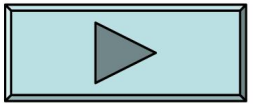
Разработка параллельных программ для основе модели PGAS…

Программа HPCS: High Productivity Computing Systems (DARPA)

Цель: Повышение продуктивности в 10 раз к 2010

(Продуктивность=Производительность+Программируемость+Переносимость+Надежность)

- **Phase II**: Cray, IBM, Sun (July 2003 – June 2006)
  - Анализ существующих архитектур
  - Три новых языка программирования (Chapel, X10, Fortress)
- **Phase III**: Cray, IBM (July 2006 – 2010)
  - Реализация
  - Работа продолжена для всех предложенных языков



Разработка параллельных программ для основе модели PGAS…

Общая характеристика языков программы HPCS:…

- Новые объектно-ориентированные языки, поддерживающие широкий набор средств программирования, параллелизм и безопасность.
- Поддерживаются глобальное пространство имен, многопотоковость и явные механизмы для работы с локальностью.
- Имеются средства для распределения массивов по вычислительным узлам системы, а также для установления связи между потоками управления и данными, которые ими обрабатываются.
- Поддерживаются как параллелизм по данным, так параллелизм задач.

Разработка параллельных программ для основе модели PGAS…

Общая характеристика языков программы HPCS:…

Управление локальностью

- Во всех трех языках обеспечивается доступ пользователей к виртуальным единицам локальности, называемых *локалями* (locale) в Chapel, *регионами* (region) в Fortress и *участками* (place) в X10.
- Каждое выполнение программы привязывается к некоторому набору таких единиц локальности, которые отображаются операционной системой на физические сущности, такие как вычислительные узлы.
- Это обеспечивает пользователей механизмом для (1) распределения коллекций данных по единицам локальности, (2) выравнивания различных коллекций данных и (3) установления связи между вычислительными потоками управления и данными, над которыми они работают.

Разработка параллельных программ для основе модели PGAS…

Общая характеристика языков программы HPCS:…

- В **Chapel** отсутствуют встроенные распределения, но обеспечивается обширная инфраструктура, которая поддерживает произвольные распределения данных, определяемые пользователями и являющиеся достаточно мощными для того, чтобы позволить работу с разреженными данными.

Многопотоковость (распараллеливание циклов).

- В **Chapel** различаются последовательные циклы «for» и параллельные циклы «forall», в которых итерации над элементами индексной области производятся без ограничений. За избежание зависимостей, приводящих к гонкам данных, отвечают пользователи.

Разработка параллельных программ для основе модели PGAS: Chapel…

Анонс – программа «Hello, world»

- Быстрое прототипирование

```
writeln(“Hello, world”);
```

- Структурное программирование

```
def main() {
  writeln(“Hello, world”);
}
```

- Приложение

```
module HelloWorld {
  def main() {
    writeln(“Hello, world”);
  }
}
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Общая характеристика:

- **Синтаксис**
  - Использование отдельных элементов C, C#, C++, Java, Ada, Perl, ...
- **Семантика**
  - Императивный, блочно-структурированный, массивы
  - Объектно-ориентированное программирование (опционно)
  - Статический контроль типов

Разработка параллельных программ для основе модели PGAS: Chapel…

Элементы языка - Диапазоны:

- **Синтаксис**

  range-expr:

  [*low*] .. [*high*] [by *stride*]

- **Семантика**

  Регулярная последовательность целых

  – *stride > 0: low, low+stride, low+2*stride, ... ≤ high*

  – *stride < 0: high, high+stride, high+2*stride, ... ≥ low*

- **Примеры**

  – 1..6 by 2 // 1, 3, 5

  – 1..6 by -1 // 6, 5, 4, 3, 2, 1

  – 3.. by 3 // 3, 6, 9, 12, ...

Разработка параллельных программ для основе модели PGAS: Chapel…

Элементы языка - Массивы:

- **Синтаксис**

  array-type:

  [*index-set-expr*] *type*

- **Семантика**

  Массив элементов, размер определяется множеством индексов

- **Примеры**

