel" />
            <description src="tech/chaple.html"/>
            <param_pref
            n_repeat="--n_repeat="
            n = "--n="
            n_th="--dataParTasksPerLocale="
            param2="--param2="
            />
        </tech>
…
```

Каждая технология должна содержать уникальный идентификатор («id»). По средствам идентификатора осуществляется связь между описанием технологий и реализацией. Технология содержит тег с именем и указанием пути к файлу с описанием технологии.

Кроме этого в теге «tech» содержится тег «param_pref». Тег описывает особенности запуска программы. Возможные атрибуты тега:

```
//возможные префиксы при передачи параметров исполняемым модулям
n_repeat="--n_repeat="
n = "--n="
n_th="--dataParTasksPerLocale="
param2="--param2="
//возможность задать количество исполнителей
//через переменные окружения
env_n_th="FOR_COARRAY_NUM_IMAGES"
//возможность запуска программы через внешний модуль
exec="c:\Program Files (x86)\MPICH\mpd\bin\MPIRun.exe"
//количество исполнителей - параметр внешнего модуля
n_th_exec="1"
```

4.3. Подключение дополнительных задач

Для создания задач в XML документе создается тег «task» см. пункт 4.1. Пример задачи ниже

```
<task nameru="Умножение матриц" nameeng="Matrix multiplication"
          disc_text="tests\MM_task.html">
…
</task>
```

Необходимые атрибуты, указываемые в задачах – название на русском и английском языке, а также путь к документу с текстовым описанием постановки решаемой задачи.

4.4. Подключение дополнительных реализаций алгоритмов

Все реализации алгоритмов заключены в одном из методов и реализованы с использованием одной из технологий. Для того что бы включить реализацию надо добавить следующие теги в XML документ:

```
...
<task nameru="Умножение матриц" nameeng="Matrix multiplication"
    disc_text="tests\MM_task.html">
  ...
  <method nameru="последовательный" nameeng="serial"
          text_src="tests\single.html">
  ...
      <use_tech id="1" >
          <run param2="false"
               exe="tests\MM\OMP\OMP_Matrix_sing.exe"
               src_code="tests\MM\OMP\OMP_Matrix_sing.cpp"
          />
      </use tech>
  ...
  </method>
  ...
</task>
...
```

Каждый метод (тег «method»), как и задача, содержит атрибуты название на русском и английском языке, а также путь к файлу с описанием метода.

Для добавления реализации используется тег «use_tech». В теге указывается один из существующих идентификаторов технологии (см. раздел 4.2). В рамках тега «use_tech» используется тег «run» для указания пути к исполняемому модулю и текстовому файлу с кодом.

5. Запуск визуального приложения ParaLib

Запуск визуального приложения ParaLib выполняется посредством выполнения команды ParaLib.jar или любого аналогичного действия (например, двойного щелчка по файлу программы).

5.1. Требования к программному обеспечению

Для работы ParaLib необходимо установить несколько пакетов, содержащих динамические библиотеки. Java Runtime необходим для запуска визуальной оболочки программного комплекса. Остальные пакеты необходимы для запуска тестовых приложений (ссылки и версии ПО приведены на момент написания руководства).

1. Java Runtime – используется в визуальном приложении.

https://java.com

2. Microsoft Visual C++ 2010 – пакет необходим для запуска большинства исполняемых приложений.

http://www.microsoft.com/en-us/download/details.aspx?id=5555

3. Redistributable Libraries for Intel(R) Parallel Studio XE 2015 for C++ and Fortran Windows* используется для запуска приложений разработанных на Fortran

https://software.intel.com/en-us/articles/redistributables-for-intel-parallel-studio-xe-2015-composer-edition-for-windows

4. MPICH 1 используется для запуска MPI приложений

http://www.mcs.anl.gov/research/projects/mpi/mpich1-old/mpich-nt/

6. Язык параллельного программирования Chapel

Данный раздел содержит информацию, необходимую для разработки и запуска программ с использованием языка параллельного программирования Chapel.

6.1. Средства компиляции и сборки

6.1.1. Сборка компилятора

Для сборки программы необходимо скомпилировать компилятор языка программирования Chapel. Компилятор можно найти по ссылке http://chapel.cray.com/download.html. На момент написание документа актуальная версия компилятора 1.11.0. По указанной ссылке необходимо загрузить файл **chapel-1.11.0.tar.gz**

Для сборки chapel под windows необходимо установить cygwin (см. https://www.cygwin.com/).

При установке необходимо выбрать установку пакета разработки (devel) (компилятор разработан на языке программирования C/C++)

Процесс установки.

1. Переходим в папку со скаченным пакетом chapel
2. Разархивируем пакет:

