Peut-on adapter un framework HEP aux architectures *"many-core"* ? Application à CMSSW

G. Grasseau

Laboratoire Leprince-Ringuet, École polytechnique, Palaiseau
email{ grasseau}@llr.in2p3.fr

JI IN2P3 2012, Agelonde, 24-26 octobre 2012

Motivation

Introduction

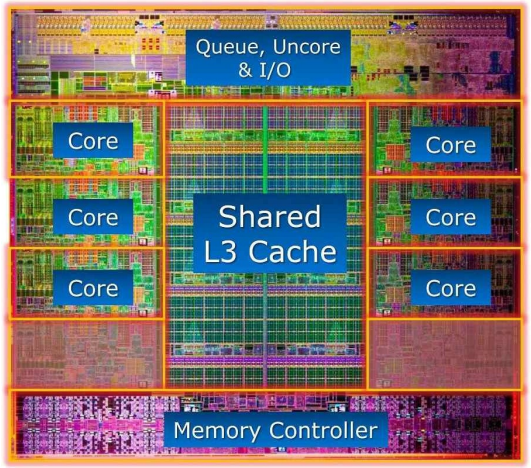


Figure: Intel Xeon E5-2600, Sandy Bridge : 2.6×10^6 transistors, 8 cores, 32 nm, 130 Watt, 1.3 TFlops, 3.0 Ghz, 1500 $

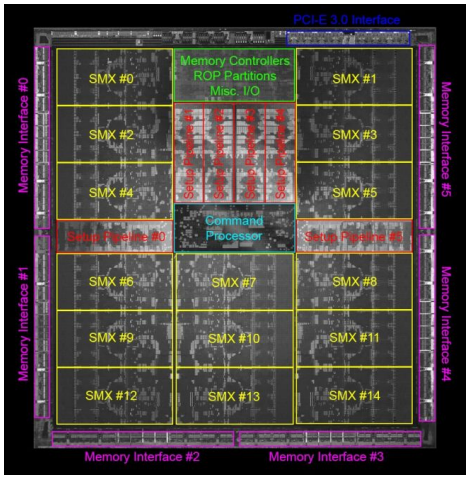


Figure: GPU Kepler : 7.1×10^6 transistors, 2880 CUDA cores, 28 nm, 200 Watt, 3 TFlops, 1 Ghz

- besoin de plus de puissance de calcul (synthèse, compression d'images 3D, vidéo, . . .)
- énergie consommée croissante $\rightarrow$ problème consommation/refroidissement
- le marché : très orienté vers les jeux vidéos (puissance de calcul) et portable (puissance de calcul/énergie)

Tendance forte : plus rentable de multiplier le nombre de coeurs que d'accroître la fréquence

Très attractif (GFlops/Watt, coût/GFlops), le monde du HPC un virage en cours (3 machines parmi le Top 10).

- GridCL P2IO : évaluer des architectures *many-core*, les intégrer dans la Grille, 2 projets pilotes (*tracking* ions lourds, *fitting* RooFit). Laboratoires : IAS, IMNC , IPNO , IRFU , LAL et LLR (D. Chamont).
- cadre gros *framework* du CERN (`CMSSW`) : accélérer le *tracking*
- Des technologies logicielles : compatible avec les processeurs *many-core* `Intel MIC`, `GPGPU` [1] → hétérogènes → `OpenCL`

