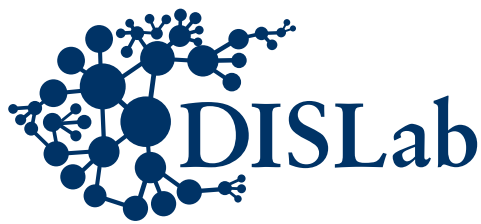
Early Experience with Intergrating Charm++ Support to Green-Marl DSL

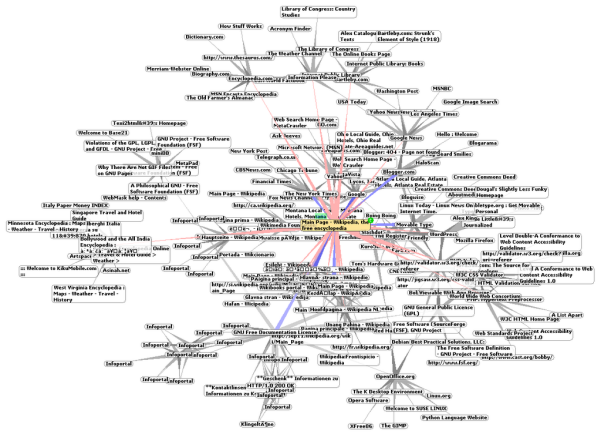
Alexander Frolov

DISLab,
«Scientific and Research Center on Computer Techonology» (NICEVT)



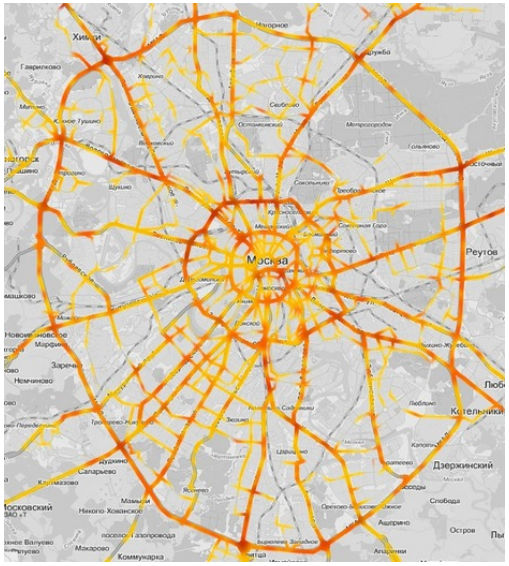
15th Annual Workshop on Charm++ and its Applications
Urbana-Champaign/Moscow (webcast), April 18, 2017

Large-scale Graphs in Real World

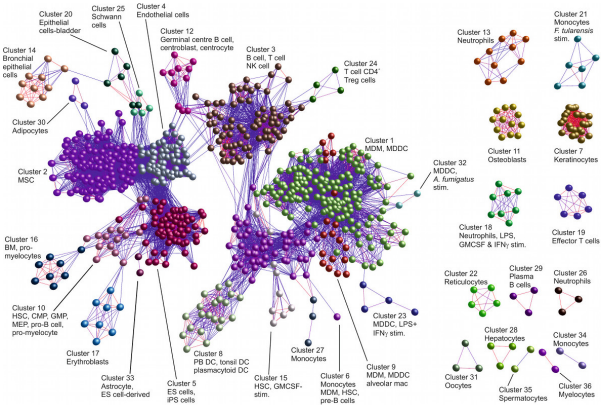


WEB-graph analysis

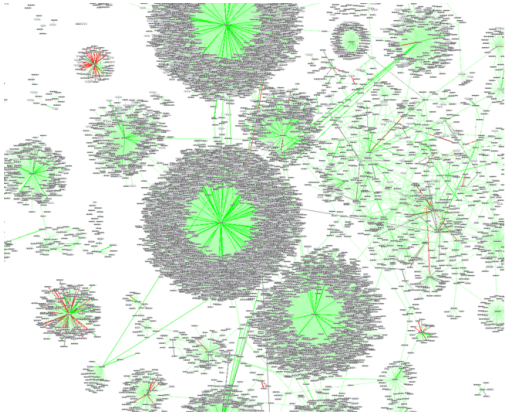
Social Network Analysis



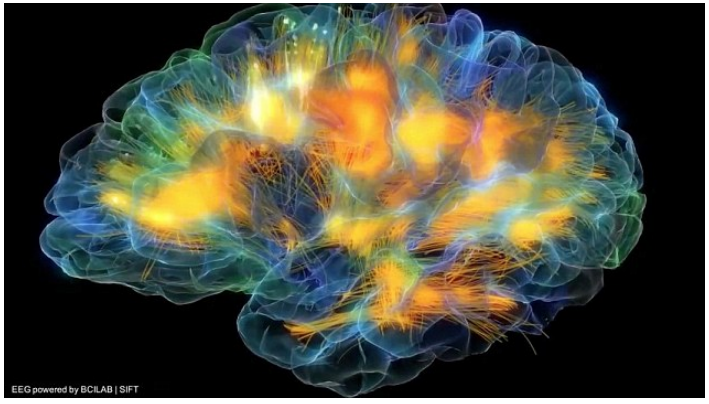
Road Networks Analysis



Bioinformatics



Cybersecurity



Human Brain Project

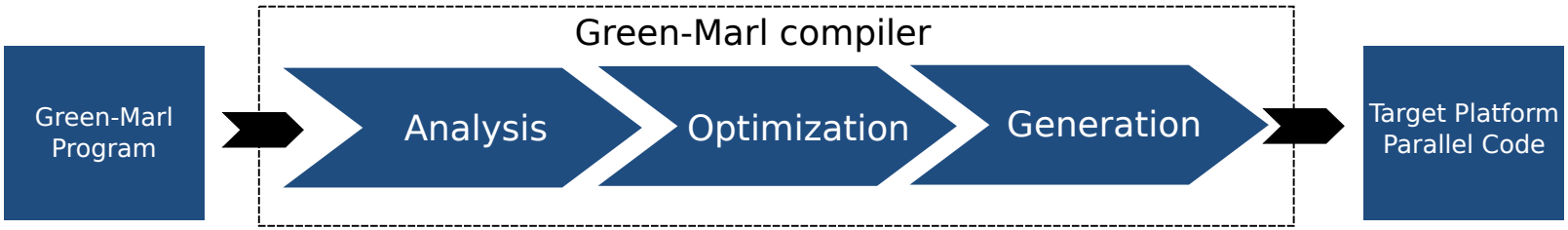
Large-scale Graph Applications: Productivity Issue

- Common challenges of parallel programming
  - efficient parallel algorithm design is difficult (axiom)
  - target system architecture dependency
- Graph specific challenges
  - short message aggregation
  - static graph distribution
  - dynamic load balancing
- No standard parallel graph library up to day!
  - Boost Parallel Graph Library (only if you C++ expert! or want to be)
  - GraphBLAS (yet still in newborn baby stage)
- Assessment of relative programming effort (in #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 – domain-specific language (DSL) for designing imperative graph analysis algorithms
- Developed in PPL @ Stanford Univeristy
  - DSL spec & GM Compiler with C++/OpenMP backend [ASPLOS 2012] [1]
  - Pregel (GPS, Giraph) backend [FOSDEM 2013] [2]
  - https://github.com/stanford-ppl/Green-Marl
- Included to PGX.D (Orable Labs)
  - PGX.D backend [SC15] [3]



[1] 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.

