

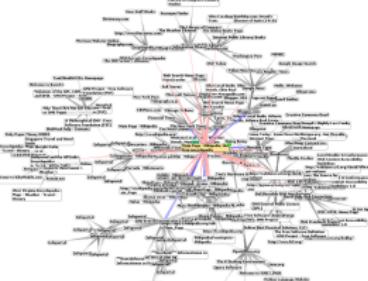
# Трансляция проблемно-ориентированного языка Green-Marl в параллельный код на Charm++

[ на примере задачи поиска сильно связных компонент в  
ориентированном графе ]

Александр Фролов, Алексей Симонов



# Большие графы в реальном мире



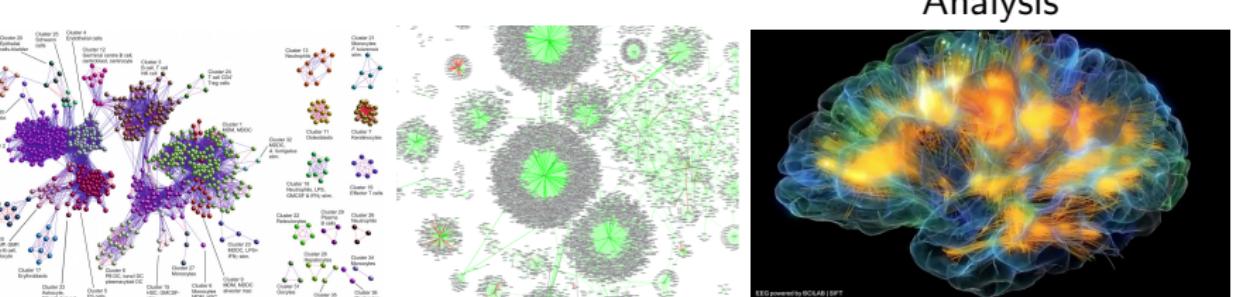
WEB-graph analysis



Social Network Analysis



Road Networks  
Analysis



Bioinformatics

Cybersecurity



Human Brain Project

## Параллельный анализ графов: проблема продуктивности

- Общие трудности параллельного программирования
  - разработка эффективных параллельных алгоритмов – это сложно (аксиома!)
  - зависимость от архитектуры вычислительной системы

## Параллельный анализ графов: проблема продуктивности

- Общие трудности параллельного программирования
  - разработка эффективных параллельных алгоритмов – это сложно (аксиома!)
  - зависимость от архитектуры вычислительной системы
- Специфические проблемы графовых задач
  - агрегация коротких сообщений
  - распределение графа по вычислительным узлам
  - динамическая балансировка вычислений

## Параллельный анализ графов: проблема продуктивности

- Общие трудности параллельного программирования
  - разработка эффективных параллельных алгоритмов – это сложно (аксиома!)
  - зависимость от архитектуры вычислительной системы
- Специфические проблемы графовых задач
  - агрегация коротких сообщений
  - распределение графа по вычислительным узлам
  - динамическая балансировка вычислений
- Отсутствует стандартная библиотека параллельного анализа графов!
  - Boost Parallel Graph Library (только если Вы – C++ гуру! или мечтаете им стать)
  - GraphBLAS (пока еще в ранней стадии разработки)

# Параллельный анализ графов: проблема продуктивности

- Общие трудности параллельного программирования
  - разработка эффективных параллельных алгоритмов – это сложно (аксиома!)
  - зависимость от архитектуры вычислительной системы
- Специфические проблемы графовых задач
  - агрегация коротких сообщений
  - распределение графа по вычислительным узлам
  - динамическая балансировка вычислений
- Отсутствует стандартная библиотека параллельного анализа графов!
  - Boost Parallel Graph Library (только если Вы – C++ гуру! или мечтаете им стать)
  - GraphBLAS (пока еще в ранней стадии разработки)
- Оценка относительных усилий на разработку параллельных графовых задач (в #LOC)

	Seq. (C)	OpenMP+C	MPI+C	Charm++	Giraph
BFS	54	80-100	155	70-80	50
SSSP	50	90	300-500	70-80	53
CC	40	44	100-200	70-80	52
SCC	46	40-50	100-200	100-200	122
Betw.Cent.	100	115	300-500	?	-
PageRank	30	37	60	70-80	100-180

# Green-Marl

- Green-Marl – проблемно-ориентированный язык (DSL) для разработки императивных параллельных алгоритмов анализа графов
- Разработан в PPL @ Stanford University
  - DSL спецификация & GM компилятор с генерацией C++/OpenMP кода [ASPLOS 2012]<sup>1</sup>
  - Поддержка Pregel (GPS, Giraph) [FOSDEM 2013]<sup>2</sup>
  - <https://github.com/stanford-ppl/Green-Marl>
- Интегрирован в PGX.D (Orable Labs)
  - Поддержка PGX.D [SC15]<sup>3</sup>



<sup>1</sup>Hong, S., Chafi, H., Sedlar, E., & Olukotun, K. (2012, March). Green-Marl: a DSL for easy and efficient graph analysis. In ACM SIGARCH Computer Architecture News (Vol. 40, No. 1, pp. 349-362). ACM.

<sup>2</sup>Hong S. et al. Simplifying scalable graph processing with a domain-specific language //Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization. – ACM, 2014. – С. 208.

<sup>3</sup>Sevenich, M., Hong, S., van Rest, O., Wu, Z., Banerjee, J., & Chafi, H. (2016). Using domain-specific languages for analytic graph databases. Proceedings of the VLDB Endowment, 9(13), 1257-1268.

# Пример программы на Green-Marl

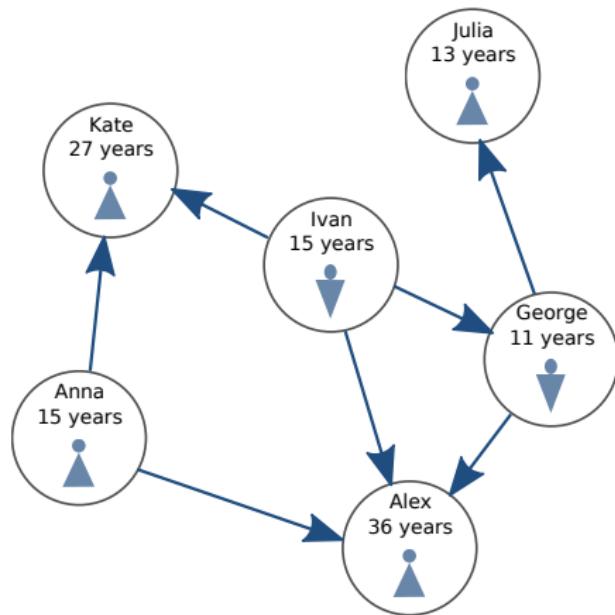
Query: How cool is your daddy? (c)

## Анализ социальных сетей:

Подсчитать среднее количество подписчиков в возрасте от 10 до 20 лет для пользователей старше, чем K.

```
Procedure avg_teen_cnt(G: Graph,  
    age, teen_cnt: N_P<Int>,  
    K: Int) : Float  
{  
    Foreach(n: G.Nodes) {  
        n.teen_cnt = Count(t:n.InNbrs)  
        (t.age>=10 && t.age<20);  
    }  
  
    Float avg = (Float) Avg(n: G.Nodes)  
    (n.age>K){n.teen_cnt};  
    Return avg;  
}
```

#LOC=10



# Язык параллельного программирования Charm++

- История

- Parallel Programming Laboratory at the University of Illinois
- создание – начало 90-х годов
- текущая версия 6.7.1

# Язык параллельного программирования Charm++

- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов
  - текущая версия 6.7.1
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)

