






# DSL pour la Fouille des Réseaux Sociaux sur des Plates-formes Multi-cœurs

État d'avancement,

1ère année de thèse (cotutelle)

**MESSI NGUELE Thomas**

Sous la co-direction de:

**Pr. Jean-François MEHAUT**
UJF/CEA

**Pr. Maurice TCHUENTE**
UY1/LIRIMA-UMMISCO

Grenoble, 15 juin 2015

## Cursus académique

- **2014 : Doctorant à UY1 (idasco) et à UJF (corse)**
- 2013 : Master 2, mention TB, vice-major de promotion
- 2011 : Master 1, mention B, major de promotion
- 2010 : Licence, mention AB, major de promotion
- 2007 : Bac. C, mention AB

## Publications (master)

- Dec. 2013 : Proceedings, CRI'2013 Yaoundé [CFNM13a]
- Nov. 2013 : Proceedings, IA3 Workshop Denver [CFNM13b]

# Contexte et problème

## Réseau social

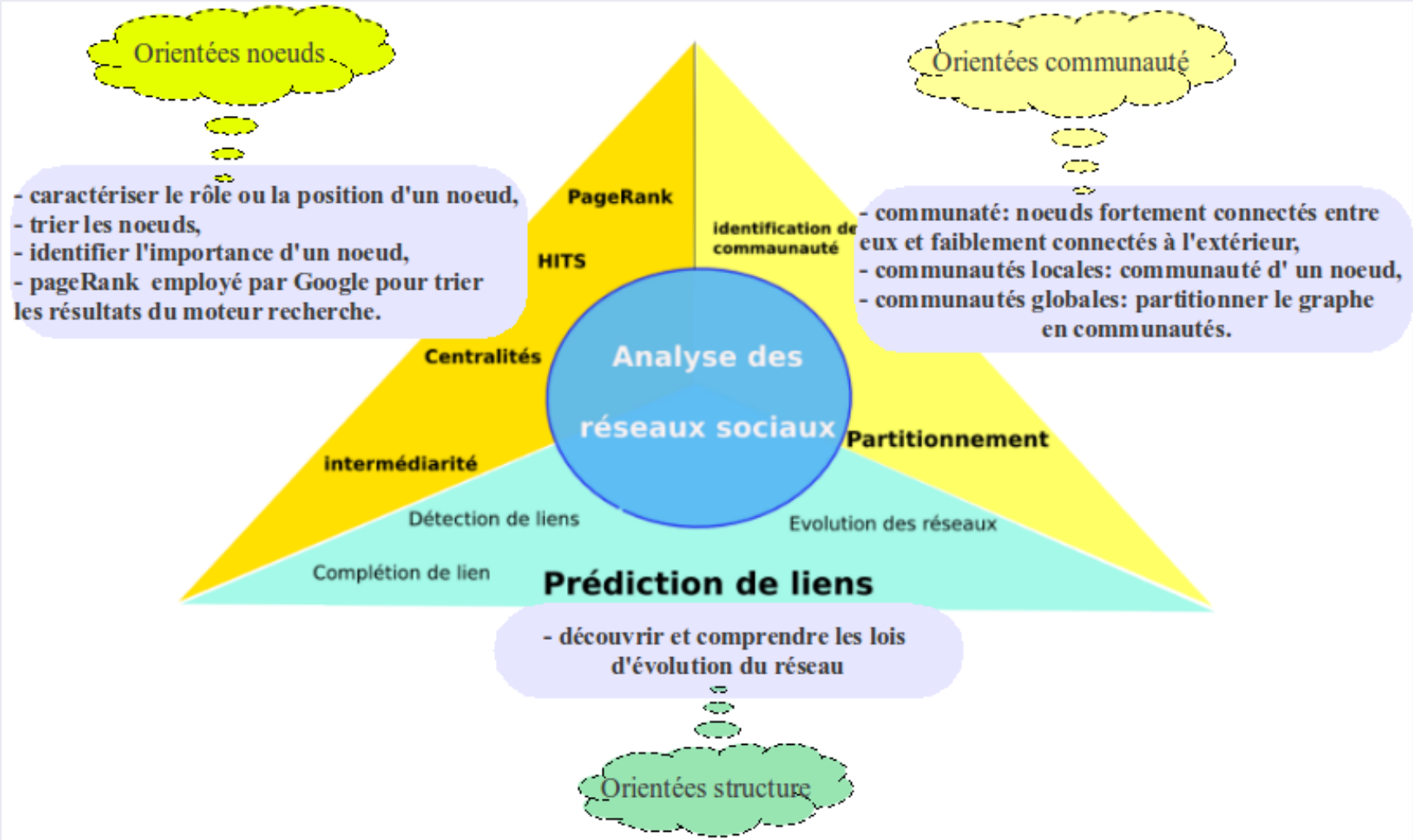
- ensemble d'entités (individus) interconnectées par des liens (amitié).
- modélisation à l'aide des **graphes**
  - **très grand nombre de noeuds**, facebook $1.44*10^9$, Twitter $308*10^6$
  - **faible densité de liens**, un noeud est lié en moyenne à 150 autres noeuds.
- génère une grande quantité de données (nécess. un grand volume de calcul).