```
tar xzf ./chapel-1.9.0.tar.gz
```

3. Переходим в папку пакета

```
cd chapel-1.9.0/
```

4. Собираем пакет командой make

6.1.2. Установка переменных окружения

После сборки Chapel можно приступать к разработке. Для удобства задания переменных окружения можно воспользоваться сгенерированным скриптом внутри пакета chapel:

```
../util/setchplenv.sh
Setting CHPL_HOME...
                    ...to /cygdrive/d/temp/chaple/chapel-1.9.0
Setting CHPL_HOST_PLATFORM...
                         ...to cygwin
Updating PATH to include...
                            .../cygdrive/d/temp/chaple/chapel-1.9.0/bin/cygwin
                       and .../cygdrive/d/temp/chaple/chapel-1.9.0/util
Updating MANPATH to include...
                            .../cygdrive/d/temp/chaple/chapel-1.9.0/man
```

6.1.3. Сборка приложений

Для сборки приложений используется собранный компилятор. Ниже представлен формат строки компилятора:

```
chpl [flags] [source files]
```

Пример командной строки сборки:

```
chpl -o hello examples/hello.chpl
```

Запуск:

```
./hello
```

Запуск без cygwin:
Для работы приложений без cygwin достаточно в папку с приложением скопировать cygwin1.dll.

6.2. Разработка приложений для среды ParaLib на языке программирования Chaple

Все приложения в системе ParaLib с применением Chapel должны принимать четыре параметра – количество потоков, размер задачи, количество запусков приложения и дополнительный параметр. Результатом работы программы должен быть вывод, содержащий только времена исполнения каждого из запусков. Каждое время должно выводиться на новой строке.

6.2.1. Передача параметров

В программах на chaple нельзя явным образом передать аргументы командной строки в программу. Для передачи параметров используются флаги. Флаги есть двух типов – пользовательские и встроенные влияющие на работу приложения.
Пример встроенного флага:
Если необходимо установить количество потоков необходимо использовать флаг «dataParTasksPerLocale». Пример запуска:

```
run.exe --dataParTasksPerLocale=2
0.134209

run.exe --dataParTasksPerLocale=1
0.187388
```

Если необходимо создать пользовательский флаг используется строка вида:

```
config const n = 100;
```

Если есть переменная со ключевым словом config ее можно задать через флаг:

```
run.exe --dataParTasksPerLocale=2 --n=200
```

6.2.2. Пример программы для системы ParaLib

Пример программы (жирным синим цветом обозначены обязательные флаги для системы ParaLib):

```
use Random, Time;

config const n_repeat = 1;
config const param2   = 0;
config const n = 100;

proc main
{
  var t = new Timer();
  var A, B, C1, C2, C3: [1..n, 1..n] real;

  fillRandom(A);
  fillRandom(B);

  for i in 1..n_repeat do
  {
        t.clear();
        t.start();
        forall j in 1..n do
            for i in 1..n do
              for k in 1..n do
                  C3(i,j) += A(i,k) * B(k, j);

        t.stop();
        writeln(t.elapsed());
  }
}
```

В коде есть два типа циклов for и forall. Последний задает параллельный цикл.

6.2.3. Интеграция примера в библиотеку Paralib

Для интеграции примера необходимо сформировать следующий конфигурационный файл:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <technology>
        <tech id="0">
            <name value="Chapel" />
            <description src="tech/chaple.html"/>
            <param_pref
            n_repeat="--n_repeat="
            n = "--n="
            n_th="--dataParTasksPerLocale="
            param2="--param2="
            />
        </tech>
    </technology>
    <task nameru="Умножение матриц"
          nameeng="Matrix multiplication"
          disc_text="tests\MM_task.html">
```

```
      <method nameru="параллельный"
              nameeng="parallel"
              text_src="tests\MM_method.html">

          <use_tech id="0" >
                <run param2="false"
                     exe="tests\MM\Chapel\CHPL_Matrix_bpar.exe"
                 src_code="tests\MM\Chapel\CHPL_Matrix_bpar.chpl"
                                />
            </use_tech>
      </method>
    </task>
  </root>
```