# Язык параллельного программирования Charm++

- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов
  - текущая версия 6.7.1
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)
- Специфические возможности
  - динамическая балансировка нагрузки
  - поддержка отказоустойчивости (сохранение контрольных точек)

# Язык параллельного программирования Charm++

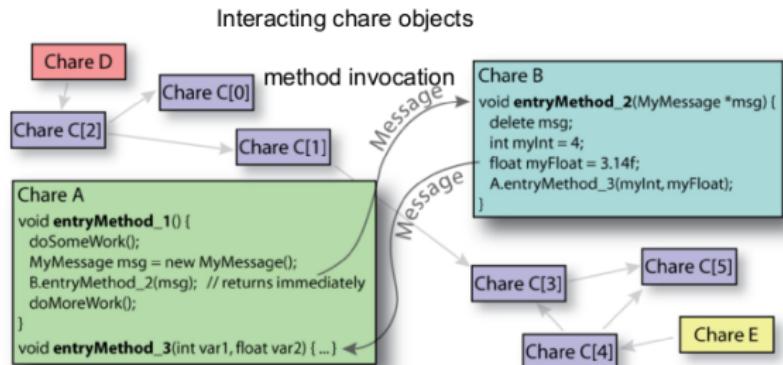
- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов
  - текущая версия 6.7.1
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)
- Специфические возможности
  - динамическая балансировка нагрузки
  - поддержка отказоустойчивости (сохранение контрольных точек)
- Поддерживаемые типы HPC-систем
  - SMP-узлы с NUMA памятью
  - кластеры с Infiniband, BlueGene/P, BlueGene/Q, Cray XK, Cray XC
  - ведутся работы по поддержке ускорителей (Xeon Phi, GPU)

# Язык параллельного программирования Charm++

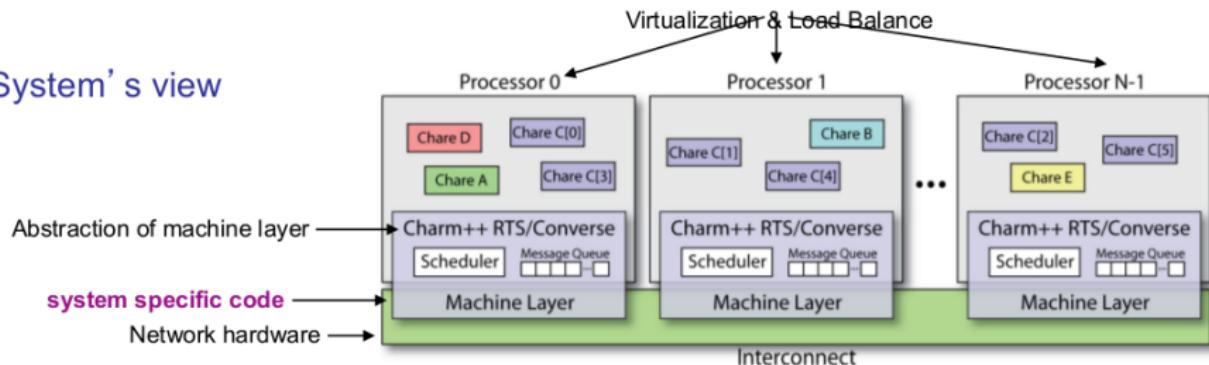
- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов
  - текущая версия 6.7.1
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)
- Специфические возможности
  - динамическая балансировка нагрузки
  - поддержка отказоустойчивости (сохранение контрольных точек)
- Поддерживаемые типы HPC-систем
  - SMP-узлы с NUMA памятью
  - кластеры с Infiniband, BlueGene/P, BlueGene/Q, Cray XK, Cray XC
  - ведутся работы по поддержке ускорителей (Xeon Phi, GPU)
- Приложения
  - NAMD, OpenAtom, ChANGa, EpiSimdemics
  - BigSim, ClothSim, имитационная модель сети "Ангара"

# Программная модель Charm++

Application's view



System's view



## Почему портировать Green-Marl на Charm++ имеет смысл

- В open-source версии Green-Marl нет поддержки HPC кластеров

## Почему портировать Green-Marl на Charm++ имеет смысл

- В open-source версии Green-Marl нет поддержки HPC кластеров
- Charm++ – зрелая система параллельного программирования с достаточно активным сообществом пользователей

## Почему портировать Green-Marl на Charm++ имеет смысл

- В open-source версии Green-Marl нет поддержки HPC кластеров
- Charm++ – зрелая система параллельного программирования с достаточно активным сообществом пользователей
- Charm++ показывает хорошую масштабируемость при запусках с использованием большого количества узлов

## Почему портировать Green-Marl на Charm++ имеет смысл

- В open-source версии Green-Marl нет поддержки HPC кластеров
- Charm++ – зрелая система параллельного программирования с достаточно активным сообщество пользователей
- Charm++ показывает хорошую масштабируемость при запускам с использованием большого количества узлов
- Программная модель Charm++ превосходно подходит для реализации (разработки) графовых алгоритмов в вершинно-ориентированном (vertex-centric) стиле

## Почему портировать Green-Marl на Charm++ имеет смысл

- В open-source версии Green-Marl нет поддержки HPC кластеров
- Charm++ – зрелая система параллельного программирования с достаточно активным сообществом пользователей
- Charm++ показывает хорошую масштабируемость при запусках с использованием большого количества узлов
- Программная модель Charm++ превосходно подходит для реализации (разработки) графовых алгоритмов в вершинно-ориентированном (vertex-centric) стиле
- Charm++ поддерживает динамическую балансировку нагрузки

## Почему портировать Green-Marl на Charm++ имеет смысл

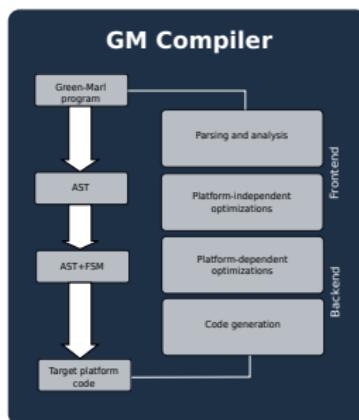
- В open-source версии Green-Marl нет поддержки HPC кластеров
- Charm++ – зрелая система параллельного программирования с достаточно активным сообщество пользователей
- Charm++ показывает хорошую масштабируемость при запускам с использованием большого количества узлов
- Программная модель Charm++ превосходно подходит для реализации (разработки) графовых алгоритмов в вершинно-ориентированном (vertex-centric) стиле
- Charm++ поддерживает динамическую балансировку нагрузки
- В компиляторе Green-Marl реализована поддержка платформ на базе модели Pregel (Giraph, Stanford GPS), что сильно упрощает портирование Green-Marl на Charm++

# Трансляция Green-Marl в Charm++/Pregel

- Основная сложность – различие программных моделей

**Green-Marl**  
DLP/PRAM DSL

```
Forall (n in G.Nodes) {  
    Forall (v in n.Nbrs) {  
        ...  
    }  
}
```



**Charm++**  
Asynchronous Message-driven Parallel Programming Language

```
class Vertex : ... {  
    ...  
    /*entry*/ void foo() {...}  
    /*entry*/ void boo() {...}  
}
```

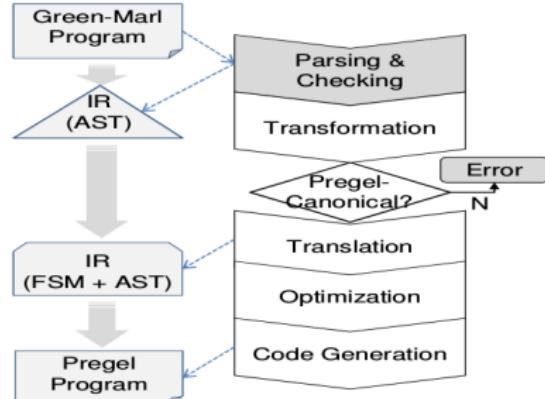
**Google Pregel**  
Vertex-centric, Bulk-Synchronous Parallel Framework

```
class Master : ... {  
    ...  
    void compute() {  
        switch(state) {  
            ...  
        }  
    }  
};  
class Vertex : ... {  
    ...  
    void compute() {  
        switch(state) {  
            ...  
        }  
    }  
}
```

# Green-Marl @ Pregel

## Pregel-canonical GM apps features:

- Finite State Management
  - GM program is non-recursive, at least on directed graph in parameters, any number of While and If-Then-Else constructs.
- Parallel Vertex & Neighborhood Iteration
  - Foreach loops can be only (at most) doubly nested: outer loop iterates over nodes, inner loop iterates over neighbours.
- Message Pushing
  - In Foreach loops that iterate over  $u$  neighbours it is not allowed to write to  $u$  attributes.
- Random Writing
  - It is allowed to randomly write to vertices properties in Foreach loops, random reading is not allowed.
- Edge Property
  - The property of the edge  $(u, v)$  is only accessed in  $u$ .



