INVENTEURS DU MONDE NUMÉRIQUE

# How to mature a 20 y.o.


François Pellegrini

EQUIPE PROJET
**BACCHUS**
**Bordeaux**
**Sud-Ouest**

02/02/2012

# Outline of the talk

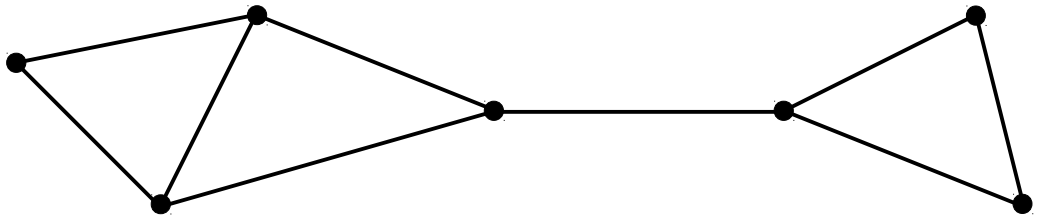
- Graph partitioning
- The 

project and history
- Licensing issues
- Some lessons (to be) learnt

# Graph partitioning

# What are graphs

- A graph is a set of vertices, linked by edges



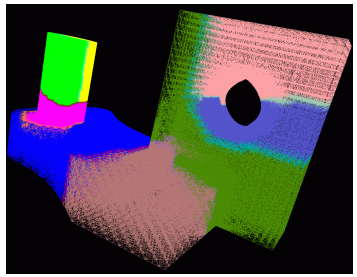
- Graphs are a versatile tool for representing problems :
  - Minimization of delivery trips
    - E.g. « Traveling Salesman Problem »
    - Search for « Hamiltonian paths »
  - Determination of maximum flow in a network
    - Search for « max flow / min cut »

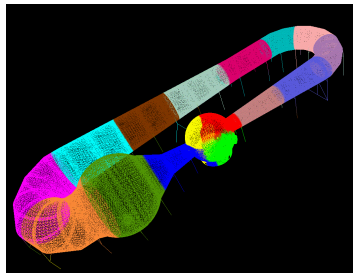
# Graph partitioning (1)

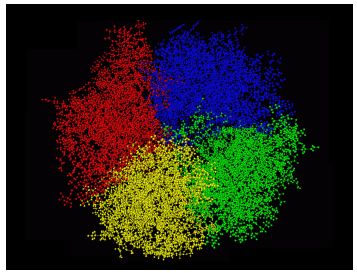
- Graph partitioning is an ubiquitous technique which has proven useful in a wide number of application fields
  - Used to model domain-dependent optimization problems
  - “Good solutions” take the form of partitions which minimize vertex or edge cuts, while balancing the weight of graph parts
- NP-hard problem in the general case
- Many algorithms have been proposed in the literature :
  - Graph algorithms, evolutionary algorithms, spectral methods, linear optimization methods, …

# Graph partitioning (2)

- Two main problems for our team, in relation to sparse linear system solving (Ax = b) :
  - Sparse matrix ordering for direct methods
  - Domain decomposition for iterative methods
- These problems can be modeled as graph partitioning problems on the adjacency graph of symmetric positive-definite matrices
  - Edge separator problem for domain decomposition
  - Vertex separator problem for sparse matrix ordering by nested dissection

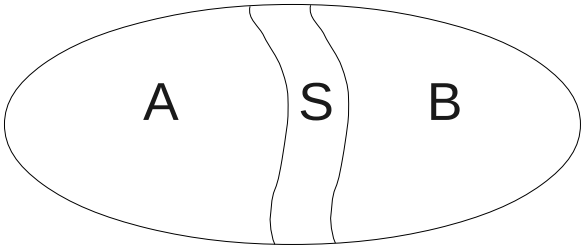


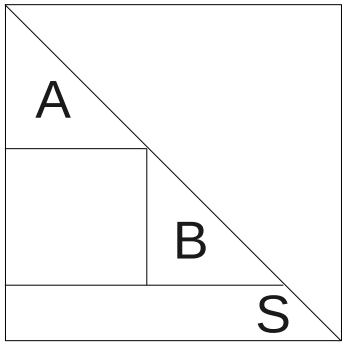


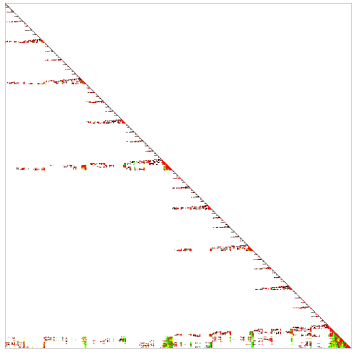


# Nested dissection

- Top-down strategy for removing potential fill-inducing paths
- Principle [George, 1973]
  - Find a vertex separator of the graph
  - Order separator vertices with available indices of highest rank
  - Recursively apply the algorithm on the separated subgraphs