Если производится интеграция с другими задачами/методами/алгоритмами, разделы `<technology>`, `</task>`, `</method>`, `</use_tech>` добавляются в основной конфигурационный файл `tasks.xml`.

7. Язык параллельного программирования CoArray Fortran

Данный раздел содержит информацию, необходимую для разработки и запуска программ с использованием языка параллельного программирования CoArray Fortran.

7.1. Средства компиляции и сборки

7.1.1. Компиляторы

Для разработки приложений с применением технологией CoArray Fortran, технологию должен поддерживать компилятор. Компилятор Intel поддерживает технологию.
Пример использования технологии в системе ParaLib будет продемонстрировано на основе компилятора Intel(R) Composer XE 2013 SP1 Update 2 (package 176). Операционная система Windows.

7.1.2. Установка переменных окружений

Для компиляции приложений необходимо установить переменные окружения – путь к компилятору, папке с заголовочными файлами и библиотекам.
Установить переменные можно вручную или воспользоваться стандартным скриптом ifortvars.bat. Ниже представлен пример применения скрипта.

```
C:\Program Files (x86)\Intel\Composer XE 2013 SP1\bin>ifortvars.bat ia32
Intel(R) Parallel Studio XE 2013 SP1
Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
Intel(R) Composer XE 2013 SP1 Update 2 (package 176)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
```

7.1.3. Сборка приложений

Для сборки приложений компилятором Intel применяется следующая команда.

```
ifort [flags] [source files]
```

Для сборки приложений компилятором Intel с применением CoArray Fortran для систем с общей памятью дополнительные ключи не нужны.

Пример строки сборки:

```
ifort main.F95
```

Запуск:

```
./main.exe
```

7.2. Разработка приложений для среды ParaLib с применением технологии CoArray Fortran

Все приложения в системе ParaLib с применением CoArray Fortran должны принимать три параметра – размер задачи, количество запусков приложения и дополнительный параметр. Количество потоков задается через переменные окружения. Результатом работы программы должен быть вывод, содержащий только времена исполнения каждого из запусков. Каждое время должно выводиться на новой строке.