[2] 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. – C. 208.

[3] 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.

Green-Marl by Example

Query: How cool is your daddy? (c)

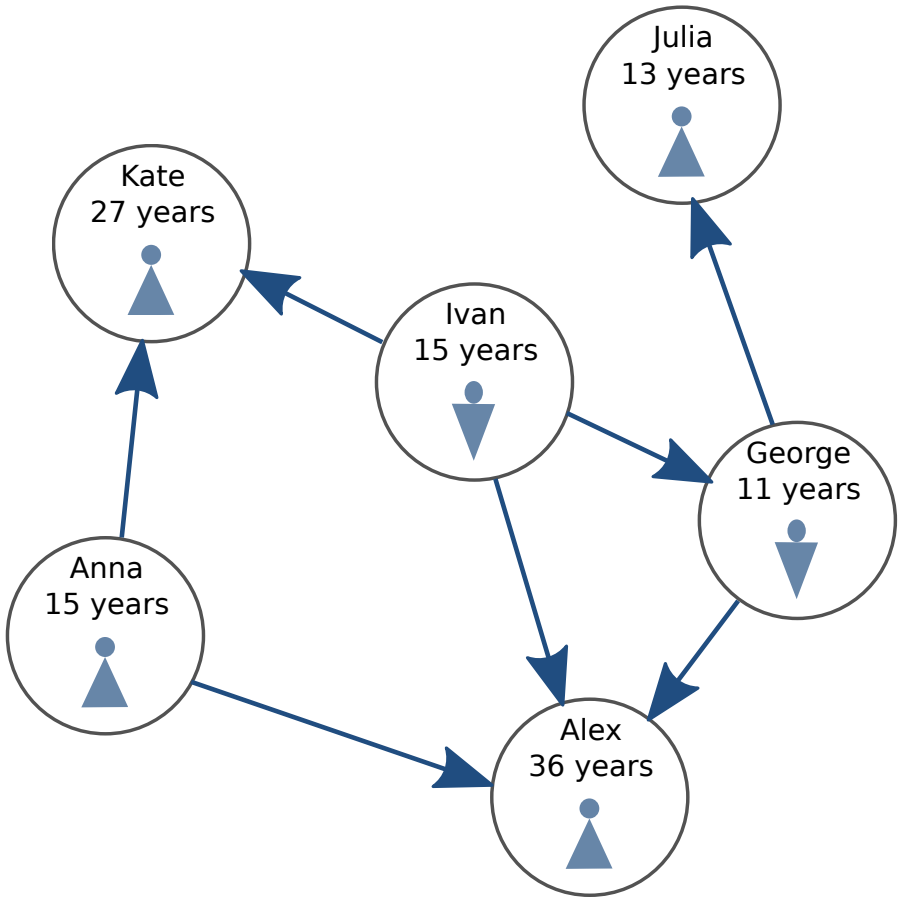
Social networks:

Count the average number of followers from 10 to 20 years old for users with age greater than 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



Graph Algorithms Implemented in Green-Marl [4]

- Closeness Centrality and variants
- Degree Centrality and variants
- Degree Distribution and variants
- Diameter
- Dijkstra's Algorithm and variants
- Bidirectional Dijkstra's Algorithm (and variants)
- Eigenvector Centrality
- Fattest-Path
- Filtered BFS
- Hyperlink-Induced Topic Search
- K-Core
- Matrix Factorization (Gradient Descent)
- PageRank and variants
- SALSA and variants
- Radius
- Random Walk with Restart
- SSSP (Bellman Ford) and variants
- SSSP (Hop Distance) and variants
- Strongly Connected Components (Kosaraju)
- Strongly Connected Components (Tarjan)
- Triangle Counting
- Vertex Betweenness Centrality and variants
- Weakly Connected Components

[4] **PGX.D Project, Oracle Labs,** https://docs.oracle.com/cd/E56133_01/2.2.1/reference/algorithms/index.html

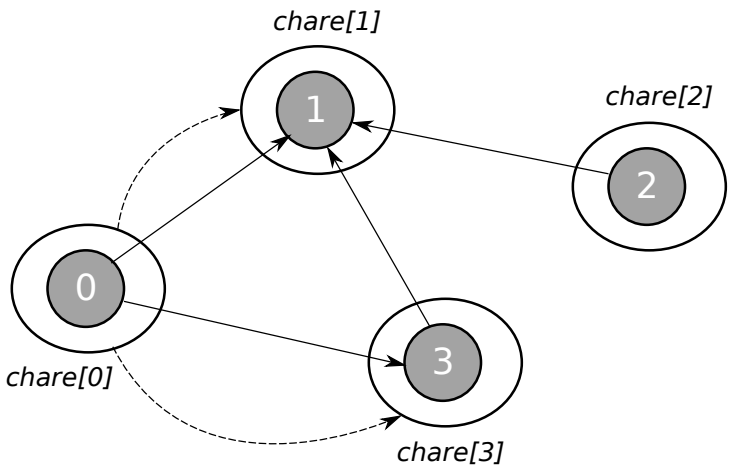
Why Green-Marl @ Charm++ is not a bad idea

- No publicly available Green-Marl backend for HPC clusters
- Charm++ is a mature framework for parallel programming with active community
- Charm++ shows nice scalability on a large number of nodes
- Charm++ asynchronous message-driven execution model is perfect for expressing vertex-centric parallel graph algorithms
- Charm++ supports dynamic load balancing
- Open-source Green-Marl compiler has support for Pregel-like backends (Giraph, Stanford GPS) which makes porting to Charm++ much easier

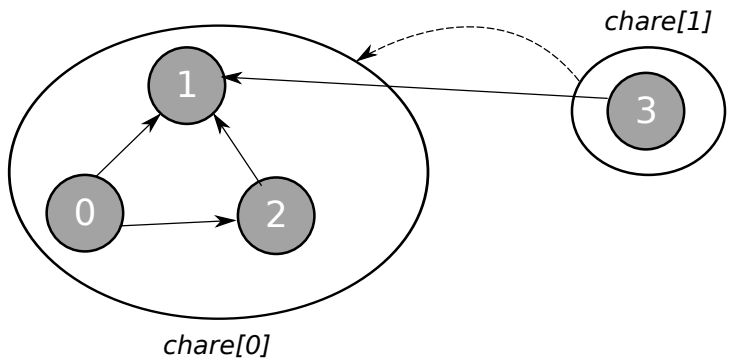
Approaches to Large-scale Graph Processing on Charm++

Vertex-centric [= Fine-grained] vs Subgraph-centric [= Coarse/Medium-grained]

- Vertex-centric
  - Graph (G) – array of chares distributed across parallel processes (PE)
  - Vertex – chare (1:1)
  - Vertices communicate via asynchronous active messages (entry method calls)
  - Program completion detected by CkStartQD



- Subgraph-centric
  - Graph (G) – array of chares distributed between parallel processes (PE).
  - Vertex – chare (n:1), any local representation possible
  - Algorithms consist of local (sequential) and global parts (parallel, Charm++).
  - Application level optimizations (aggregation, local reductions, etc.)
  - Program completion detected by CkStartQD or manually