## Réseau de chercheurs physique quantique (1060 nœuds, 1044 arêtes)

- arêtes possibles : **561270**, soit une densité de **0.2%** ;
- katz [Kat53] ou rules [KTV13] tourne sur **560226** soit **99.8%**.
- **Plusieurs heures d'exécution en mode séquentiel**(prévision de liens).

- **Question** : Comment réduire le temps d'exécution ?
- **Réponse** : **Usage du parallélisme**

# Exemples d'analyse des réseaux sociaux



# Fouille parallèle des réseaux sociaux

## Usage des modèles de programmation parallèle existant

- Description : à partir d'un algo séquentiel,
  1. Version 1 : Prototypage avec python ou R (pour validation)
  2. Version 2 : Implémentation séquentielle avec C ou C++ (pour les perf.)
  3. Version 3 : Parallélisation puis implémentation avec
     - OpenMP : détection des communautés sur multi-cœur [RBM12]
     - CUDA : détection des communautés sur **GPU** [SN11]
     - Posix thread, MPI, ...
- Connais. néces. en fouille des réseaux sociaux et en parallélisme.

## Usage des DSLs parallèles de graphe

- Implémentation avec le formalisme du DSL (une seule version)
- **Approche plus intuitive pour l'algorithmicien.**

# Objectif de la thèse

## Objectif de la thèse

- Fournir un **langage dédié** et un compilateur offrant une **facilité de programmation** et un **code optimisé** pour la **fouille des réseaux sociaux** sur **plates-formes muti-cœurs**.

## Caractéristiques visées du langage dédié

- Portabilité
- Efficacité, performance
- Productivité (pour les utilisateurs/algorithmiciens)
  - réduire le temps de dévelop. pour qu'ils se concentrent sur les aspects algorith.

# Les DSLs

## Définition et exemples

- Langage de programmation conçu pour un domaine précis
- Exemples : TeX, LaTeX, HTML, SQL, YACC, ...

## Approches d'implémentation [DKV00, KKP+09, MHS05]

1. DSL interne ou embarqué dans un langage généraliste (biblio. spécialisée)
2. DSL externe ou stand alone (son propre compilateur, ...)

## Travaux sur les DSLs parallèles :

- [CDM+10] propose un langage générique pour le dévelop. des DSLs parallèles
- **Green-Marl** [HCSO12], un DSL externe pour l'analyse des graphes
- **Galois** [NLP13], un DSL interne pour l'analyse des graphes.

# Étapes de mise en place du DSL

## Deux principales étapes

1. Définition du langage dédié à la fouile des réseaux sociaux
   - primitives algorithmiques et structures de données,
   - comment le runtime les prend en charge (ex : le stockage en mémoire).
2. Élaboration des méthodes de génération et d'optimisation de code
   - conformes aux observations obtenues,
   - tenant compte des spécificités des architectures multi-coeurs.