## Green-Marl compiler stages:

- Syntax Expansion
- Loop Dissection
- Edge Flipping
- Loop Merging
- State Extraction
- State Merging

Non Pregel-canonical GM apps → transformed to canonical (if possible)

# Green-Marl @ Charm++

## Example: Avg Teen Followers (1/4)

### Green-Marl (original, non Pregel-canonical)

```
Procedure avg_teen_cnt(G: Graph,
    age, teen_cnt: N_P<Int>,
    K: Int) : Float
{
    Foreach(n: G.Nodes) {
        n.teen_cnt = Count(t:n.InNbrs
            (t.age>=10 && t.age<20));
    }

    Float avg = (Float) Avg(n: G.Nodes
        (n.age>K){n.teen_cnt});
    Return avg;
}
```

### Green-Marl (transformed, Pregel-canonical)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int) : Float
{
    __S2 = 0;
    _cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n.__S1prop = 0;
    }
    Foreach (t : G.Nodes)
        If (((t.age >= 10) && (t.age < 20) ) )
    {
        Foreach (n : t.Nbrs)
        {
            n.__S1prop += 1 @ t ;
        }
    }
    Foreach (n : G.Nodes)
    {
        n.teen_cnt = n.__S1prop;
        If ((n.age > K) )
        {
            __S2 += n.teen_cnt @ n ;
            _cnt3 += 1 @ n ;
        }
    }
    _avg4 = (0 == _cnt3) ?
        0.000000 : (___S2 / (Double) _cnt3) ;
    avg = (Float) _avg4;
    Return avg;
}
```

### Green-Marl compiler stages:

- Syntax Expansion
- Loop Dissection
- Edge Flipping
- Loop Merging
- State Extraction
- State Merging

# Green-Marl @ Charm++

## Example: Avg Teen Followers (1/4)

Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    _S2 = 0;
    _cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n._S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10)  && (t.age < 20) )
        {
            Foreach (n : t.Nbrs)
            {
                n._S1prop += 1 @ t ;
            }
        }
        Foreach (n : G.Nodes)
        {
            n.teen_cnt = n._S1prop;
            If ((n.age > K) )
            {
                _S2 += n.teen_cnt @ n ;
                _cnt3 += 1 @ n ;
            }
        }
        _avg4 = (0 == _cnt3)  ?
            0.000000 : (_S2 / (Double ) _cnt3)  ;
        avg = (Float ) _avg4;
        Return avg;
    }
}
```

# Green-Marl @ Charm++

## Example: Avg Teen Followers (1/4)

### Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    __S2 = 0;
    __cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n.__S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10)  && (t.age < 20) )  )
        {
            Foreach (n : t.Nbrs)
            {
                n.__S1prop += 1 @ t ;
            }
        }
        Foreach (n : G.Nodes)
        {
            n.teen_cnt = n.__S1prop;
            If ((n.age > K) )
            {
                __S2 += n.teen_cnt @ n ;
                __cnt3 += 1 @ n ;
            }
        }
        __avg4 = (0 == __cnt3)  ?
            0.000000 : (__S2 / (Double ) __cnt3) ;
        avg = (Float ) __avg4;
        Return avg;
    }
```

S0 (SEQ)

S1 (PAR)

S2 (PAR)

S3 (PAR)

S4 (SEQ)

# Green-Marl @ Charm++

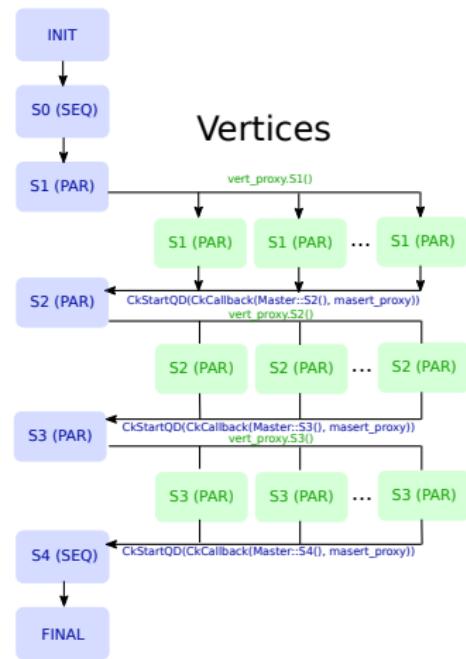
## Example: Avg Teen Followers (1/4)

### Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    __S2 = 0;
    _cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n.__S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10) && (t.age < 20) ) )
        {
            Foreach (n : t.Nbrs)
            {
                n.__S1prop += 1 @ t ;
            }
        }
    }
    Foreach (n : G.Nodes)
    {
        n.teen_cnt = n.__S1prop;
        If ((n.age > K) )
        {
            __S2 += n.teen_cnt @ n ;
            _cnt3 += 1 @ n ;
        }
    }
    _avg4 = (0 == _cnt3) ?
        0.000000 : (__S2 / (Double ) _cnt3) ;
    avg = (Float ) _avg4;
    Return avg;
}
```

### State Machine

#### Master



# Green-Marl @ Charm++

## Example: Avg Teen Followers (1/4)

### Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    __S2 = 0;
    __cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n.__S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10) && (t.age < 20) ) )
        {
            Foreach (n : t.Nbrs)
            {
                n.__S1prop += 1 @ t ;
            }
        }
        Foreach (n : G.Nodes)
        {
            n.teen_cnt = n.__S1prop;
            If ((n.age > K) )
            {
                __S2 += n.teen_cnt @ n ;
                __cnt3 += 1 @ n ;
            }
        }
        avg4 = (0 == __cnt3) ?
            0.000000 : (__S2 / (Double) __cnt3) ;
        avg = (Float) avg4;
        Return avg;
    }
```

S0 (SEQ)

S1 (PAR)

S2 (PAR)

S3 (PAR)

S4 (SEQ)

### Charm++ (generated)

```
#include "avg_teen_count.decl.h"
class avg_teen_count_vertex :
    public CBase_avg_teen_count_vertex {
    ...
    private:
        int age;
        int teen_count;
        int __S1prop;
    public:
    ...
};

class avg_teen_count_master :
    public CBase_avg_teen_count_master {
    ...
    private:
        int __S2;
        int __cnt3;
        int K;
        float avg;
        double __avg4;
    public:
    ...
    /*entry*/ void __ep_state_0 () {
        __S2 = 0;
        __cnt3 = 0;
        thisProxy.__ep_state_1();
    }
    ...
};