```
– var A: [1..3] int,          // Массив из 3 элементов
      B: [1..3, 1..5] real, // Двумерный массив
      C: [1..3][1..5] real; // Массив массивов
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Элементы языка - Цикл:

- **Синтаксис**

  for-loop:

  for index-expr in iterator-expr { stmt-list }

- **Семантика**

  Выполнение тела цикла для каждого значения переменной цикла

- **Примеры**

```
- var A: [1..3] string = (“DO“, “RE“, “MI“);
- for i in 1..3 do write(A(i));          // DOREMI
- for a in A { a += “LA“; write(a); } // DOLARELAMILA
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Элементы языка - Условия:

- **Условие**

```
if cond then computeA() else computeB();
```

- **Цикл while**

```
while cond {
  compute();
}
```

- **Выбор**

```
select key {
  when value1 do compute1();
  when value2 do compute2();
  otherwise compute3();
}
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Элементы языка - Функции:

- **Пример**

```
def area(radius: real)
  return 3.14 * radius**2;

writeln(area(2.0)); // 12.56
```

- **Пример функции с значениями по умолчанию**

```
def writeCoord(x: real = 0.0, y: real = 0.0) {
  writeln("(", x, ", ", y, ")");
}
writeCoord(2.0);   // (2.0, 0.0)
writeCoord(y=2.0); // (0.0, 2.0)
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Параллелизм задач: Оператор begin

- **Синтаксис**

  begin-stmt:

  begin *stmt*

- **Семантика**
  - Создает параллельную задачу для выполнения **stmt**
  - Выполнение программы-родителя не останавливается
- **Пример**

```
begin writeln("hello world");
writeln("good bye");
```

Возможный вывод

```
(1) hello world   (2) good bye
    good bye          hello world
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Параллелизм задач: Тип sync

- **Синтаксис**

  sync-type:

  sync ***type***

- **Семантика**
  - Переменные типа **sync** имеют два состояния – «full», «empty»
  - Запись в переменную типа **sync** переводит ее в состояние «full»
  - Чтение из переменной типа **sync** переводит в состояние «empty»

  - **Примеры**

(1)
```
var lock$: sync bool;
lock$ = true;
critical();
lock$;
```

(2)
```
var future$: sync real;
begin future$ = compute();
computeSomethingElse();
useComputeResults(future$);
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Параллелизм задач: Оператор cobegin

- **Синтаксис**

  cobegin-stmt:

  cobegin { *stmt-list* }

- **Семантика**
  - Порождает параллельную задачу для каждого оператора stmt-list
  - Синхронизация при завершении блока
  - **Пример**

```
cobegin {
  consumer(1);
  consumer(2);
  producer();
}
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Параллелизм задач: Оператор coforall

- **Синтаксис**

```
coforall-loop:
  coforall index-expr in iterator-expr { stmt }
```

- **Семантика**
  - Порождает параллельных задач для каждой итерации цикла
  - Синхронизация при завершении цикла
  - **Пример** (задача «производитель-потребитель»)

```
begin producer();
coforall i in 1..numConsumers {
  consumer(i);
}
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Редукция данных: Операция reduce

- **Синтаксис**

```
reduce-expr:
  reduce-op reduce iterator-expr
```

- **Семантика**

  Применяет для каждого элемента данных операцию **reduce-op**

  - **Пример**

```
total = + reduce A;
bigDiff = max reduce [i in InnerD] abs(A(i)-B(i));
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Редукция данных : Операция scan

- **Синтаксис**

  scan-expr:

  scan-op ***scan*** *iterator-expr*

- **Семантика**

  Применяет для каждого элемента данных операцию **reduce-op** (с получением всех частных результатов)