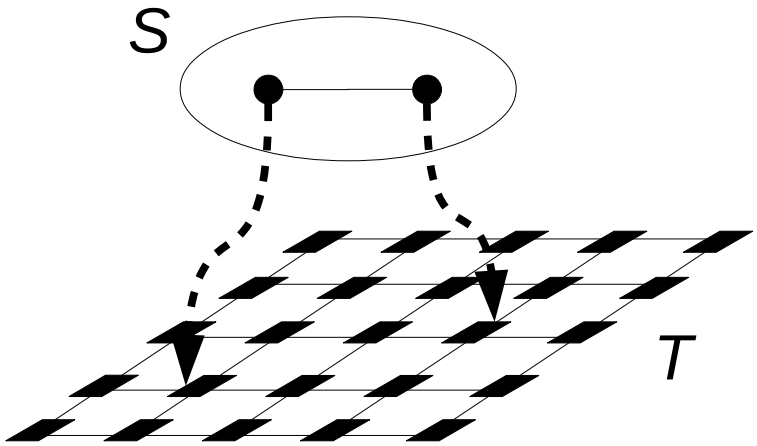



# The  project and history

# The Scotch project (1)

- Provide a set of fast heuristic algorithms and tools for vertex and edge graph partitioning and for static mapping
- Static mapping is a generalization of the graph partitioning problem in which vertices of a source graph S have to be mapped onto vertices of a target graph T
  - Communication cost function accounts for distance




$$f_C(\tau_{S,T}, \rho_{S,T}) \stackrel{\text{def}}{=} \sum_{e_S \in E(S)} w(e_S)\, |\rho_{S,T}(e_S)|$$

# The Scotch project (2)

- Previous roadmap : should handle graphs of more than a billion vertices distributed across one thousand processors

DONE !

- Current roadmap : should handle graphs of a trillion vertices distributed across one million processors
  - Account for heavily non uniform parallel architectures
  - Asynchronous algorithms

# The Scotch history (1)

- Dec. 1992 : Start coding of v0.0
  - Algorithms for static mapping
- May 1994 : First published conference paper
- Jul. 1995 : Start coding of V3.0
  - First version planned to be publicly released
    - Competing non-free software MeTiS was available from the web
- Aug. 1996 : Start coding of v3.2
  - Algorithms for sparse matrix ordering
- Sep. 1996 : First website for public release of v3.0 under binary form
- Sep. 1999 : First license form for source code

```
# Version 0.0  : from : 02 dec 1992
                 to     18 may 1993
# Version 1.3  : from : 30 apr 1994
                 to     18 may 1994
# Version 2.0  : from : 06 jun 1994
                 to     18 aug 1994
# Version 3.0  : from : 07 jul 1995
                 to     28 sep 1995
# Version 3.1  : from : 28 nov 1995
                 to     28 nov 1995
# Version 3.2  : from : 07 sep 1996
                 to     15 sep 1998
# Version 3.3  : from : 28 sep 1998
                 to     23 mar 1999
# Version 3.4  : from : 20 mar 2000
                 to     20 mar 2000
# Version 4.0  : from : 24 nov 2001
                 to     03 mar 2006
# Version 5.0  : from : 03 mar 2006
                 to     01 jun 2008
# Version 5.1  : from : 11 aug 2010
                 to     04 nov 2010
# Version 6.0  : from : 03 mar 2011
                 to     04 sep 2011
```

# The Scotch history (2)

- Nov. 2001 : Start coding of v4.0
- Oct. 2004 : Start coding of v5.0
  - Parallel versions of sparse matrix ordering code
- Feb. 2006 : Release of v4.0 as free software under LGPL
  - Project hosted by Inria Gforge
- Aug. 2007 : Release of v5.0 as free software under CeCILL-C
  - PT-Scotch parallel offspring
- Sep. 2008 : Start coding of v6.0
- Dec. 2008 : Start coding of v6.1
- Dec. 2012 : Release of v6.0
  - 20 years after coding of v0.0 started