Pas une description du parallélisme avec `OpenCL`

1

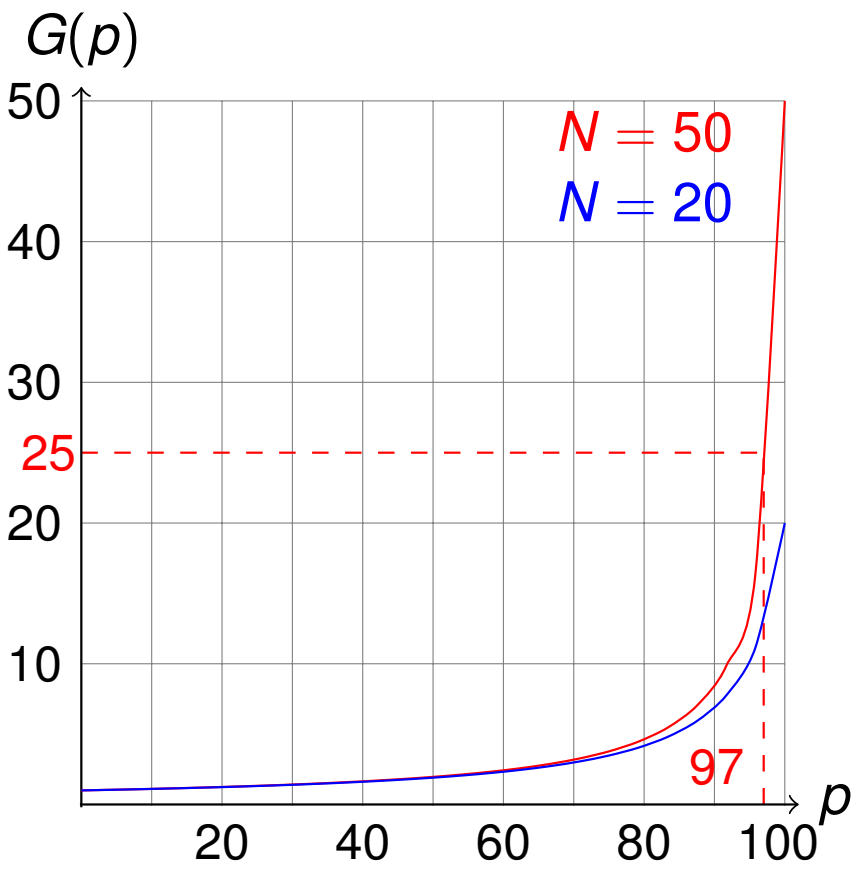


Figure: Loi d'Amdhal, accélération $G(p)$ en fonction de la fraction du code parallélisée p (en %) pour 20 (bleu) et 50 processeurs (rouge)

Loi d'Amdhal :

$$G = \frac{T_s}{T_p} = \frac{1}{(1-p) + p/N}$$

T_s temps d'exécution sur 1 processeur

p fraction de temps d'exécution //

T_p temps d'exécution sur N processeurs
$= (1-t)T_s + p.T_s/N$

Le taux $p \simeq 1$ pour N grand (> 20)

Conception d'une application parallèle :

- vue globale : algorithmes, schémas num., E/S
- algorithme plus coûteux (instructions) mais "scalable" (extensible).
- la mémoire (données) doit être distribuée et décroître en $1/N$
- localité des données, ↘ les échanges

Identification des difficultés

Introduction

Obstacles majeures :

- *framework* très riche, flexible, mais le parallélisme demande un approche globale . Gestion très lourde (`CMSSW` plus de 8000 classes, plus de 24 heures pour la compilation totale, . . .).
- architecture objet (écrit par des experts C++). → standard `OpenCL` (`C99`). Comment gérer le passage LOO → non LOO ?

Difficultés mineures :

- parallélisme / `SIMD GPGPU`, hiérarchie mémoire, `OpenCL`

Notre approche :

- pas un démonstrateur de type produit matriciel, ... peu représentatif des *framework*.
- une approche globale, prendre en considération la pérennisation des développements.
- règles d'écriture (simples mais structurées) : projection C++/C99 de l'organisation, de l'architecture (objet) du *framework*.