Green-Marl Translation to Asynchronous Message-driven Models

- The main challenge is a gap between imperative shared memory Green-Marl and async. object-based data-driven Pregel and Charm++

Green-Marl

Data-level Parallel (PRAM) Domain-specific Language

```
Forall (n in G.Nodes) {
   Forall (v in n.Nbrs) {
     ...
   }
}
...
Forall (n in G.Nodes) {
    tot += n.A;
}
```

Google Pregel

Vertex-centric, Bulk-Synchronous Parallel Framework

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

Charm++

Asynchronous Message-driven Parallel Programming Language

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

Green-Marl @ Pregel [5]

Green-Marl compiler:

- Build the Finite State Machine (FSM) with master/slave control flow.
- Apply transformations & optimizations to the IR (AST+FSM)

Green-Marl Program

```
Procedure foo(g:Graph,...) {
  Bool fin = False;
  While (!fin) {
     Foreach(n: g.Nodes) {
       ...
     }
  }
}
```

Pregel Program

```
class Master : ... {
  Bool fin;
  ...
  void compute() {
    switch(current_state) {
      case 0: do_state_0(); break;
      case 1: do_state_1(); break;
      case 2: do_state_2(); break;
      ...
    }
  }
  void do_state_0() {...}
  void do_state_1() {...}
  void do_state_2() {...}
  ...
}

class Vertex : ... {
  Bool fin;
  ...
  void compute() {
    switch(current_state) {
      case  1: do_state_1(); break;
       ...
    }
  }
 void  do_state_1() {...}
 ...
}
```

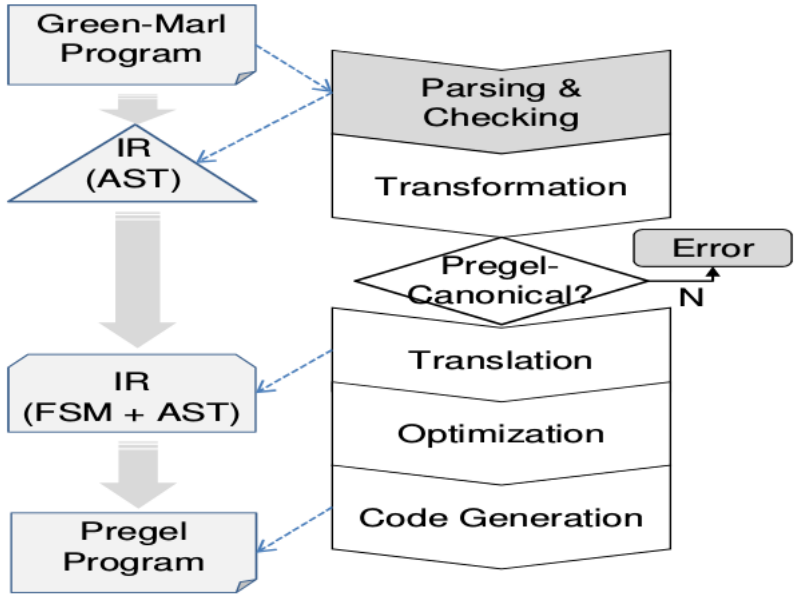
[5] Hong S. et al. Simplifying scalable graph processing with a domain-specific language //Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization. – ACM, 2014. – C. 208.

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.

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



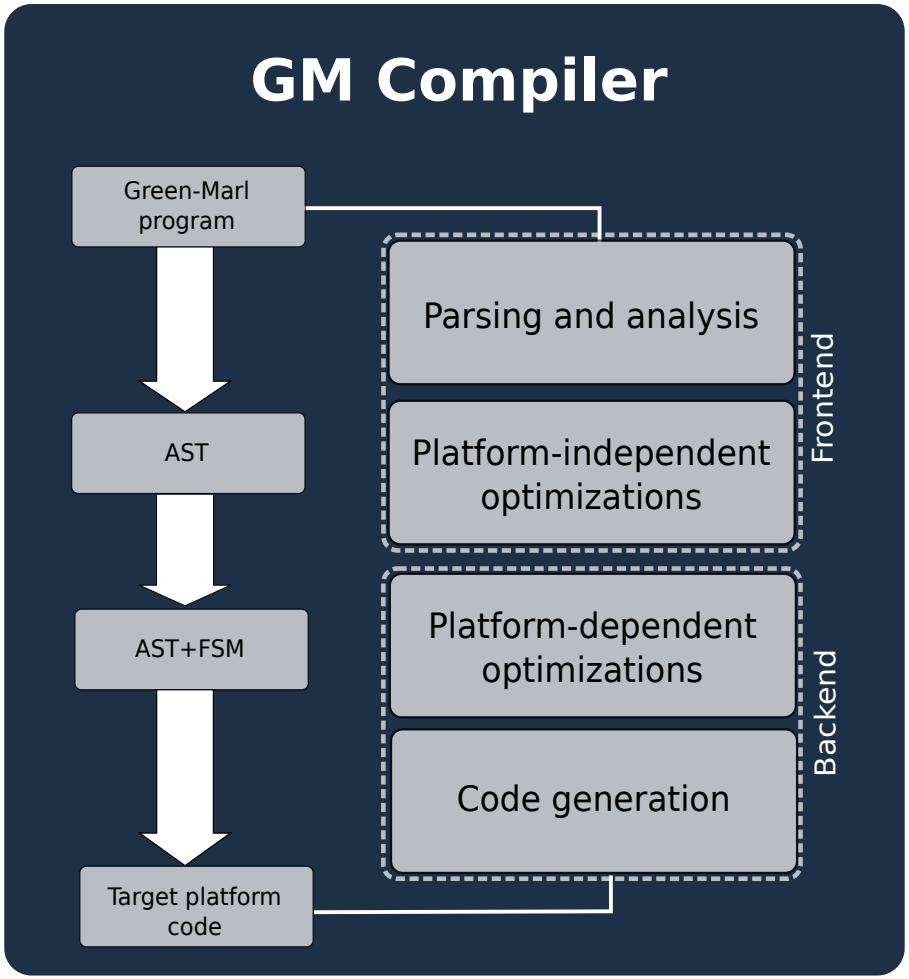
Green-Marl compiler stages:

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

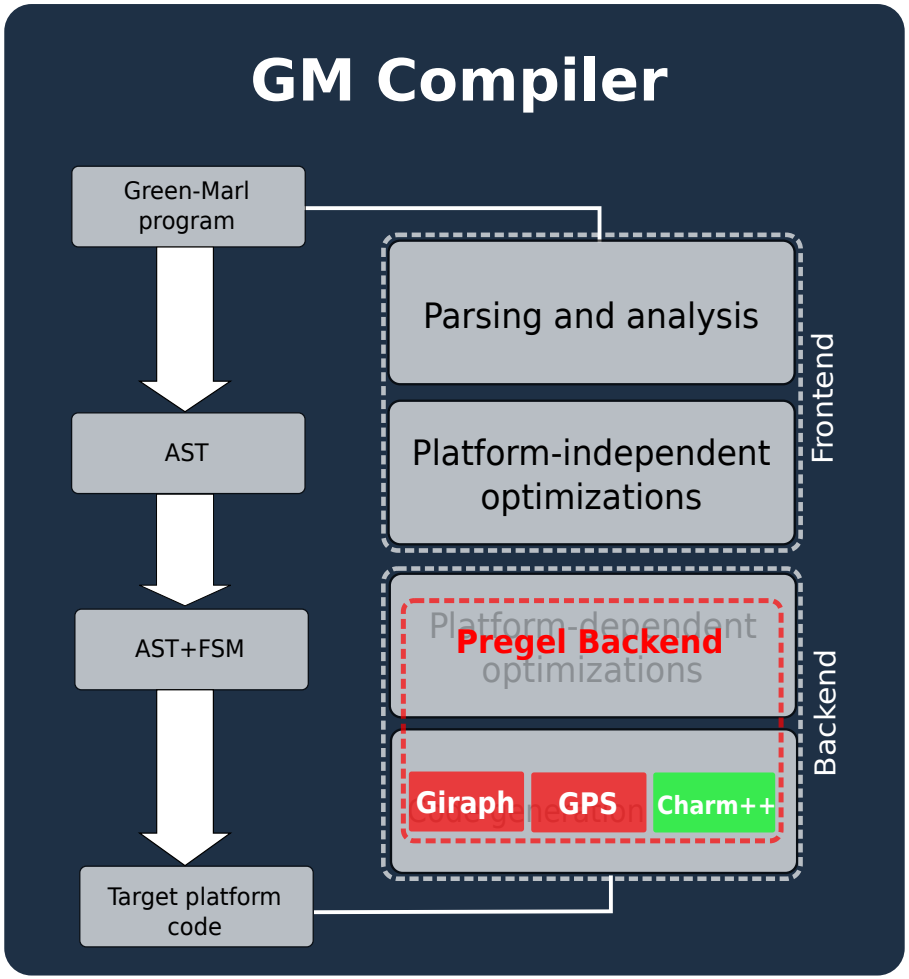
Charm++ vs. Pregel

	Charm++	Pregel
Computation model	Asynchronous, message driven	Step-based, Bulk-Synchronous Parallel
Master/Slave model	No	Yes (Giraph, GPS)
Vertex Impl.	Chares	Pregel Objects
Edges Impl.	Any container	
Vertex Distribution	Static (1D,2D,...,6D block distribution)	Static (RTS)
Vertex Migration	Yes	No
Remote computation	entry methods	compute method
Shared memory	No	No
Aggregation	Yes (TRAM)	Yes
Global variables	Readonly	Yes
Reduction	Yes	Yes
Termination	Automatic (QD)	Semi-Automatic (VoteToHalt)
Usage	General-purpose	Graph applications

Green-Marl Compiler



Green-Marl Compiler



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 compiler stages:

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

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 @ 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;                                          S0 (SEQ)
  Foreach (n : G.Nodes)
  {
    n.__S1prop = 0;                                   S1 (PAR)
  }
  Foreach (t : G.Nodes)
    If (((t.age >= 10)  && (t.age < 20) )  )
    {
      Foreach (n : t.Nbrs)
      {
        n.__S1prop += 1 @ t ;                         S2 (PAR)
      }
    }
  Foreach (n : G.Nodes)
  {
    n.teen_cnt = n.__S1prop;
    If ((n.age > K) )
    {
      __S2 += n.teen_cnt @ n ;
      _cnt3 += 1 @ n ;                                S3 (PAR)
    }
  }
  _avg4 = (0 == _cnt3)  ?
    0.000000 : (__S2 /  (Double ) _cnt3)  ;
  avg =  (Float ) _avg4;                              S4 (SEQ)
  Return avg;
}
```

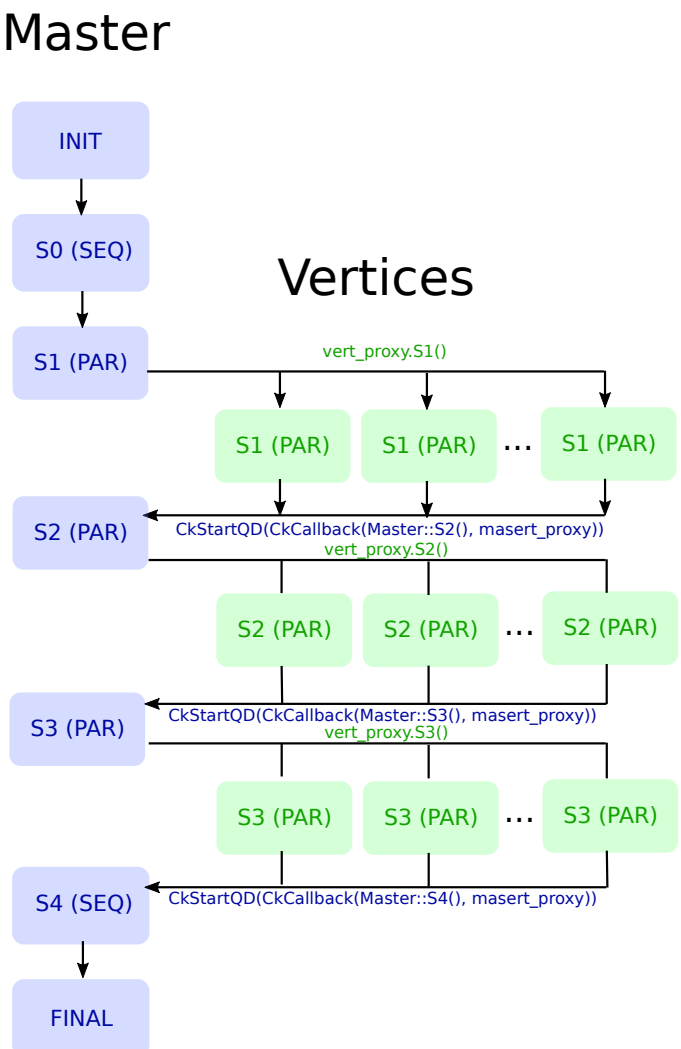
Green-Marl @ Charm++

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;                                          S0 (SEQ)
  Foreach (n : G.Nodes)
  {
    n.__S1prop = 0;                                   S1 (PAR)
  }
  Foreach (t : G.Nodes)
    If (((t.age >= 10)  && (t.age < 20) )  )
    {
      Foreach (n : t.Nbrs)
      {                                               S2 (PAR)
        n.__S1prop += 1 @ t ;
      }
    }
  Foreach (n : G.Nodes)
  {
    n.teen_cnt = n.__S1prop;
    If ((n.age > K) )
    {                                                 S3 (PAR)
      __S2 += n.teen_cnt @ n ;
      _cnt3 += 1 @ n ;
    }
  }
  _avg4 = (0 == _cnt3)  ?
    0.000000 : (__S2 /  (Double ) _cnt3)  ;          S4 (SEQ)
  avg =  (Float ) _avg4;
  Return avg;
}
```