# Green-Marl

## Description générale

- DSL stand-alone,
- compilateur génère du code c++ (aussi gps et giraph),
- parallélisme avec des directives openMP,
- représentation de Yale [EGSS77].

## Éléments du langage

- DGraph, Ugraph, Node, Edge, Node/Egde properties ;
- Foreach (exécution par.) vs for (exécution séq.)
- BFS (parcours en largeur, par.) vs DFS (en profondeur, seq.)

# Green-Marl, $attach\_score(x, y) = |\Gamma(x)|.|\Gamma(y)|$

```
Procedure preferential_attachment(G:Graph){
  Foreach(n1:G.Nodes){
    Foreach(n2:G.Nodes)(n1 < n2){
      Int pref_attach = n1.Degree() * n2.Degree();
 } } }
```

```
 #include "preferential_attachment.h"
void preferential_attachment(gm_graph& G){
gm_rt_initialize(); G.freeze(); //Initializations
for (node_t n1 = 0; n1 < G.num_nodes(); n1 ++)
{ #pragma omp parallel for
  for (node_t n2 = 0; n2 < G.num_nodes(); n2 ++)
  { if (n1 < n2){
      int32_t pref_attach = 0 ;
      pref_attach = (G.begin[n1+1] - G.begin[n1])*
                    (G.begin[n2+1] - G.begin[n2]) ;}}}}
```

# Avantages et limites de Green-Marl

## Avantages : facilité de programmation offerte

- parallélisme transparent au programmeur,
- primitives proches des experts de l'analyse des graphes

## Les limites de Green-Marl

- ne permet pas de définir de nouveaux types de donnée,
- aucune action particulière à faire par runtime,
- ne tient pas compte des propriétes des graphes sociaux.

# DSL Galois

## Description

- DSL embarqué dans C++,
- usage des templates (squelette à remplir) c++,
- spécifie la partie du code qui sera exécutée en parallèle
- intègre son propre ordonnanceur gérant les threads pendant l'exécution
- représentation sous forme de Yale [EGSS77]

## Squelette divisé en 3

- Partie déclaration du graphe,
- Partie opérateur : fonction exécutée pour chaque noeud
- Partie itération : on précise les bornes

# Galois, $attach\_score(x, y) = |\Gamma(x)|.|\Gamma(y)|$

Déclaration

Opérateur

Itération

```
typedef Galois::Graph::FirstGraph<int,void,true> Graph;
typedef Graph::GraphNode GNode;
typedef std::pair<unsigned, GNode> UpdateRequest;
Graph graph;
std::vector<GNode> node_vect;
int nb_neighbors(int node, Graph& graph){...}

void preferential_attach_node(Graph& G, int32_t n1)
{
            int nb_node = node_vect.size(), n2, attach_scr;
            for(n2 = 0; n2 < nb_node; n2++)
            {
                if(n1!=n2){
                        attach_scr = nb_neighbors(n1, G)*nb_neighbors(n2, G);
                        printf("%d %d,%d\n",n1,n2,attach_scr);
                }
            }
}

struct preferential_attach{
  void operator()(GNode& n, Galois::UserContext<GNode>& ctx) const {
     preferential_attach_node(graph, graph.getData(n));
  }
};
int main(int argc, char **argv) {
  node_vect = load_graph_gml(argv[1], &graph);
  Galois::for_each(graph.begin(), graph.end(), preferential_attach());
  return 0;
}
```

# Avantages et limites de Galois

## Avantage : bonne gestion du runtime

- ordonnancement optimisé des threads pendant le runtime,
- temps d'exécution plus court (vs ligra [SB13], powergraph [GLG+12]).

## Limites de Galois

- la facilité de programmation n'est pas assurée (Comparé à green-marl),
- n'exploite pas les propriétés des graphes sociaux,
- n'a pas une bonne documentation.