#include "avg_teen_count.decl.h"
```

# Green-Marl @ Charm++

## Example: Avg Teen Followers (2/4)

### Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    __S2 = 0;
    _cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n.__S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10) && (t.age < 20) ) )
        {
            Foreach (n : t.Nbrs)
            {
                n.__S1prop += 1 @ t ;
            }
        }
        Foreach (n : G.Nodes)
        {
            n.teen_cnt = n.__S1prop;
            If ((n.age > K) )
            {
                __S2 += n.teen_cnt @ n ;
                _cnt3 += 1 @ n ;
            }
        }
    }
    _avg4 = (0 == _cnt3) ?
        0.000000 : (__S2 / (Double) _cnt3) ;
    avg = (Float) _avg4;
    Return avg;
}
```

S0 (SEQ)

S1 (PAR)

S2 (PAR)

S3 (PAR)

S4 (SEQ)

### Charm++ (generated)

```
#include "avg_teen_count.decl.h"
class avg_teen_count_vertex :
    public CBase_avg_teen_count_vertex {
    ...
/*entry*/ void __ep_state_1 () {
    __S1prop = 0;
}
...
};

class avg_teen_count_master : Collective Vertex call
    public CBase_avg_teen_count_master {
    ...
/*entry*/ void __ep_state_1 () {
    graph_proxy.__ep_state_1();
    CkStartQD(CkCallback(
        CkIndex_avg_teen_count_master::__ep_state_2(),
        thisProxy));
}
};

#include "avg_teen_count.decl.h"
```

**Collective Vertex call**

**Quiescence Detection**

# Green-Marl @ Charm++

## Example: Avg Teen Followers (3/4)

### Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    _S2 = 0;
    _cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n._S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10) && (t.age < 20) ) )
        {
            Foreach (n : t.Nbrs)
            {
                n._S1prop += 1 @ t ;
            }
        }
        Foreach (n : G.Nodes)
        {
            n.teen_cnt = n._S1prop;
            If ((n.age > K) )
            {
                _S2 += n.teen_cnt @ n ;
                _cnt3 += 1 @ n ;
            }
        }
    }
    _avg4 = (0 == _cnt3) ?
        0.000000 : (_S2 / (Double) _cnt3) ;
    avg = (Float) _avg4;
    Return avg;
}
```

S0 (SEQ)

S1 (PAR)

S2 (PAR)

S3 (PAR)

S4 (SEQ)

### Charm++ (generated)

```
#include "avg_teen_count.decl.h"
class avg_teen_count_vertex :
    public CBase_avg_teen_count_vertex {
    ...
    /*entry*/ void __ep_state_2 () {
        if ((age >= 10) && (age < 20) ) {
            for (Edges::Iterator i = edges.begin;
                i != edges.end(); i++) {
                thisProxy[i->v].__ep_state_2_recv();
            }
        }
    /*entry*/ void __ep_state_2_recv () {
        teen_count = teen_count + 1;
    }
    ...
};

class avg_teen_count_master :
    public CBase_avg_teen_count_master {
    ...
    /*entry*/ void __ep_state_2 () {
        graph_proxy.__ep_state_2();
        CkStartQD(CkCallback(
            CkIndex_avg_teen_count_master::__ep_state_3(),
            thisProxy));
    }
};

#include "avg_teen_count.decl.h"
```

Call to Nbrs

# Green-Marl @ Charm++

## Example: Avg Teen Followers (4/4)

### Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    _S2 = 0;
    _cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n._S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10) && (t.age < 20) ) )
        {
            Foreach (n : t.Nbrs)
            {
                n._S1prop += 1 @ t ;
            }
        }
        Foreach (n : G.Nodes)
        {
            n.teen_cnt = n._S1prop;
            If ((n.age > K) )
            {
                _S2 += n.teen_cnt @ n ;
                _cnt3 += 1 @ n ;
            }
        }
    }
    _avg4 = (0 == _cnt3) ?
        0.000000 : (_S2 / (Double ) _cnt3) ;
    avg = (Float ) _avg4;
    Return avg;
}
```

S0 (SEQ)

S1 (PAR)

S2 (PAR)

S3 (PAR)

S4 (SEQ)

### Charm++ (generated)

```
class avg_teen_count_vertex :
    public CBase_avg_teen_count_vertex {
    /*entry*/ void __ep_state_3 ( __Message_state_3 *m) {
        int K = m->K;
        long _cnt3;
        int _S2;
        delete m;
        if (age > K) {
            _S2 = teen_cnt;
            contribute(sizeof(int), &_S2,
                CkReduction::sum_int,
                CkCallback(CkReductionTarget(
                    avg_teen_cnt_master, __reduction__S2),
                    master_proxy));
            _cnt3 = 1;
            contribute(sizeof(long), &_cnt3,
                CkReduction::sum_long,
                CkCallback(CkReductionTarget(
                    avg_teen_cnt_master, __reduction__cnt3),
                    master_proxy));
        }
    }
    class avg_teen_count_master :
        public CBase_avg_teen_count_master {
    /*entry*/ void __ep_state_3 () { ... }
    /*entry, reduct*/ void __reduction__S2 (int t) { ... }
    /*entry, reduct*/ void __reduction__cnt3 (long t) { ... }
};
```

reduction

# Green-Marl @ Charm++

## Example: Avg Teen Followers (5/5)

### Green-Marl (transformed)

```
Procedure avg_teen_cnt( G : Graph,
    age : N_P <Int>(G), teen_cnt : N_P <Int>(G),
    K : Int ) : Float
{
    _S2 = 0;
    _cnt3 = 0;
    Foreach (n : G.Nodes)
    {
        n._S1prop = 0;
    }
    Foreach (t : G.Nodes)
    {
        If (((t.age >= 10) && (t.age < 20) ) )
        {
            Foreach (n : t.Nbrs)
            {
                n._S1prop += 1 @ t ;
            }
        }
        Foreach (n : G.Nodes)
        {
            n.teen_cnt = n._S1prop;
            If ((n.age > K) )
            {
                _S2 += n.teen_cnt @ n ;
                _cnt3 += 1 @ n ;
            }
        }
    }
    avg4 = (0 == _cnt3) ?
        0.000000 : (_S2 / (Double ) _cnt3) ;
    avg = (Float ) avg4;
    Return avg;
}
```

S0 (SEQ)

S1 (PAR)

S2 (PAR)

S3 (PAR)

S4 (SEQ)

### Charm++ (generated)

```
class avg_teen_count_vertex :
    public CBase_avg_teen_count_vertex {
    ...
};

class avg_teen_count_master :
public CBase_avg_teen_count_master {
/*entry*/ void __ep_state_3 () {
    _avg4 = (0 == _cnt3)?((float)(0.000000)):
        (_S2 / ((double)_cnt3));
    avg = (float)_avg4;
    done_callback.send();
}
};
```