State Machine



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;                                          S0 (SEQ)
  _cnt3 = 0;
  Foreach (n : G.Nodes)
  {                                                  S1 (PAR)
    n.__S1prop = 0;
  }
  Foreach (t : G.Nodes)
    If (((t.age >= 10)  && (t.age < 20) ) )
    {
      Foreach (n : t.Nbrs)                           S2 (PAR)
      {
        n.__S1prop += 1 @ t ;
      }
    }
  Foreach (n : G.Nodes)
  {
    n.teen_cnt = n.__S1prop;                         S3 (PAR)
    If ((n.age > K) )
    {
      __S2 += n.teen_cnt @ n ;
      _cnt3 += 1 @ n ;
    }
  }
  _avg4 = (0 == _cnt3)  ?
    0.000000 : (__S2 /  (Double ) _cnt3) ;           S4 (SEQ)
  avg =  (Float ) _avg4;
  Return avg;
}
```

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;                                          S0 (SEQ)
  Foreach (n : G.Nodes)
  {                                                   S1 (PAR)
    n.__S1prop = 0;
  }
  Foreach (t : G.Nodes)
    If (((t.age >= 10)  && (t.age < 20) )  )
    {
      Foreach (n : t.Nbrs)                            S2 (PAR)
      {
        n.__S1prop += 1 @ t ;
      }
    }
  Foreach (n : G.Nodes)
  {
    n.teen_cnt = n.__S1prop;
    If ((n.age > K) )
    {                                                 S3 (PAR)
      __S2 += n.teen_cnt @ n ;
      _cnt3 += 1 @ n ;
    }
  }
  _avg4 = (0 == _cnt3)  ?
    0.000000 : (__S2 /  (Double ) _cnt3)  ;          S4 (SEQ)
  avg =  (Float ) _avg4;
  Return avg;
}
```

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 :
  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;                                          S0 (SEQ)
  Foreach (n : G.Nodes)
  {                                                   S1 (PAR)
    n.__S1prop = 0;
  }
  Foreach (t : G.Nodes)
    If (((t.age >= 10)  && (t.age < 20) )  )
    {
      Foreach (n : t.Nbrs)                            S2 (PAR)
      {
        n.__S1prop += 1 @ t ;
      }
    }
  Foreach (n : G.Nodes)
  {
    n.teen_cnt = n.__S1prop;
    If ((n.age > K) )                                 S3 (PAR)
    {
      __S2 += n.teen_cnt @ n ;
      _cnt3 += 1 @ n ;
    }
  }
  _avg4 = (0 == _cnt3)  ?
    0.000000 : (__S2 /  (Double ) _cnt3)  ;          S4 (SEQ)
  avg =  (Float ) _avg4;
  Return avg;
}
```

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;                                          S0 (SEQ)
  Foreach (n : G.Nodes)
  {                                                   S1 (PAR)
    n.__S1prop = 0;
  }
  Foreach (t : G.Nodes)
    If (((t.age >= 10)  && (t.age < 20) )  )
    {
      Foreach (n : t.Nbrs)                            S2 (PAR)
      {
        n.__S1prop += 1 @ t ;
      }
    }
  Foreach (n : G.Nodes)
  {
    n.teen_cnt = n.__S1prop;                          S3 (PAR)
    If ((n.age > K) )
    {
      __S2 += n.teen_cnt @ n ;
      _cnt3 += 1 @ n ;
    }
  }
  _avg4 = (0 == _cnt3)  ?
    0.000000 : (__S2 /  (Double ) _cnt3)  ;           S4 (SEQ)
  avg =  (Float ) _avg4;
  Return avg;
}
```

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;                                          S0 (SEQ)
  Foreach (n : G.Nodes)
  {                                                   S1 (PAR)
    n.__S1prop = 0;
  }
  Foreach (t : G.Nodes)
    If (((t.age >= 10)  && (t.age < 20) )  )
    {
      Foreach (n : t.Nbrs)                            S2 (PAR)
      {
        n.__S1prop += 1 @ t ;
      }
    }
  Foreach (n : G.Nodes)
  {
    n.teen_cnt = n.__S1prop;
    If ((n.age > K) )                                 S3 (PAR)
    {
      __S2 += n.teen_cnt @ n ;
      _cnt3 += 1 @ n ;
    }
  }
  _avg4 = (0 == _cnt3)  ?
    0.000000 : (__S2 /  (Double ) _cnt3)  ;          S4 (SEQ)
  avg =  (Float ) _avg4;
  Return avg;
}
```

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();
  }
};
```

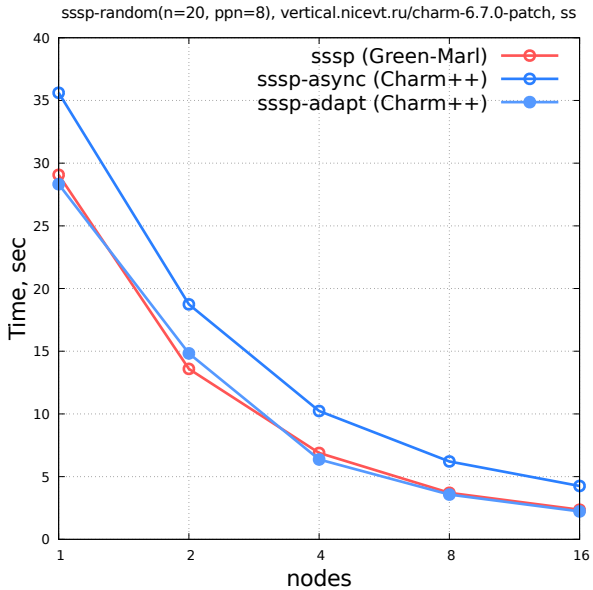
Callback to boilerplate code

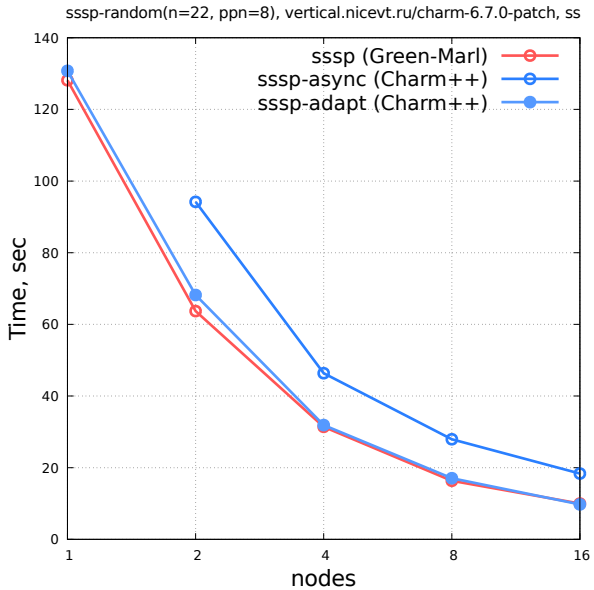
Performance evaluation

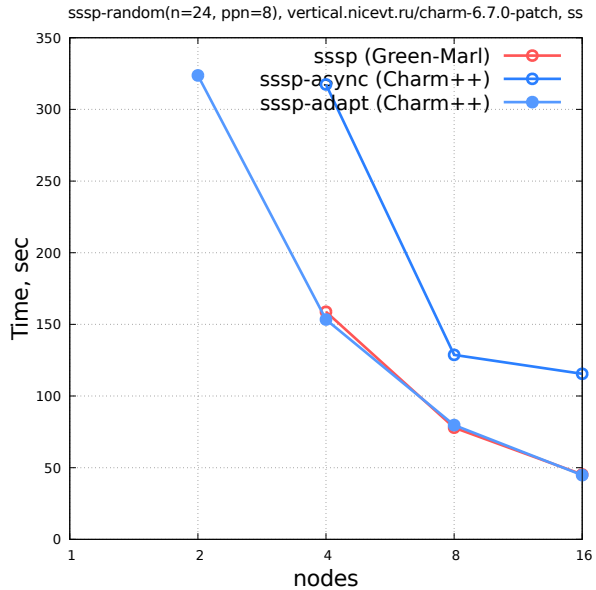
- **Benchmarks**
  - Single-Source Shortest Path (SSSP)
  - Connected Components (CC)
  - PageRank
  - Strongly Components Components (SCC)
- **Graphs**
  - RMAT (Graph500)
  - Random
- **System**
  - 36-node NICEVT HPC cluster
  - 2x Intel Xeon E6-2630, 2.3GHz/64GB