Généralités

Étude en performance

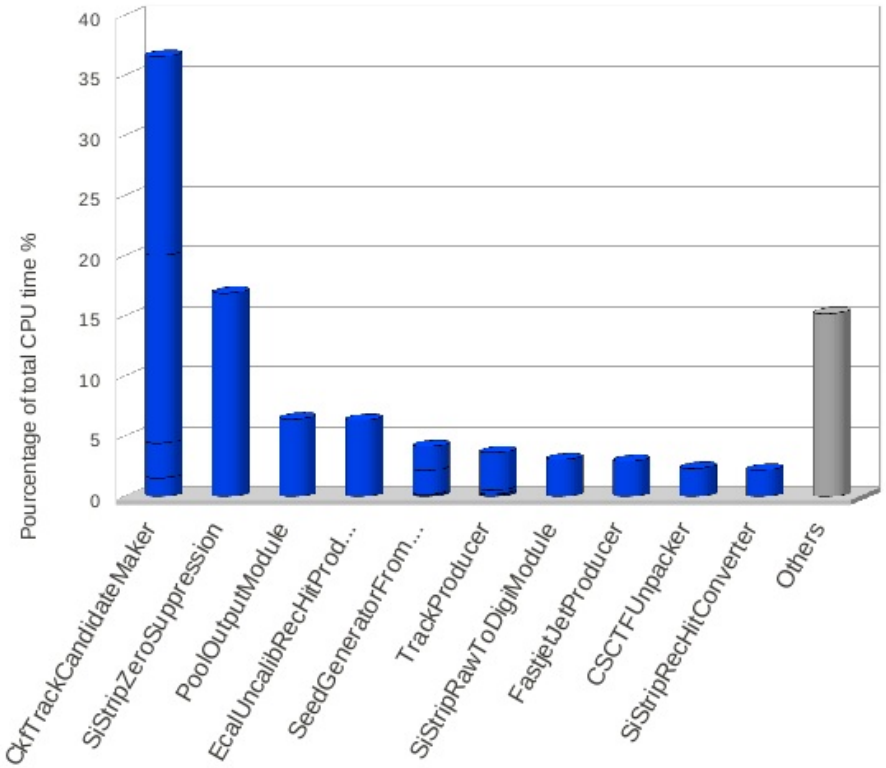
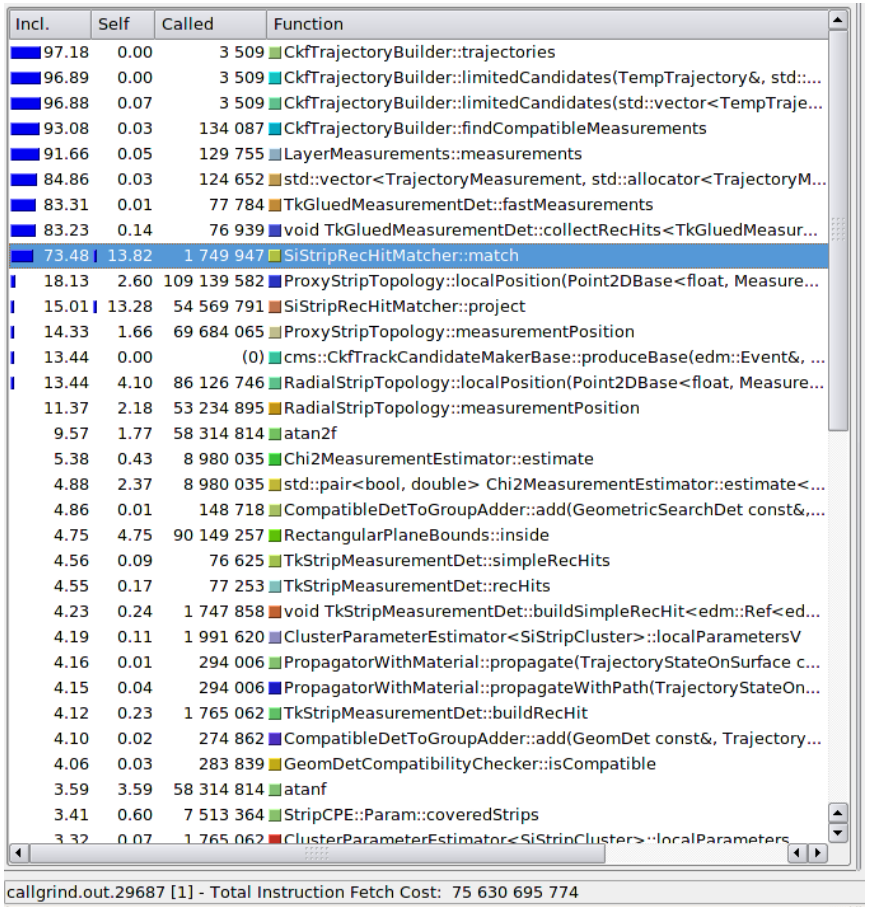


Figure: Les 10 modules les plus consommateurs de temps CPU - Mode itératif pour 100 évènements Pb-Pb (670 sec.)

- Parallélisation (locale) pas très favorable (Amdhal). Pourtant le traitement semble être fortement parallèle , il faudrait avoir une approche globale.
- HLT Alice : Approche par automate cellulaire sur GPU : distribution de la géométrie (mémoire décroît en $1/N$), ensemble de capteurs associés à un *thread GPU*.
- *framework* (CMSSW) : difficulté de changer cet environnement de travail