  - **Пример**

```
var A, B, C: [1..5] int;
A = 1;                  // A: 1 1 1 1 1
B = + scan A;           // B: 1 2 3 4 5
B(3) = -B(3);           // B: 1 2 -3 4 5
C = min scan B; // C: 1 1 -3 -3 -3
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Модель вычислительной системы: Понятие locale

- **Определение**
  - Представляет абстракцию вычислительного элемента
  - Содержит обрабатывающее устройство и память
- **Свойства**
  - Задачи, исполняемые на locale, имеют однородный доступ к локальной памяти
  - Доступ к памяти других locale происходит более медленно
- **Пример**

locale – SMP, многоядерный процессор

Перевод – locale, локал, **исполнитель**,… !?

Разработка параллельных программ для основе модели PGAS: Chapel…

Модель вычислительной системы

- **Описание множества всех locale**

```
config const numLocales: int;
const LocaleSpace: domain(1) = [0..numLocales-1];
const Locales: [LocaleSpace] locale;
```

- **Определение множества locale при запуске**

```
prompt> a.out --numLocales=8
```

- **Определение топологии множества local**

```
var TaskALocs = Locales[0..1];
var TaskBLocs = Locales[2..numLocales-1];
var Grid2D = Locales.reshape([1..2,1..4]);
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Модель вычислительной системы: Операции

- **Получение индекса locale**

```
def locale.id: int { ... }
```

- **Получение имени locale**

```
def locale.name: string { ... }
```

- **Получение количества ядер в locale**

```
def locale.numCores: int { ... }
```

- **Получение размера доступной памяти в locale**

```
def locale.physicalMemory(...) { ... }
```

- **Пример – получение размера всей памяти**

```
const totalSystemMemory =
  + reduce Locales.physicalMemory();
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Модель вычислительной системы: Связывание задач и locale

- **Связывание задач и locale – операция on**

```
on-stmt:
  on expr { stmt }
```

- **Семантика**

Выполнение `stmt` на locale, определяемый `expr`

- **Пример**

```
var A: [LocaleSpace] int;
coforall loc in Locales do on loc do
  A(loc.id) = compute(loc.id);
```

Разработка параллельных программ для основе модели PGAS: Chapel…

Модель вычислительной системы: Пример

```
var x, y: real; // x и y на locale 0
on Locales(1) { // migrate task to locale 1
  var z: real;  // z на locale 1
  z = x + y;    // удаленный доступ к x и y
  on Locales(0) do // возврат на locale 0
    z = x + y;  // удаленный доступ к z
  // возврат на locale 1
  on x do         // переход на locale 0
    z = x + y;    // удаленный доступ к z
  // переход на locale 1
} // возврат на locale 0
```

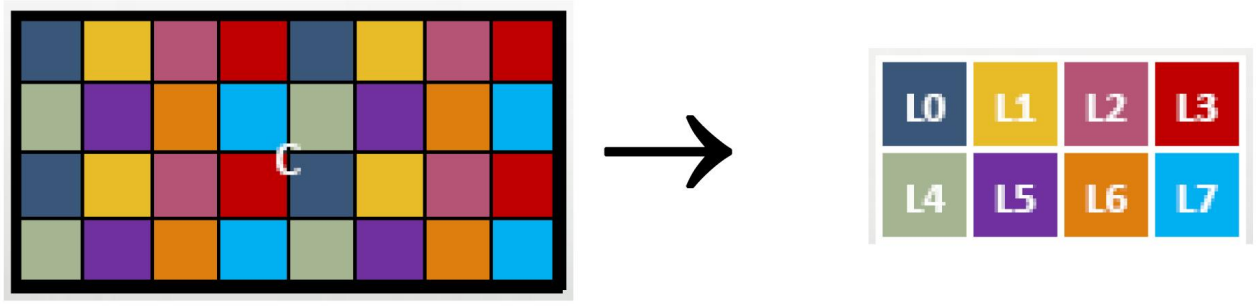
Разработка параллельных программ для основе модели PGAS: Chapel…

Модель вычислительной системы: Распределение данных

- Распределение диапазонов (данных) между множеством locale: пример