7.2.1. Пример программы для системы ParaLib

Пример программы (жирным синим цветом обозначены обязательные строки кода для системы ParaLib):

```
PROGRAM MatrixMultiplication

!!!!!!!!!!!!!!!!!!!!!!
!Объявление параметров!
!!!!!!!!!!!!!!!!!!!!!!

!перемножаемые матрицы
real, allocatable :: LmatrixL(:,:)[:], LmatrixR(:,:)[:],LmatrixRes(:,:)[:]
real, allocatable :: matrixL(:,:), matrixR(:,:),matrixRes(:,:)
!индексы
integer :: matrix_size,i,j,blok_size,images_num,blok_dimention,t
integer :: stolb,temp2,temp3,strok, err

!timer variable
integer, parameter :: DP = selected_real_kind(15)
real :: T1,T2,Seconds2, Seconds
integer :: pulses, PPS

!!!!!!!!!!!!!!!!!!!!!!
!Объявление параметров!
!!!!!!!!!!!!!!!!!!!!!!

!Чтение параметров командной строки
character(LEN=10) :: r
integer :: count

count = command_argument_count()
IF (count < 4) STOP

CALL GET_COMMAND_ARGUMENT(1,r)
read(r,*) nrepeat

CALL GET_COMMAND_ARGUMENT(2,r)
read(r,*) matrix_size

CALL GET_COMMAND_ARGUMENT(4,r)
read(r,*) param2
```

```
!Параметры запуска
me = this_image()
images_num = num_images()!количество  images
blok_size = int(matrix_size/images_num)!размер блока
n = matrix_size

allocate (LmatrixL(matrix_size,matrix_size)[*])
allocate (LmatrixR(matrix_size,matrix_size)[*])
allocate (LmatrixRes(matrix_size, matrix_size)[*])

!print *, n
!print *, blok_size

do irep=1, nrepeat

    !!!!!!!!!!!!!!!!!!!!!
    !Инициализаия матрицы!
    !!!!!!!!!!!!!!!!!!!!!
    if (me==1) then
        allocate (matrixL(matrix_size,matrix_size))
        allocate (matrixR(matrix_size,matrix_size))
        allocate (matrixRes(matrix_size,matrix_size))



        do i=1, matrix_size
            do j=1, matrix_size
                matrixL(i,j)=cos(real(i))*cos(real(j))
                matrixR(i,j)=sin(real(i))*sin(real(j))
                matrixRes(i,j) = 0.0

            end do
        end do
    end if

    sync all

    if (me==1) then
        CALL SYSTEM_CLOCK(COUNT=Pulses,COUNT_RATE=PPS)
        T1 = REAL(Pulses,DP)/PPS
    end if

    sync all

    if (me==1) then

        do k = 1, images_num
            LmatrixL(:,:)[k]=matrixL(matrix_size,matrix_size)
            LmatrixR(:,:)[k]=matrixR(matrix_size,matrix_size)
        end do
    endif

      do i=1, matrix_size
            do j=1, matrix_size
                  LmatrixRes(i,j)[me] = 0.0
            end do
      end do

    sync all
        do j=1, n
            do i=1, blok_size
                do k=1, n
```

```
                    LmatrixRes(i,j)[me]=    &
                      LmatrixRes(i,j)[me] + &
                      LmatrixL(i     +   (me   -   1)   *   blok_size,k)[me]   *
LmatrixR(k,j)[me]
                end do
            end do
        end do

    if (me==1) then
        do j=1, n
            do i=blok_size * images_num, matrix_size
                do k=1, n
                    matrixRes(i,j)=    matrixRes(i,j)    +    matrixL(i,k)    *
matrixR(k,j)
                end do
            end do
        end do
    endif

    sync all

    if (me==1) then
        do k = 1, images_num
            do j=1, n
                do i=1, blok_size
                    matrixRes(i + (k-1) * blok_size,j)=LmatrixRes(i,j)[k]
                end do
            end do
        end do
    endif

    sync all

    if (me==1) then
        !конец таймера 1
        CALL SYSTEM_CLOCK(COUNT=Pulses,COUNT_RATE=PPS)
        T2 = REAL(Pulses,DP)/PPS
        Seconds=T2-T1

        print *, Seconds
        deallocate (matrixL)
        deallocate (matrixR)
        deallocate (matrixRes)

    end if

end do

deallocate (LmatrixL)
deallocate (LmatrixR)
deallocate (LmatrixRes)

END PROGRAM MatrixMultiplication
```

7.2.2. Интеграция в систему Paralib

Для интеграции примера необходимо сформировать следующий конфигурационный файл:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <technology>
        <tech  id="2">
            <name value="CoArray Fortran" />
```

```
            <description src="tech/coarray.html"/>
            <param_pref
            env_n_th="FOR_COARRAY_NUM_IMAGES"

            />
        </tech>
    </technology>
    <task nameru="Умножение матриц" nameeng="Matrix multiplication"
          disc_text="tests\MM_task.html">
        <method nameru="параллельный" nameeng="parallel"
                text_src="tests\MM_method.html">
            <use_tech id="2" >
                <run param2="false"
                     exe="tests\MM\CAF\CAF_Matrix_bpar.exe"
                 src_code="tests\MM\CAF\CAF_Matrix_bpar.f90"
                             />
            </use_tech>
        </method>
    </task>
</root>
```

Если производится интеграция с другими задачами/методами/алгоритмами, разделы `<technology>`, `</task>`, `</method>`, `</use_tech>` добавляются в основной конфигурационный файл `tasks.xml`.

8. Технология MPI

8.1. Средства компиляции и сборки

8.1.1. Библиотеки MPI

Для разработки приложений с применением технологии MPI необходимо использовать одну из библиотек реализующих стандарт.
В примере в качестве реализации библиотеки будет использоваться MPICH (http://www.mpich.org/).

8.1.2. Сборка приложений

Для сборки приложений необходимо при компиляции указать путь к библиотеке и заголовочным файлам.