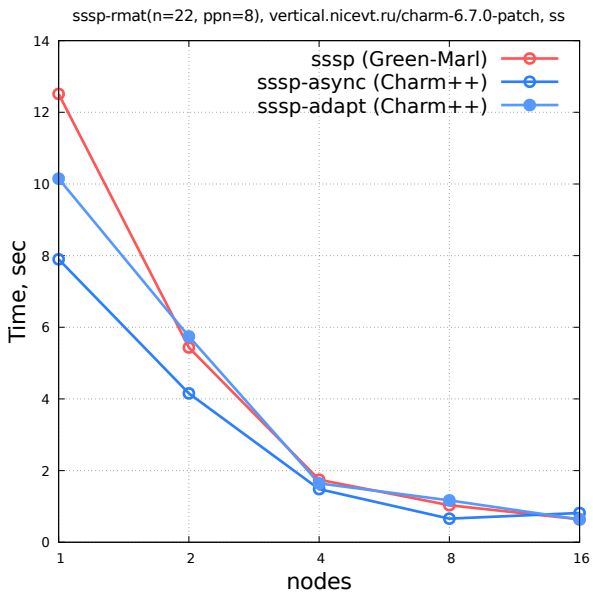
Performance evaluation

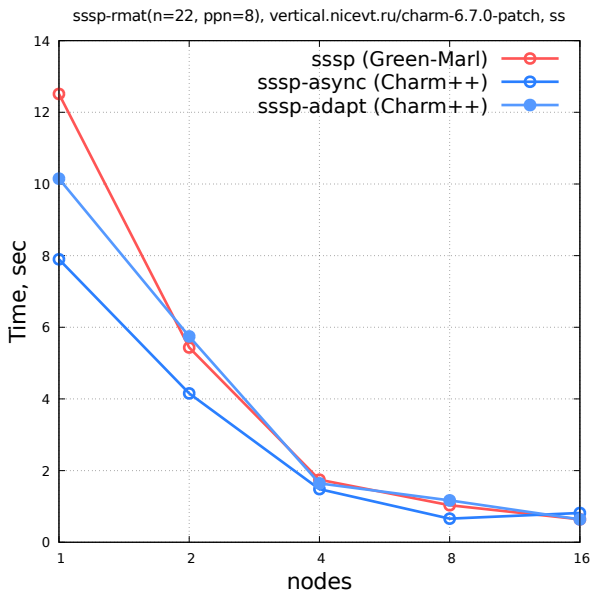
SSSP, RMAT/Random, scale=20,22,24, PPN=8, NICEVT HPC cluster (2x Intel Xeon E6-2630, 2.3GHz/64GB)