```
const Dist = new dmap(new Cyclic(startIdx=(1,1)));
varDom: domain(2) dmapped Dist = [1..4, 1..8];
```

Данные распределяются на решетке locale

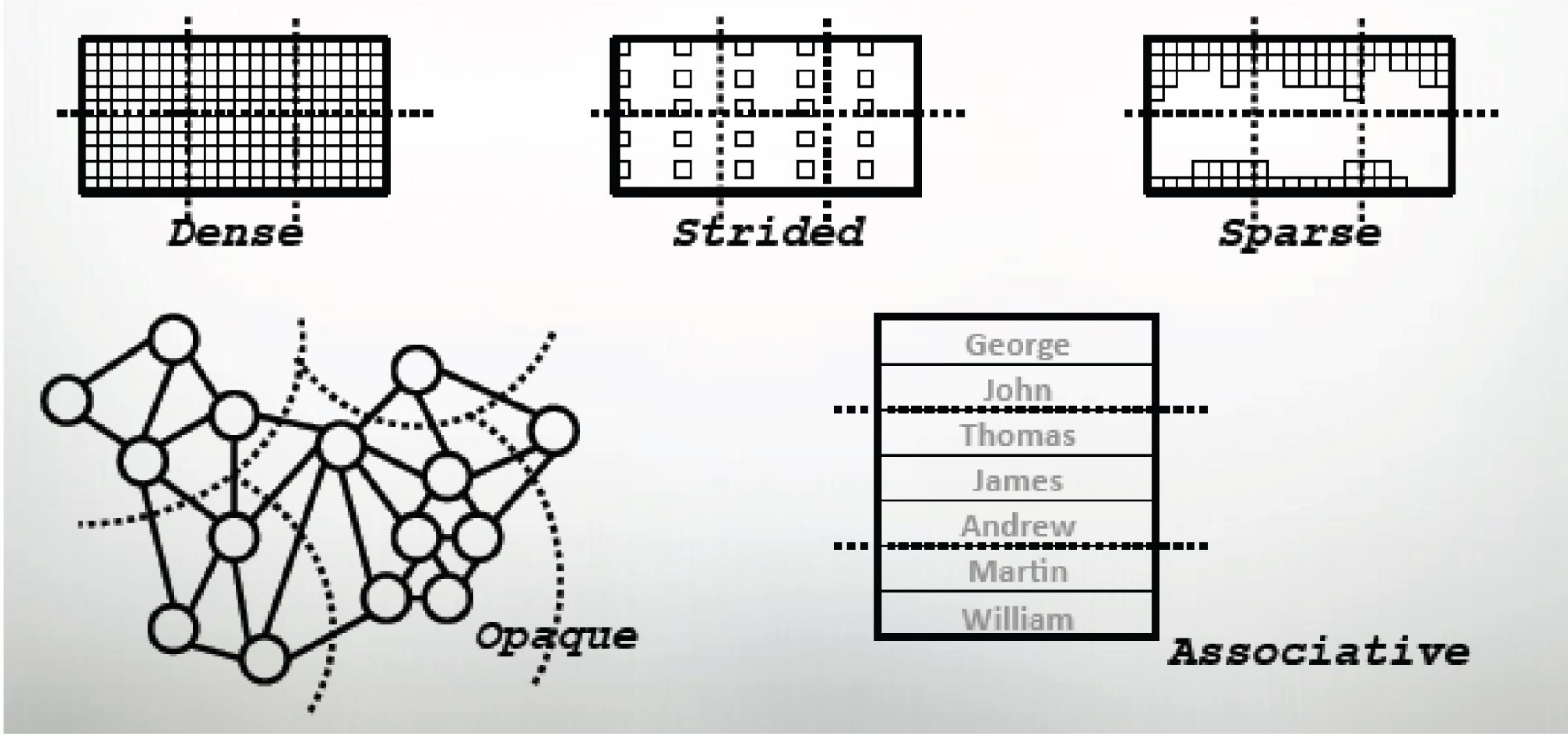


- Имеется библиотека распределений
- Возможно создание новых распределений

Разработка параллельных программ для основе модели PGAS: Chapel…

Модель вычислительной системы: Распределение данных

- Распределение диапазонов осуществляется по единой схеме для всех типов диапазонов



СПАСИБО ЗА ВНИМАНИЕ