# (Free) software in science

# Place of software in research

- In the world of research, one can see software :
  - As an end :
    - Demonstrator of algorithmic feasibility
    - Mathematical proof of existence
  - As a mean :
    - Self-crafted tool
    - Necessary to the obtainment of some results
  - It is usually both at the same time
- Scientific reproducibility imposes that software be available along with papers that exhibit its results
  - A policy regarding technical and legal means for accessing such software must be set up

# What to do with produced software ? (1)

- A research laboratory is not supposed to be a software editor
  - A software may become useless from a research point of view but still be highly valuable from an application point of view
  - The value placed into the former development of such software must not be lost
    - Unused software is wasted money
  - Leadership on software development and maintenance may evolve
    - This has to be anticipated and encouraged
    - Free software licenses are most often a very suitable tool for this purpose

# What to do with produced software ? (2)

- Application maintenance is not part of the tasks of a scientist
  - Yet, it is necessary to build and maintain a user community
  - Its cost/benefit ratio has to be carefully evaluated

# What to do with produced software ? (3)

- The cost of turning research software into production-grade products can be high
- Yet, this step is necessary so as not to lose software value
- Several complementary means can be envisioned :
  - Technology transfer contracts with industry
    - But community is likely to lose further developments if the industrial version becomes privative/proprietary
  - Allocation of dedicated means by the research institution
    - Software engineers, not PhD's or post-doc's !
    - Beware of interns ! ;-)

# License issues

# Ownership of author's rights (1)

- Software is covered by author's rights, like many other works of the mind
  - Yet, standard author's rights do not apply
- Software authors who are civil servants or company employees see their patrimonial author's rights automatically transferred to their employer
- Only the employer can decide about :
  - Whether the software can be made publicly available or not
  - Under what license(s) it can be made available

# Ownership of author's rights (2)

- Necessity to track contributions
  - Whenever handling licensing issues, author's rights must be asserted
    - Better to do it beforehand
- Beware of interns !
  - The author's rights of unpaid interns are not automatically transferred to the employer !
  - Problem of searching for the members of the “Disappeared Intern's Society”...
    - Some projects had to hire employees to re-code many critical modules

# Choosing the proper license

- Select a license that is suitable to your project and acceptable by your community
  - As a civil servant, my results have to be used by the majority of the taxpayers and citizens
    - Weak copyleft licenses are interesting in this respect
- Advocate the fact of releasing your code to your employer
  - This process can be long, all the more when several institutions participated in the funding
    - In the case of 

: CNRS, ENSEIRB, Inria, Université Bordeaux 1
  - Find relevant arguments :
    - “My software is crap and nobody will use it anyway”
    - There already exist competitors using these licenses
    - ...

# Benefits of going free software

- Inclusion of software on the form of packages within the main free software distributions
  - Increased visibility : Linux (Debian, Ubuntu), FreeBSD, ...
  - Packaging done by autonomous mainteners (Debian Science, ...)
- Exclusive use within academic and/or industrial free software
  - E.g. OpenFOAM
- No contribution to the software itself
  - Expertise is scarce, mostly owned by competitors
    - Build a testbed environment that they can join !

# Choosing the proper license (2)

- Within a given class, choose the license according to its own merits and to environmental constraints
- In the case of 