Candidats à l'optimisation

Étude en performance



Incl.	Self	Called	Function
97.18	0.00	3 509	CkfTrajectoryBuilder::trajectories
96.89	0.00	3 509	CkfTrajectoryBuilder::limitedCandidates(TempTrajectory&, std::...
96.88	0.07	3 509	CkfTrajectoryBuilder::limitedCandidates(std::vector<TempTraje...
93.08	0.03	134 087	CkfTrajectoryBuilder::findCompatibleMeasurements
91.66	0.05	129 755	LayerMeasurements::measurements
84.86	0.03	124 652	std::vector<TrajectoryMeasurement, std::allocator<TrajectoryM...
83.31	0.01	77 784	TkGluedMeasurementDet::fastMeasurements
83.23	0.14	76 939	void TkGluedMeasurementDet::collectRecHits<TkGluedMeasur...
73.48	13.82	1 749 947	SiStripRecHitMatcher::match
18.13	2.60	109 139 582	ProxyStripTopology::localPosition(Point2DBase<float, Measure...
15.01	13.28	54 569 791	SiStripRecHitMatcher::project
14.33	1.66	69 684 065	ProxyStripTopology::measurementPosition
13.44	0.00	(0)	cms::CkfTrackCandidateMakerBase::produceBase(edm::Event&, ...
13.44	4.10	86 126 746	RadialStripTopology::localPosition(Point2DBase<float, Measure...
11.37	2.18	53 234 895	RadialStripTopology::measurementPosition
9.57	1.77	58 314 814	atan2f
5.38	0.43	8 980 035	Chi2MeasurementEstimator::estimate
4.88	2.37	8 980 035	std::pair<bool, double> Chi2MeasurementEstimator::estimate<...
4.86	0.01	148 718	CompatibleDetToGroupAdder::add(GeometricSearchDet const&,...
4.75	4.75	90 149 257	RectangularPlaneBounds::inside
4.56	0.09	76 625	TkStripMeasurementDet::simpleRecHits
4.55	0.17	77 253	TkStripMeasurementDet::recHits
4.23	0.24	1 747 858	void TkStripMeasurementDet::buildSimpleRecHit<edm::Ref<ed...
4.19	0.11	1 991 620	ClusterParameterEstimator<SiStripCluster>::localParametersV
4.16	0.01	294 006	PropagatorWithMaterial::propagate(TrajectoryStateOnSurface c...
4.15	0.04	294 006	PropagatorWithMaterial::propagateWithPath(TrajectoryStateOn...
4.12	0.23	1 765 062	TkStripMeasurementDet::buildRecHit
4.10	0.02	274 862	CompatibleDetToGroupAdder::add(GeomDet const&, Trajectory...
4.06	0.03	283 839	GeomDetCompatibilityChecker::isCompatible
3.59	3.59	58 314 814	atanf
3.41	0.60	7 513 364	StripCPE::Param::coveredStrips
3.32	0.07	1 765 062	ClusterParameterEstimator<SiStripCluster>::localParameters

callgrind.out.29687 [1] - Total Instruction Fetch Cost: 75 630 695 774

Figure: Outils `cachegrind/callgrind`, uniquement *tracking*

Semble pas trop compliqué (commence à travailler) . . .

Candidats à l'optimisation

Étude en performance

Incl.	Self	Called	Function
97.18	0.00	3 509	CkfTrajectoryBuilder::trajectories
96.89	0.00	3 509	CkfTrajectoryBuilder::limitedCandidates(TempTrajectory&, std::...
96.88	0.07	3 509	CkfTrajectoryBuilder::limitedCandidates(std::vector<TempTraje...
93.08	0.03	134 087	CkfTrajectoryBuilder::findCompatibleMeasurements
91.66	0.05	129 755	LayerMeasurements::measurements
84.86	0.03	124 652	std::vector<TrajectoryMeasurement, std::allocator<TrajectoryM...
83.31	0.01	77 784	TkGluedMeasurementDet::fastMeasurements
83.23	0.14	76 939	void TkGluedMeasurementDet::collectRecHits<TkGluedMeasur...
73.48	13.82	1 749 947	SiStripRecHitMatcher::match
18.13	2.60	109 139 582	ProxyStripTopology::localPosition(Point2DBase<float, Measure...
15.01	13.28	54 569 791	SiStripRecHitMatcher::project
14.33	1.66	69 684 065	ProxyStripTopology::measurementPosition
13.44	0.00	(0)	cms::CkfTrackCandidateMakerBase::produceBase(edm::Event&, ...
13.44	4.10	86 126 746	RadialStripTopology::localPosition(Point2DBase<float, Measure...
11.37	2.18	53 234 895	RadialStripTopology::measurementPosition
9.57	1.77	58 314 814	atan2f
5.38	0.43	8 980 035	Chi2MeasurementEstimator::estimate
4.88	2.37	8 980 035	std::pair<bool, double> Chi2MeasurementEstimator::estimate<...
4.86	0.01	148 718	CompatibleDetToGroupAdder::add(GeometricSearchDet const&,...
4.75	4.75	90 149 257	RectangularPlaneBounds::inside
4.56	0.09	76 625	TkStripMeasurementDet::simpleRecHits
4.55	0.17	77 253	TkStripMeasurementDet::recHits
4.23	0.24	1 747 858	void TkStripMeasurementDet::buildSimpleRecHit<edm::Ref<ed...
4.19	0.11	1 991 620	ClusterParameterEstimator<SiStripCluster>::localParametersV
4.16	0.01	294 006	PropagatorWithMaterial::propagate(TrajectoryStateOnSurface c...
4.15	0.04	294 006	PropagatorWithMaterial::propagateWithPath(TrajectoryStateOn...
4.12	0.23	1 765 062	TkStripMeasurementDet::buildRecHit
4.10	0.02	274 862	CompatibleDetToGroupAdder::add(GeomDet const&, Trajectory...
4.06	0.03	283 839	GeomDetCompatibilityChecker::isCompatible
3.59	3.59	58 314 814	atanf
3.41	0.60	7 513 364	StripCPE::Param::coveredStrips
3.32	0.07	1 765 062	ClusterParameterEstimator<SiStripCluster>::localParameters...