# Remarques exploitables dans la suite de la thèse

## Questions pour la suite

- Faut-il refaire un nouveau DSL ou alors étendre un existant ?
  - Étendre
- Lequel étendre ?
  - Green-Marl
- Comment ?
  - Au niveau de la syntaxe (amélioration)
  - Au niveau du code généré (analyse et optimisation)
  - Au niveau runtime (analyse optimisation suivant la plate-forme cible)

## Travail actuel : propriétes des graphes sociaux vs temps d'exécution

1. **structure de données exploitant le regroup. en comm. des nœuds**
2. ordonnancement des threads exploitant la distribution en puissance des nœuds

# Merci de votre aimable attention!


Hassan Chafi, Zach DeVito, Adriaan Moors, Tiark Rompf, Arvind K Sujeeth, Pat Hanrahan, Martin Odersky, and Kunle Olukotun.
Language virtualization for heterogeneous parallel computing.
In *ACM Sigplan Notices*, volume 45, pages 835–847. ACM, 2010.


Marcio Castro, Emilio Francesquin, Thomas Messi Nguélé, and Jean-François Méhaut.
Multicœurs et manycœurs : Une analyse de la performance et l'Éfficacité Énergétique d'une application irrégulière.
*Proceedings of the CRI'2013, Conférence de Recherche en Informatique*, decembre 2013.
Accepted.


Márcio Castro, Emilio Francesquini, Thomas M Nguélé, and Jean-François Méhaut.
Analysis of computing and energy performance of multicore, numa, and manycore platforms for an irregular application.
In *Proceedings of the 3rd Workshop on Irregular Applications : Architectures and Algorithms*, page 5. ACM, 2013.


Arie Van Deursen, Paul Klint, and Joost Visser.
Domain-specific languages : An annotated bibliography.
*ACM SIGPLAN NOTICES*, 35 :26–36, 2000.


Stanley C Eisenstat, MC Gursky, MH Schultz, and AH Sherman.
Yale sparse matrix package. i. the symmetric codes.
Technical report, DTIC Document, 1977.


Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin.
Powergraph : Distributed graph-parallel computation on natural graphs.
In *OSDI*, volume 12, page 2, 2012.


Sungpack Hong, Hassan Chafi, Edic Sedlar, and Kunle Olukotun.
Green-marl : a dsl for easy and efficient graph analysis.
In *ACM SIGARCH Computer Architecture News*, volume 40, pages 349–362. ACM, 2012.


Leo Katz.
A new status index derived from sociometric analysis.

*Psychometrica-vol.18, No.1*, 18(1), March 1953.


Gabor Karsai, Holger Krahn, Class Pinkernell, Bernhard Rumpe, Martin Schneider, and Steven Völkel.
Design guidelines for domain specific languages.
In Matti Rossi, Jonathan Sprinkle, Jeff Gray, and Juha-Pekka Tolvanen, editors, *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, pages 7–13, 2009.


Vanessa Kamga, Maurice Tchuente, and Emmanuel Viennet.
Prévision des liens dans les graphes bipartites avec attributs.
Mémoire de master 2 recherche, Université de Yaoundé 1, Département d'Informatique, Septembre 2013.


Marjan Mernik, Jan Heering, and Anthony M Sloane.
When and how to develop domain-specific languages.
*ACM computing surveys (CSUR)*, 37(4) :316–344, 2005.


Donald Nguyen, Andrew Lenharth, and Keshav Pingali.
A lightweight infrastructure for graph analytics.

In *Proceedings of ACM Symposium on Operating Systems Principles*, SOSP '13, pages 456–471, 2013.


Jason Riedy, David A. Bader, and Henning Meyerhenke.
Scalable multi-threaded community detection in social networks.
*IEEE Computer Society Washington, DC, USA 2012*, 18(1) :1619–1628, 2012.
IPDPSW '12 Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum.


Julian Shun and Guy E Blelloch.
Ligra : a lightweight graph processing framework for shared memory.
In *ACM SIGPLAN Notices*, volume 48, pages 135–146. ACM, 2013.


Jyothish Soman and Ankur Narang.
Fast community detection algorithm with gpus and multicore architectures.
*IEEE International Parallel Distributed Processing Symposium*, 2011.