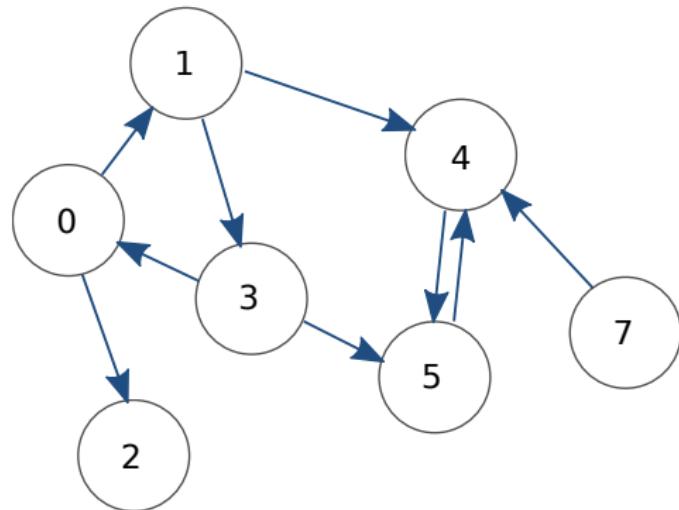
Callback to boilerplate code

# Поиск сильно связных компонент (SCC) (1/7)

Реализация алгоритма на основе раскраски графа на Green-Marl

0: Инициализация:

```
Foreach (v : G.Nodes) {  
    v.SCC = -1;  
}
```

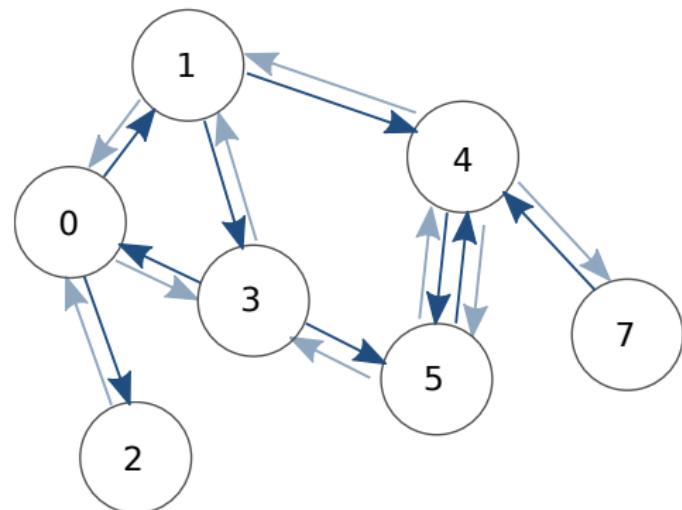


## Поиск сильно связных компонент (SCC) (2/7)

Реализация алгоритма на основе раскраски графа на Green-Marl

1: Построение транспонированного графа:

```
Foreach (v : G.Nodes){v.SCC == -1} {  
    Foreach (u : v.Nbrs) {  
        u.transEdges.add(v);  
    }  
}
```

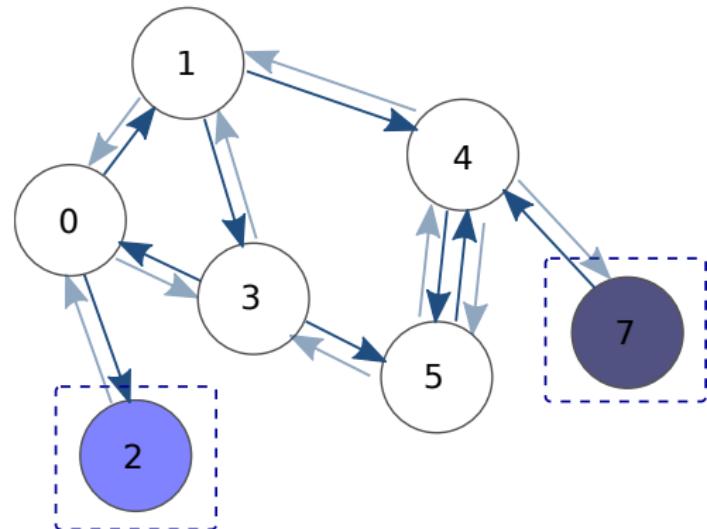


## Поиск сильно связных компонент (SCC) (3/7)

Реализация алгоритма на основе раскраски графа на Green-Marl

2: Поиск "висящих" вершин:

```
Foreach (v : G.Nodes){v.SCC == -1}{{  
    If (v.edges.empty() == 0 ||  
        v.transEdges.empty())  
        v.SCC = v;  
}
```



## Поиск сильно связных компонент (SCC) (4/7)

Реализация алгоритма на основе раскраски графа на Green-Marl

### 3: Поиск слабо связных компонент:

```

Foreach (v : G.Nodes){v.SCC == -1}{

    v.Color = v;
    v.updated = True;
    v.updated_next = False;

}

While (!done) {

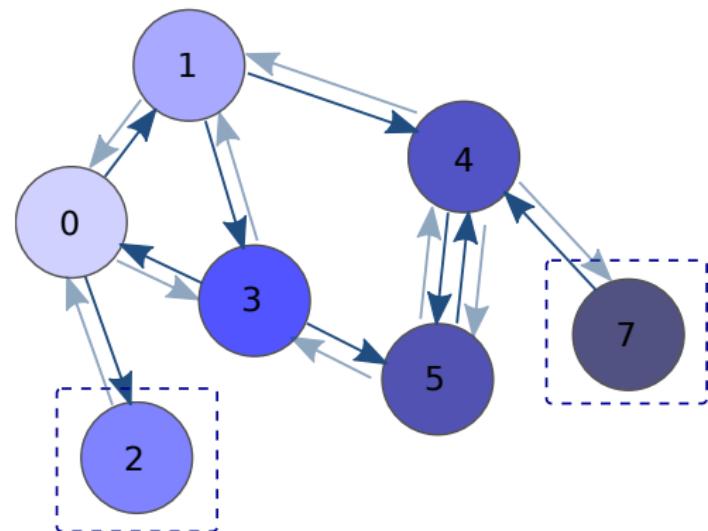
    Foreach (v : G.Nodes)
        {v.SCC == -1 && v.updated}{

            Foreach (u : v.Nbrs){
                If (v.Color < u.Color) {
                    u.Color = v.Color;
                    u.updated_next = True;
                }
            }

            G.updated = G.updated_next;
            G.updated_next = False;
            done = !Exist(v in V){v.updated};

        }
}

```



# Поиск сильно связных компонент (SCC) (5/7)

Реализация алгоритма на основе раскраски графа на Green-Marl

3: Поиск слабо связных компонент:

```
Foreach (v : G.Nodes){v.SCC == -1}{

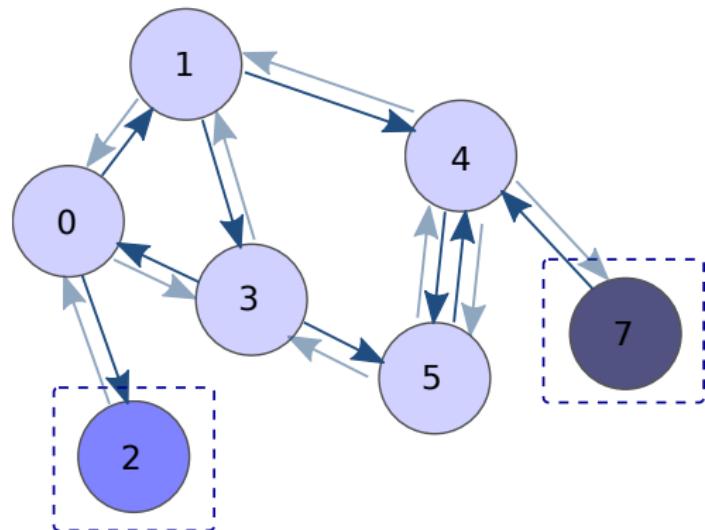
    v.Color = v;
    v.updated = True;
    v.updated_next = False;
}

While (!done) {

    Foreach (v : G.Nodes)
        {v.SCC == -1 && v.updated}{

            Foreach (u : v.Nbrs){
                If (v.Color < u.Color) {
                    u.Color = v.Color;
                    u.updated_next = True;
                }
            }

            G.updated = G.updated_next;
            G.updated_next = False;
            done = !Exist(v in V){v.updated};
        }
}
```