callgrind.out.29687 [1] - Total Instruction Fetch Cost: 75 630 695 774

Figure: Outils `cachegrind/callgrind`, uniquement *tracking*

Semble pas trop compliqué (commence à travailler) . . .

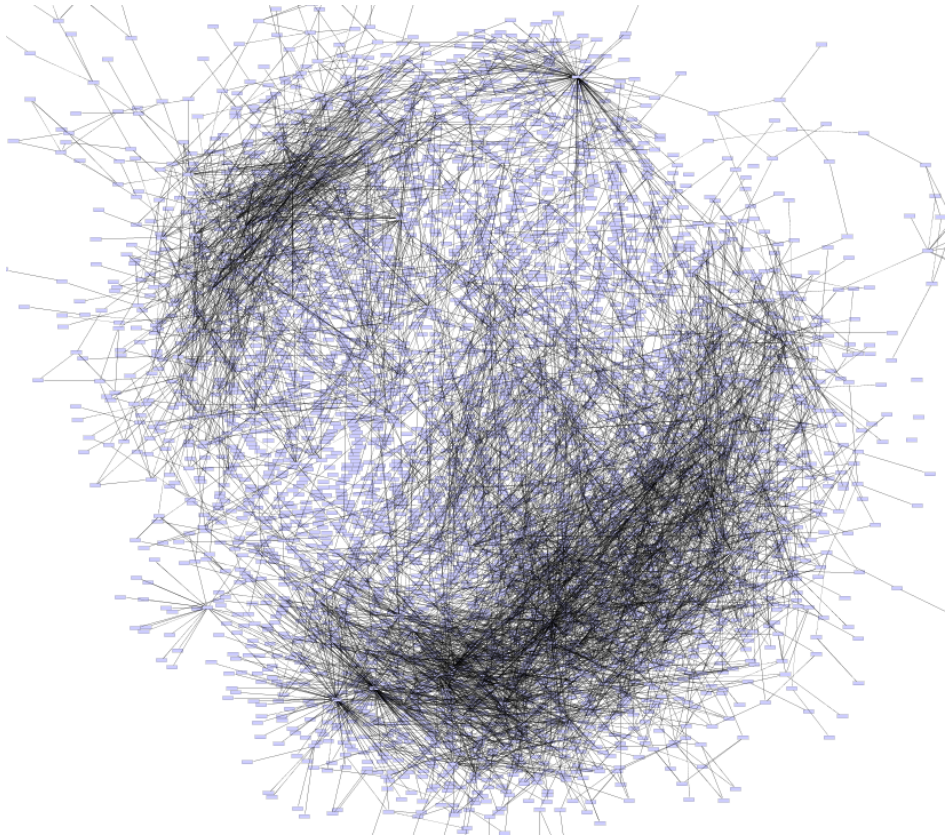


Figure: Arbre d'appel des méthodes/fonctions (`callgrind.xxx` → outil `python` → `Yed` (gros graphes). Méthodes `inline`, `std::`, `new`, `delete` pas représentées