```
cl -I"C:\Program Files (x86)\MPICH\SDK\include\" [source files] /link
/LIBPATH:"C:\Program Files (x86)\MPICH\SDK\lib\" mpich.lib
```

Пример строки сборки:

```
cl -I"C:\Program Files (x86)\MPICH\SDK\include\" main.cpp /link
/LIBPATH:"C:\Program Files (x86)\MPICH\SDK\lib\" mpich.lib
```

Для запуска приложений используется специальная программа mpiexec.exe. Ниже приведен пример запуска приложения в четыре процесса

```
mpiexec –n 4 ./main.exe
```

8.2. Разработка приложений для среды ParaLib с применением технологии MPI

Все приложения в системе ParaLib с применением технологии MPI должны принимать три параметра – размер задачи, количество запусков приложения и дополнительный параметр. Результатом работы программы должен быть вывод, содержащий только времена исполнения каждого из запусков. Каждое время должно выводиться на новой строке.

8.2.1. Пример программы для системы ParaLib

Пример программы (жирным синим цветом обозначены обязательные строки для системы ParaLib):

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char** argv)
{
  …
  int RowNum, ColNum, BlockSize;
  int ProcNum, ProcRank;
  int i;
  int Size = 10;
  int Thread = 1;
  int repit = 1;
  double time;

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

  if(argc < 3)
  {
    printf("\nERROR! Not enough parameters to start.\n");
    exit(-1);
  }

  repit = atoi(argv[1]);
  Size = atoi(argv[2]);
  for(i = 0; i < repit; i++)
  {
      time = MPI_Wtime();

      //реализация алгоритма
      //…

      time = MPI_Wtime() - time;
      if(ProcRank == 0)
      {
        printf("%lf\n", time);
      }
  }
  MPI_Finalize();
  return 0;
}
```

8.2.2. Интеграция в систему Paralib

Для интеграции примера необходимо сформировать следующий конфигурационный файл:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<root>
    <technology>
        <tech  id="3">
            <name value="MPI" />
            <description src="tech/mpi.html"/>
            <param_pref
            n_th = "-localonly "
            exec="c:\Program Files (x86)\MPICH\mpd\bin\MPIRun.exe"
            n_th_exec="1"
            />
        </tech>
    </technology>

    <task nameru="Умножение матрицы на векторов"
          nameeng="Matrix-Vector multiplication"
          disc_text="tests\MV_task.html">

        <method nameru="базовый параллельный алг." nameeng="parallel"
                text_src="tests\MV_method.html">
            <use_tech id="3" >
                <run param2="false"
                     exe     ="tests\MV\MPI\MPI_Matrix_vector_bpar.exe"
                   src_code="tests\MV\MPI\MPI_Matrix_vector_bpar.c"
                            />
            </use_tech>
        </method>
    </task>
</root>
```

Если производится интеграция с другими задачами/методами/алгоритмами, разделы `<technology>`, `</task>`, `</method>`, `</use_tech>` добавляются в основной конфигурационный файл `tasks.xml`.

9. Технология OpenMP

9.1. Средства компиляции и сборки

9.1.1. Компиляторы

Для разработки приложений с применением OpenMP технологию должен поддерживать компилятор. Полный перечень компиляторов поддерживающих технологию представлен на официальном сайте OpenMP (http://openmp.org/wp/openmp-compilers/).
Пример использования технологии в системе ParaLib будет продемонстрировано на основе компилятора Intel(R) Composer XE 2013 SP1 Update 2 (package 176). Операционная система Windows.

9.1.2. Установка переменных окружений

Для компиляции приложений необходимо установить переменные окружения – путь к компилятору, папке с заголовочными файлами и библиотекам.
Установить переменные можно вручную или воспользоваться стандартным скриптом iclvars.bat. Ниже представлен пример применения скрипта.