# Поиск сильно связных компонент (SCC) (6/7)

Реализация алгоритма на основе раскраски графа на Green-Marl

4: Поиск обратно-достижимых вершин от любой вершины, принадлежащей слабо связной компоненте

```
Foreach (v : G.Nodes){v.SCC == -1}{

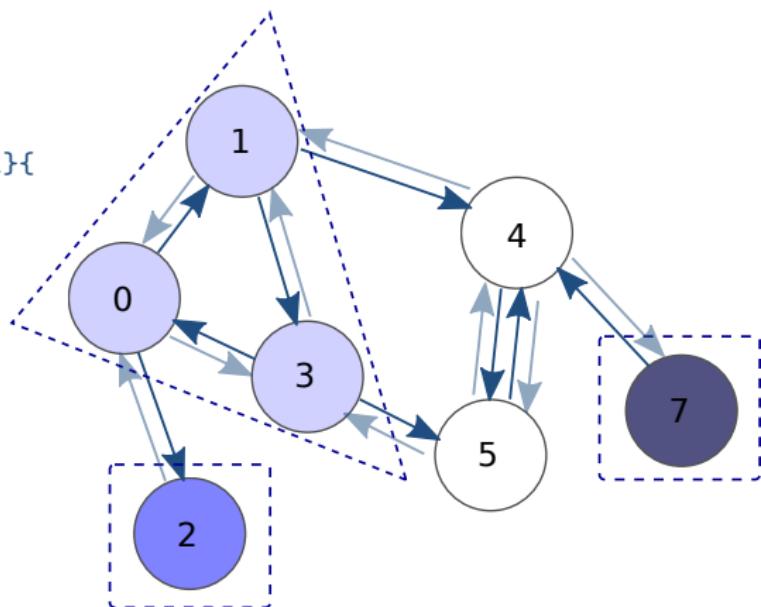
    If (v.Color == v)
        v.updated = True;
        v.updated_next = False;
    }

    While (!done) {
        Foreach (v : G.Nodes)
            {v.SCC == -1 && v.updated}{

                v.SCC = v.Color
                Foreach (u : v.Nbrs){
                    If (v.Color == u.Color)
                        u.updated_next = True;
                }

                G.updated = G.updated_next;
                G.updated_next = False;
                done = !Exist(v in V){v.updated};

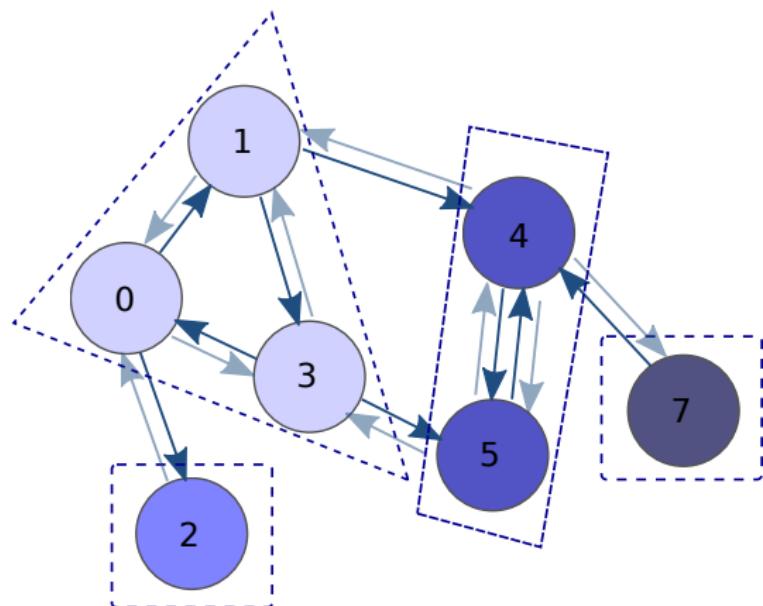
            }
    }
}
```



# Поиск сильно связных компонент (SCC) (7/7)

Реализация алгоритма на основе раскраски графа на Green-Marl

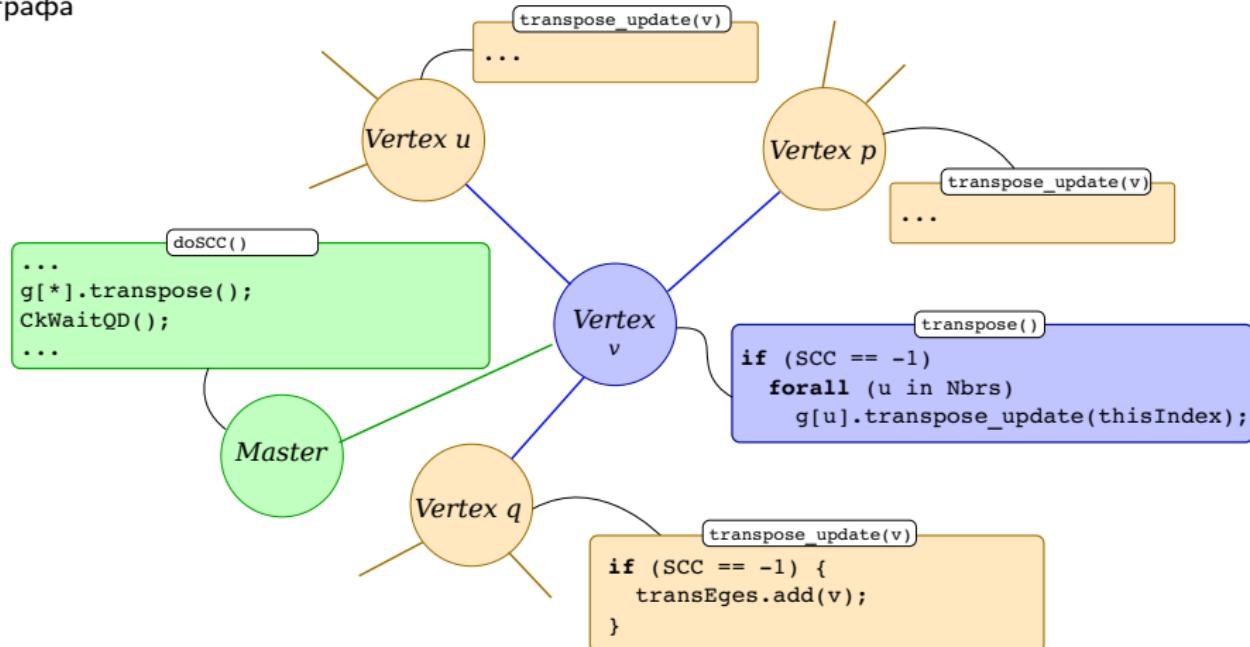
- 0: Инициализация
- 1: Построение транспонированного графа
- 2: Поиск “висящих” вершин
- 3: Поиск слабо связных компонент
- 4: Поиск обратно-достижимых вершин от любой вершины, принадлежащей слабо связной компоненте
- 5: Если есть хоть одна вершина, такая что  $v.\text{SCC} == -1$ , вернуться к шагу 1