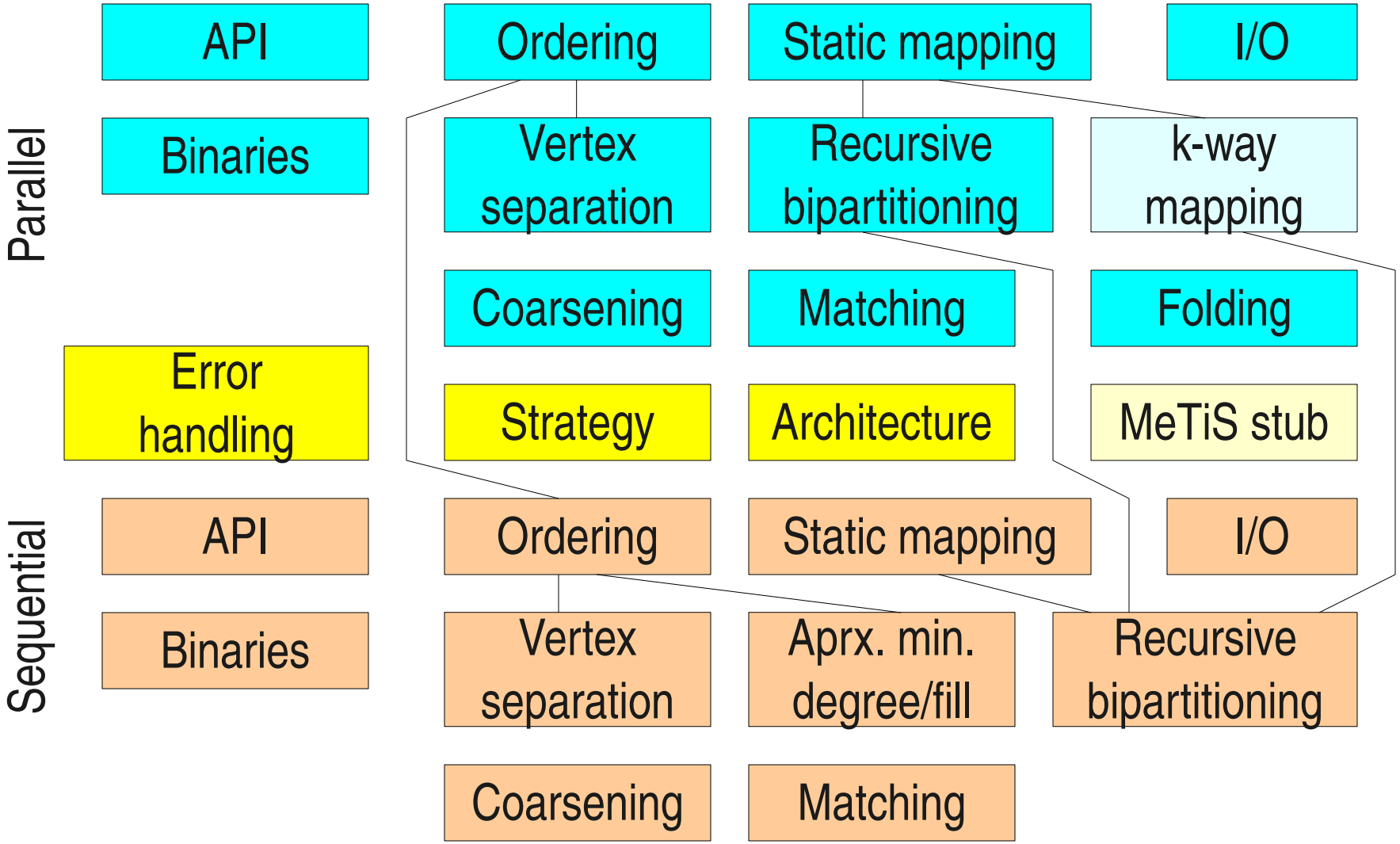
, for weak copyleft licenses :
  - LGPL allows “legal leaking” towards GPL
  - Inria is my employer
  - So... CeCILL-C
- Define a licensing policy from the inception of your project
  - Using a free software license reduces the impact of external contributors as long as the software is kept within the same license perimeter

# Some lessons (to be) learnt

# Be paranoid about quality (1)

- Strict rules have to be defined and enforced since the inception of the project regarding :
  - Architectural conventions
    - The structure of the software should be clearly exposed
  - Naming conventions
    - Names should reflect architecture and function
    - A given variable or routine function should result in a single canonical name
  - Coding standards
    - For reader's and writer's sake
- Always aim at durability and extensibility !

# Structure of the Scotch package (1)



# Structure of the Scotch package (2)

- All data structures are defined by a C type (aka “class”)
  - `Graph` type in graph.h, etc...
- Routines are grouped by type name and function (methods)
  - `arch_*` : target architectures
  - `bgraph_*` : sequential graph bipartitioning
  - `bdgraph_*` : parallel graph bipartitioning
  - `dgraph_*` : parallel graph handling
  - `kdgraph_*` : parallel k-way static mapping
  - `vdgraph_*` : parallel vertex separation
  - `vgraph_*` : sequential vertex separation
  - ...

# Structure of the Scotch package (3)

- Method files are identified by their type of computation :
  - `b?graph_bipart_xy` : edge graph bipartitioning method
  - `k?graph_map_xy` : static mapping method
  - `h?graph_order_xy` : graph ordering method
  - `v?graph_separate_xy` : vertex graph separation method
  - `hmesh_order_xy` : node mesh ordering method
  - `vmesh_separate_xy` : node mesh separation method
  - ...
  -

# Be paranoid about quality (2)

- Every data structure should have an axiom checker routine attached to it
  - Written before the data structure is used !
  - Called at the end of every routine that modifies a data structure of its kind
- When used at the beginning of the library API routines, they help debug user's software
  - Eternal worshiping easily earned... ;-)

# Thank you for your attention !

Any questions ?

http://scotch.gforge.inria.fr/