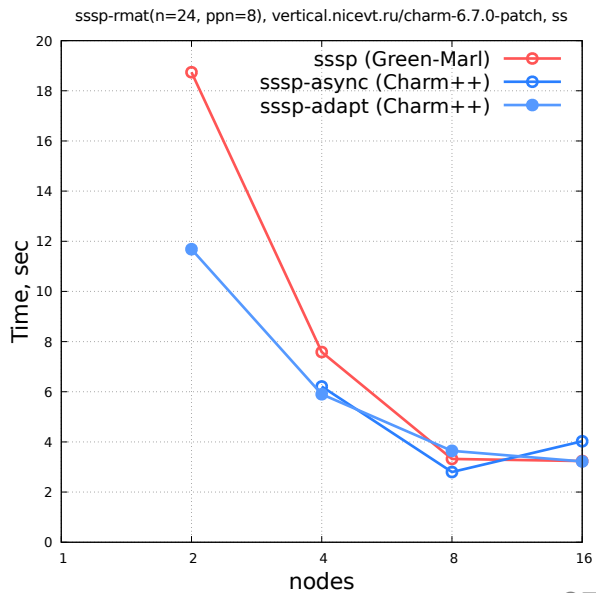






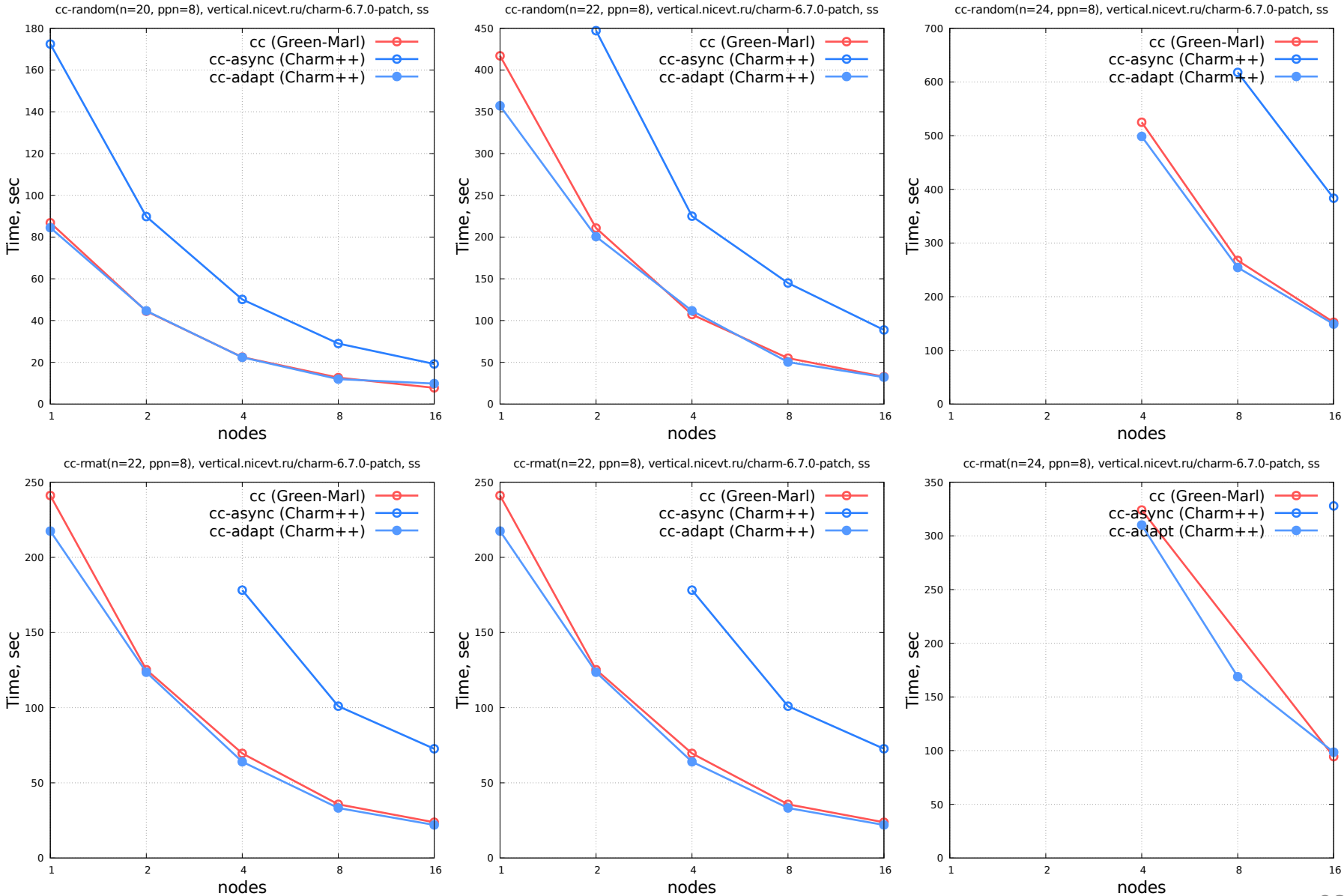






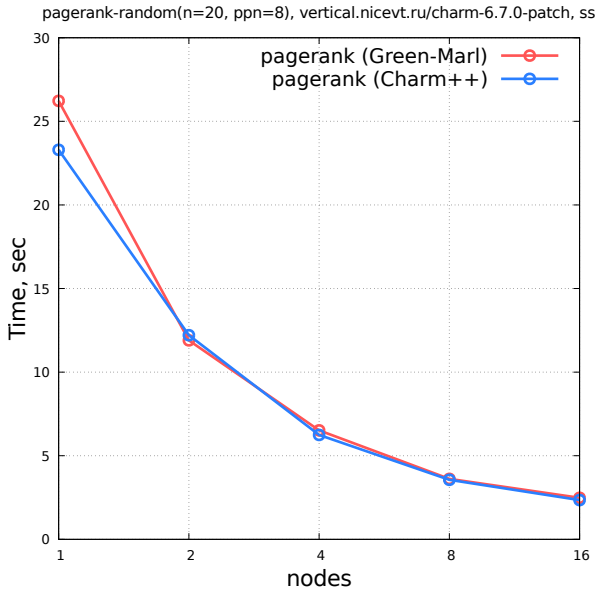
Performance evaluation

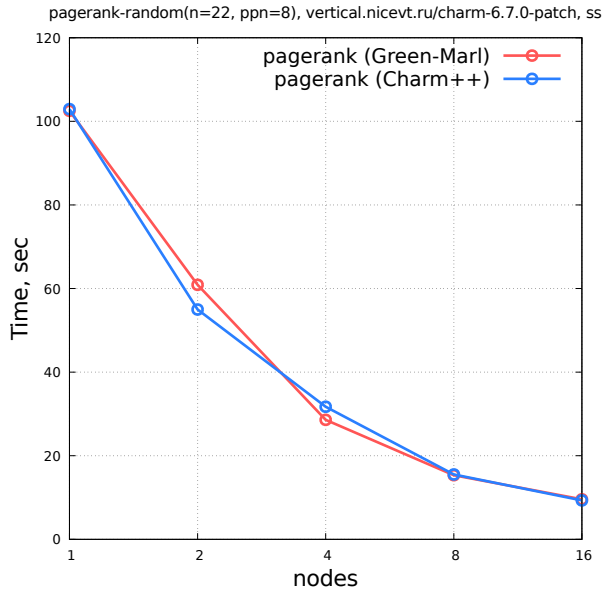
CC, RMAT/Random, scale=20,22,24, PPN=8, NICEVT HPC cluster (2x Intel Xeon E6-2630, 2.3GHz/64GB)

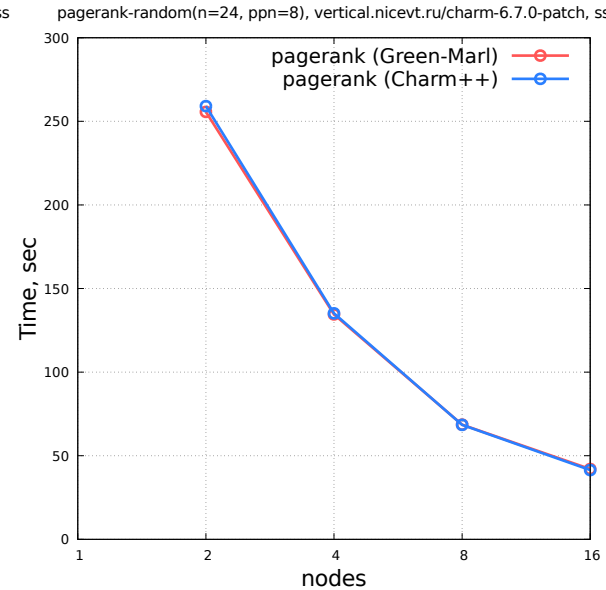


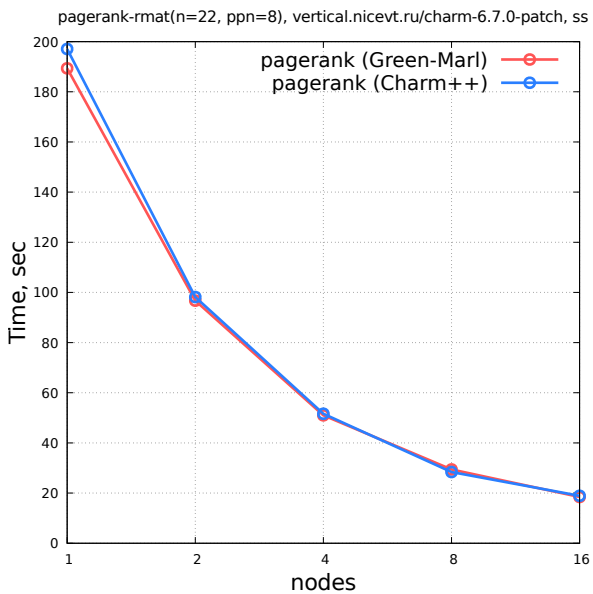
Performance evaluation

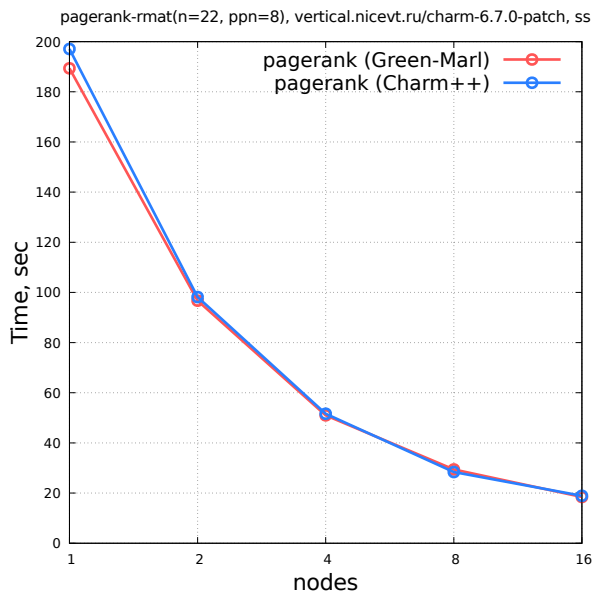
PageRank, RMAT/Random, scale=20,22,24, PPN=8, NICEVT HPC cluster (2x Intel Xeon E6-2630, 2.3GHz/64GB)