# Поиск сильно связных компонент (SCC) (1/4)

Реализация алгоритма на основе раскраски графа на Charm++

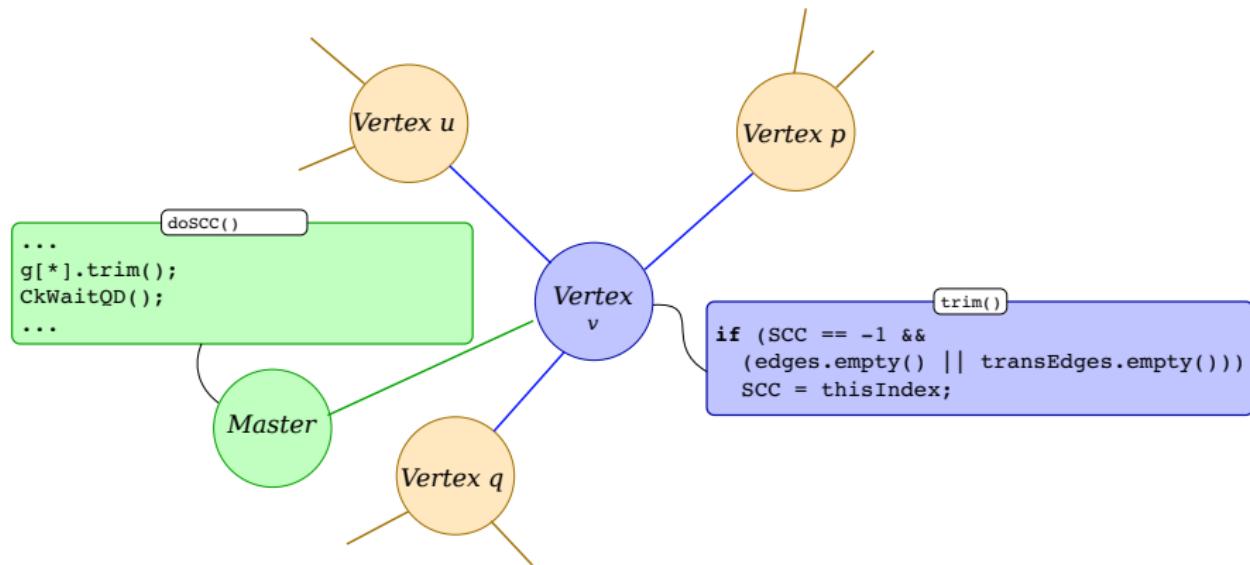
1: Построение транспонированного графа



# Поиск сильно связных компонент (SCC) (2/4)

Реализация алгоритма на основе раскраски графа на Charm++

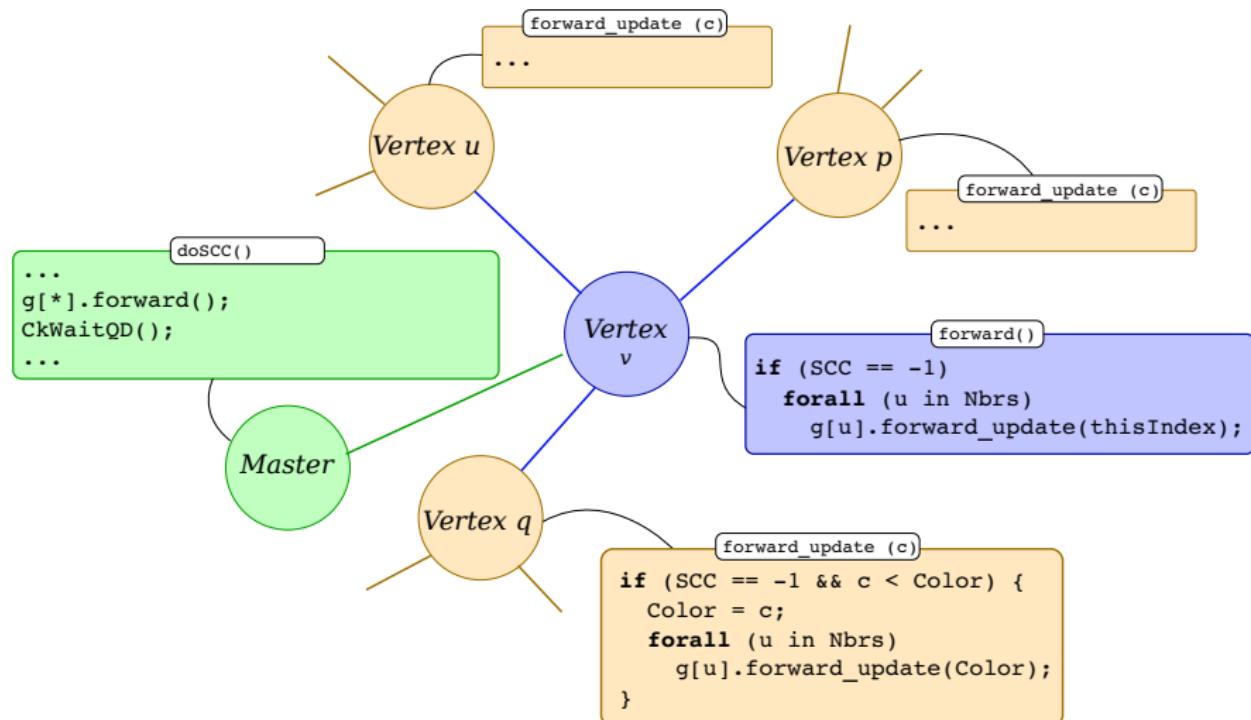
2: Поиск “висящих” вершин



# Поиск сильно связных компонент (SCC) (3/4)

Реализация алгоритма на основе раскраски графа на Charm++

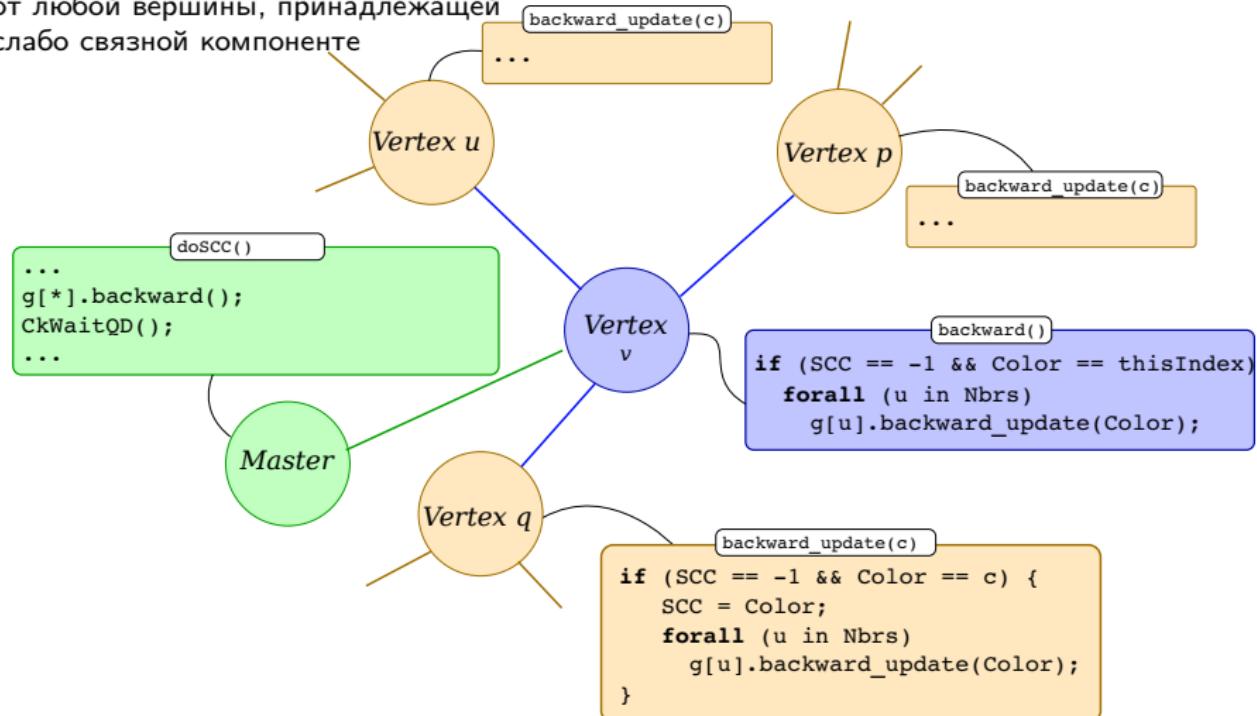
## 3: Поиск слабо связных компонент



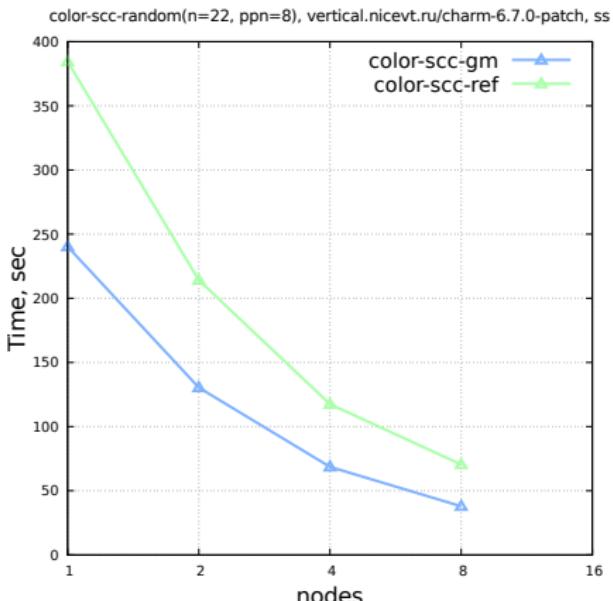
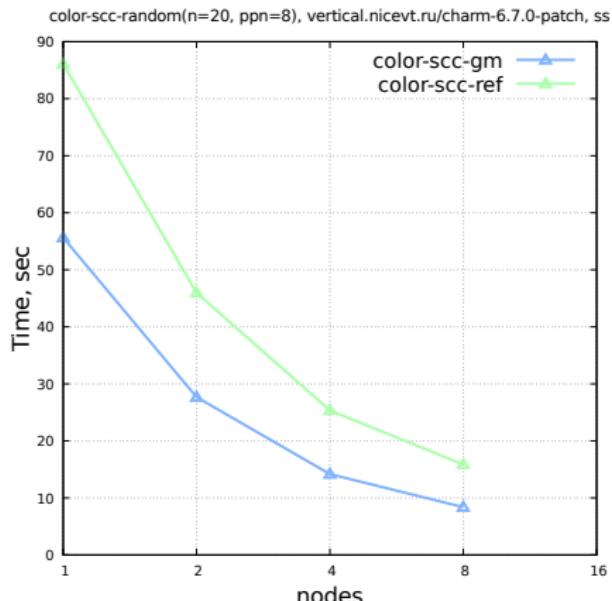
# Поиск сильно связных компонент (SCC) (4/4)

Реализация алгоритма на основе раскраски графа на Charm++

4: Поиск обратно-достижимых вершин  
от любой вершины, принадлежащей  
слабо связной компоненте

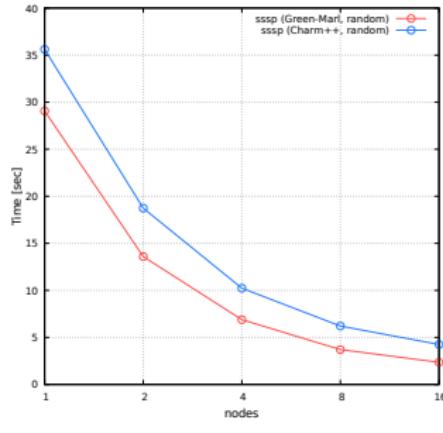


# Оценка эффективности полученного кода scc

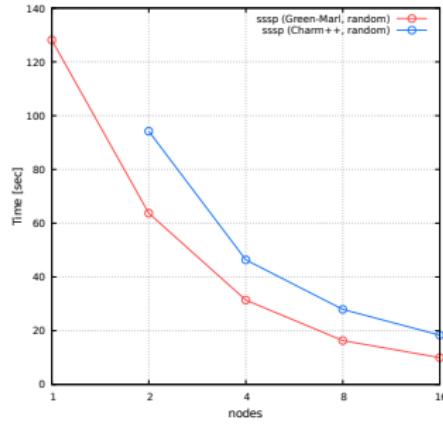


# Оценка эффективности полученного кода SSSP

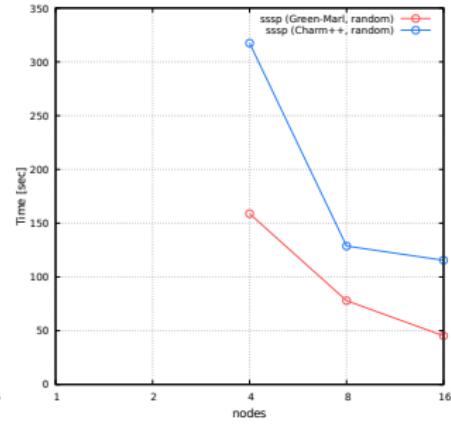
sssp-random( $n=20$ ,  $ppn=8$ ), vertical.nicevt.ru/charm-6.7.0-patch, ss



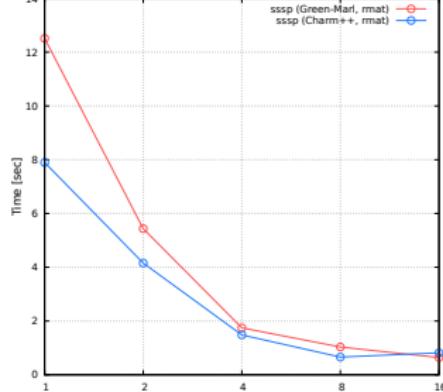
sssp-random( $n=22$ ,  $ppn=8$ ), vertical.nicevt.ru/charm-6.7.0-patch, ss



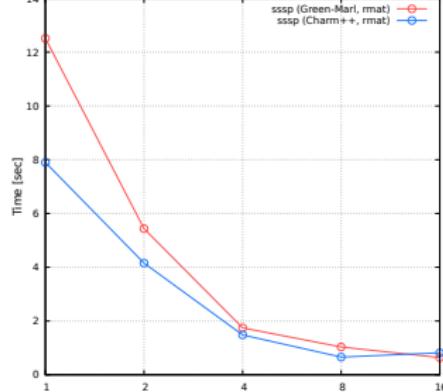