```
c:\>"c:\Program Files (x86)\Intel\Composer XE 2013 SP1\bin\iclvars.bat" ia32
Intel(R) Parallel Studio XE 2013 SP1
Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
Intel(R) Composer XE 2013 SP1 Update 2 (package 176)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
```

9.1.3. Сборка приложений

Для сборки приложений компилятором Intel применяется следующая команда.

```
icl [flags] [source files]
```

Для сборки приложений компилятором Intel с применением OpenMP применяется дополнительный ключ /Qopenmp.

```
icl /Qopenmp [flags] [source files]
```

Пример строки сборки:

```
icl /Qopenmp main.cpp
```

Запуск:
```
./main.exe
```

9.2. Разработка приложений для среды ParaLib с применением технологии OpenMP

Все приложения в системе ParaLib с использованием технологии OpenMP должны принимать четыре параметра – количество потоков, размер задачи, количество запусков приложения и дополнительный параметр. Результатом работы программы должен быть вывод, содержащий только времена исполнения каждого из запусков. Каждое время должно выводиться на новой строке.

9.2.1. Пример программы для системы ParaLib

Пример программы (жирным синим цветом обозначены обязательные флаги для системы ParaLib):

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int OMP_baseMatrixVectorMultiplication(
    const double* pMatrix,
    const double* pVector,
    double* pResult,
    const int Size)
{
  int i, j;
  #pragma omp parallel for private (j)
  for (i=0; i<Size; i++)
  {
    for (j=0; j<Size; j++)
    {
      pResult[i] += pMatrix[i*Size+j]*pVector[j];
    }
  } //for (i=0; i<Size; i++)
  return 0;
}

int main(int argc, char** argv)
{
    int i;
    double *pMatrix, *pVector, *pResult;
```

```
    int Size = 10;
    int Thread = 1;
    int repeat = 1;
    double time;

    if(argc < 4)
    {
      printf("\nERROR! Not enough parameters to start.\n");
      exit(-1);
    }

    repeat = atoi(argv[1]);
    Size = atoi(argv[2]);
    Thread = atoi(argv[3]);

    omp_set_num_threads(Thread);

    for(i = 0; i < repeat; i++)
    {
      ProcessMatrVectInit(&pMatrix, &pVector, &pResult, Size);

      time = omp_get_wtime();
      OMP_baseMatrixVectorMultiplication(pMatrix, pVector,
                                         pResult, Size);
      time = omp_get_wtime() - time;
      printf("%lf\n", time);

      ProcesMatrVectTermination(pMatrix, pVector, pResult);
    }

    return 0;
}
```

9.2.2. Интеграция в систему Paralib

Для интеграции примера необходимо сформировать следующий конфигурационный файл:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <technology>
        <tech  id="1">
            <name value="OpenMP" />
            <description src="tech/openmp.html"/>
            <param_pref

            />
        </tech>
    </technology>
    <task nameru="Умножение матрицы на векторов"
          nameeng="Matrix-Vector multiplication"
          disc_text="tests\MV_task.html">
        <method nameru="базовый параллельный алг." nameeng="parallel"
                text_src="tests\MV_method.html">
            <use_tech id="1" >
                <run param2="false"
                     exe     ="tests\MV\OMP\OMP_Matrix_vector_bpar.exe"
                         src_code="tests\MV\OMP\OMP_Matrix_vector_bpar.c"
                              />
            </use_tech>
        </method>
    </task>
</root>
```

Если производится интеграция с другими задачами/методами/алгоритмами, разделы `<technology>`, `</task>`, `</method>`, `</use_tech>` добавляются в основной конфигурационный файл `tasks.xml`.

10. Литература

1. Гергель В.П. Теория и практика параллельных вычислений. - М.: Интернет-университет информационных технологий – ИНТУИТ.РУ, 2007. 424 с.
2. Учебный курс CS338. Многопроцессорные вычислительные системы и параллельное программирование. [http://www.hpcc.unn.ru/mskurs/cs338_ppr_index.htm]