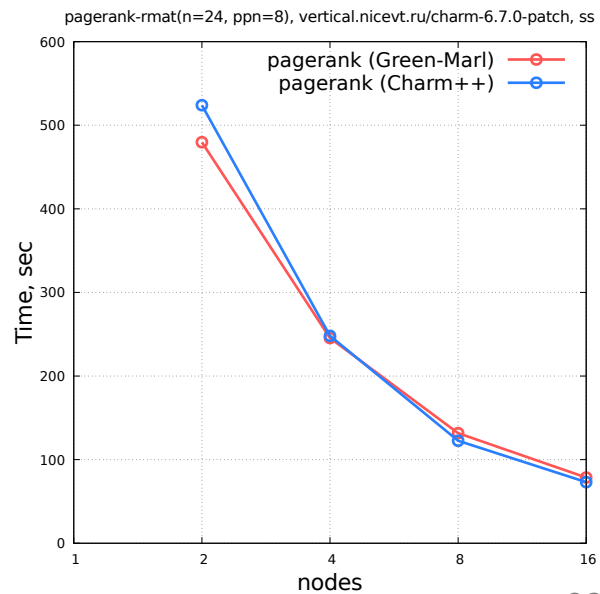






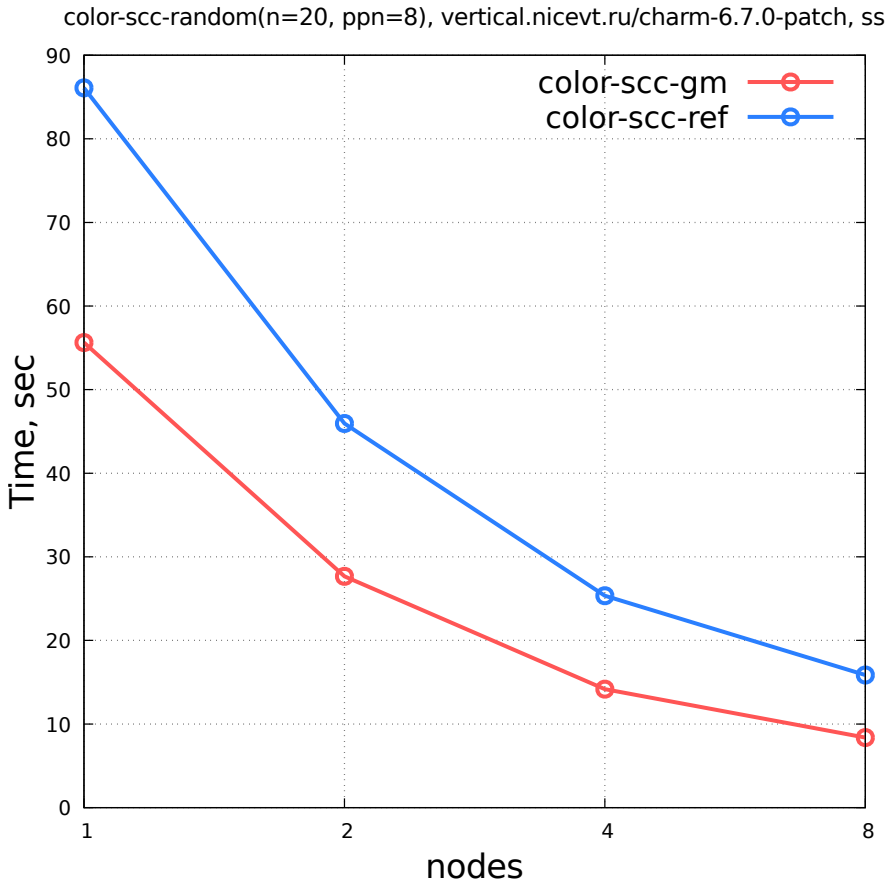


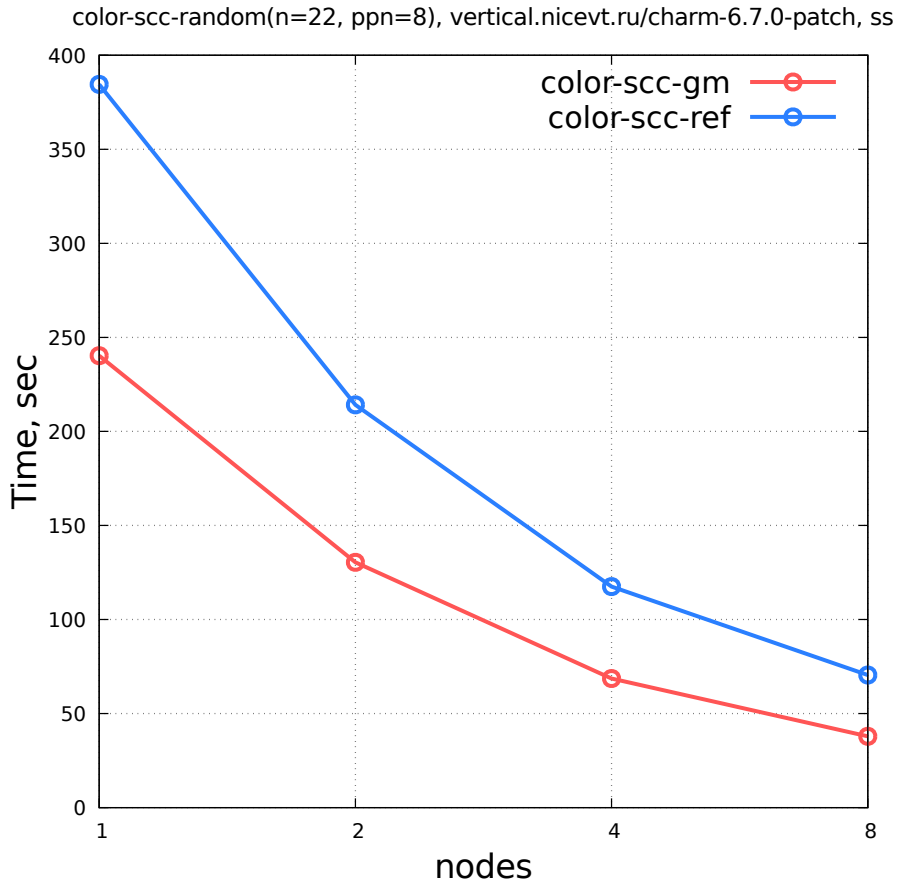




Performance evaluation

SCC, Random, scale=20, PPN=8, NICEVT HPC cluster (2x Intel Xeon E6-2630, 2.3GHz/64GB)





Conclusions & Future Plans

- Conclusions:
  - A **proof-of-concept** implementation of Charm++ backend for Green-Marl has been developed.
  - Early evaluation of the Charm++ backend showed comparable performance to hand-written tests, and suprisingly generated code appeared to be more effective than hand-written naïve implementations.
  - https://github.com/alexfrolov/Green-Marl
- Future Plans:
  - Add support for other features of Green-Marl (like build-in BFS traverse)
  - Add TRAM support to Charm++ backend
  - Large-scale peformance evaluation
- Acknowledgements
  - This work is partially supported by Russian Foundation for Basic Research (RFBR) under Contract 15-07-09368.

Thank you! Questions?