sssp-random( $n=24$ ,  $ppn=8$ ), vertical.nicevt.ru/charm-6.7.0-patch, ss



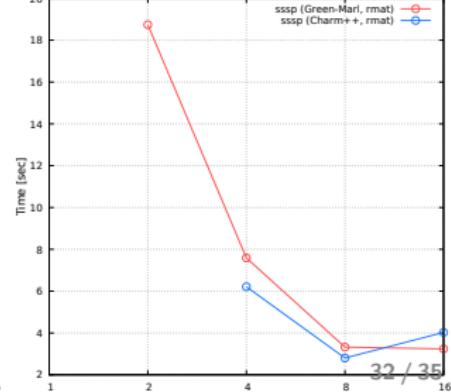
sssp-rmat( $n=22$ ,  $ppn=8$ ), vertical.nicevt.ru/charm-6.7.0-patch, ss



sssp-rmat( $n=22$ ,  $ppn=8$ ), vertical.nicevt.ru/charm-6.7.0-patch, ss



sssp-rmat( $n=24$ ,  $ppn=8$ ), vertical.nicevt.ru/charm-6.7.0-patch, ss



## Выводы и планы на будущее

- Выводы:
  - Разработана прототипная реализация генератора Charm++ кода в компиляторе проблемно-ориентированного языка программирования Green-Marl.
  - Предварительные результаты тестирования показывают, что эффективность генерируемого кода сопоставима с эффективностью “ручных” реализаций.
- Планы на будущее:
  - Добавить поддержку оставшихся возможностей языка Green-Marl (например, встроенную функцию BFS обхода графа)
  - Добавить поддержку библиотеки TRAM в генераторе Charm++
  - Провести оценочное тестирование на различных тестовых задачах
- Благодарности
  - Работа выполнена при поддержке Российского Фонда Фундаментальных Исследований (РФФИ), грант 15-07-09368.

Спасибо! Вопросы?



- Семинар “Параллельная обработка больших графов” (в рамках RuSCDays'2017)
- Тематика: приложения, алгоритмы, технологии, архитектура
- Подача статей – 15 мая 2017
  - <http://russianscdays.org/workshop/GraphWorkshop>