. . . trop gros pour commencer → `SiStripRecHitMatcher::match`

Candidats à l'optimisation

Étude en performance

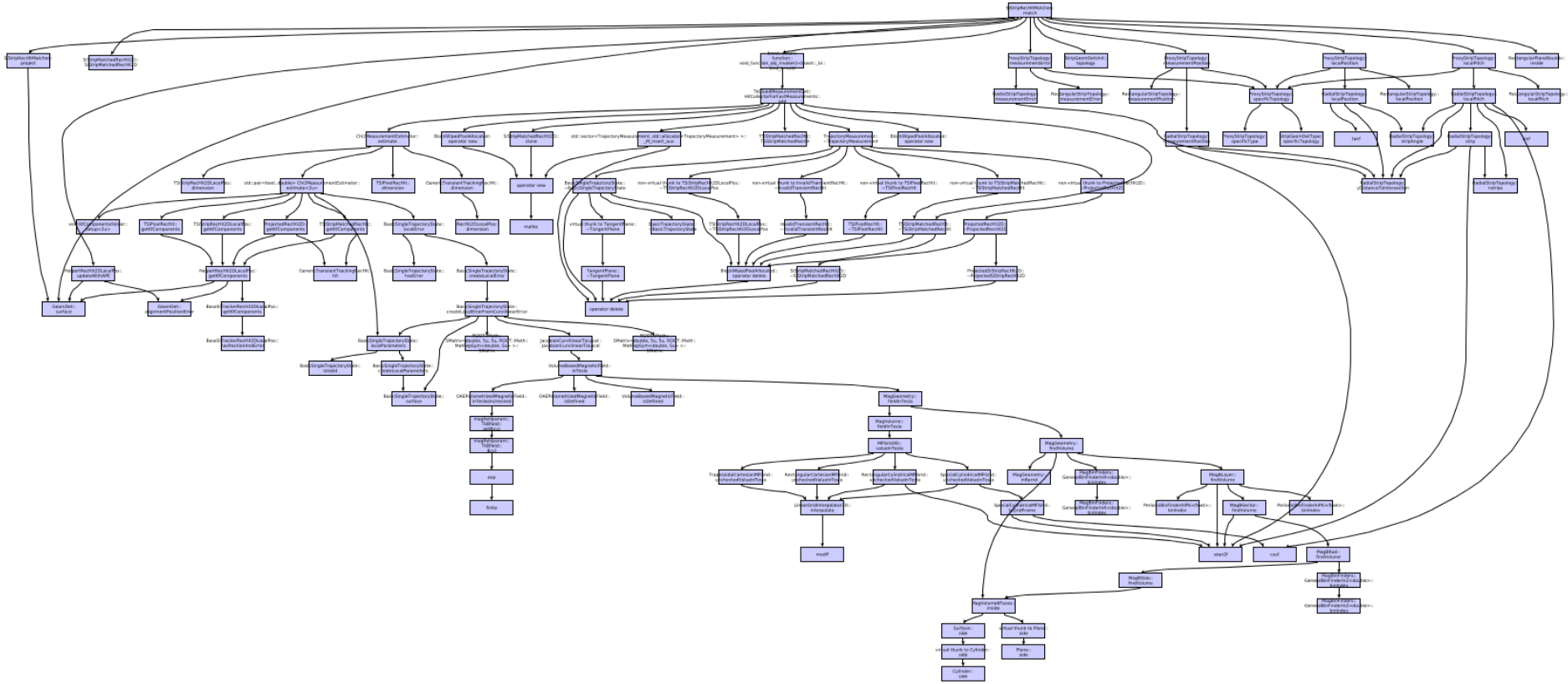


Figure: Arbre d'appel des méthodes/fonctions de `SiStripRecHitMatcher::match`

- Compromis, identification $\frac{\text{gain en performance}}{\text{coût développement}}$

Outils indispensables

Étude en performance

Trouver de bons outils pour les *frameworks*.

- IDE `NetBeans` (`Eclipse`) - nécessite 8 Go, configuration pas immédiate (`scram b`, ...), débogage difficile $\rightarrow$ limites.
- Documentation `CMSSW` (Doxygen): http://cmssdt.cern.ch/SDT/doxygen/CMSSW_4_4_2_patch2/doc/html/
- Arbre des appels à l'exécution (méthodes virtuelles) : `Valgrind/Callgrind`, `gprof`
  - limitation de l'IHM (`Valgrind`, `NetBeans`)
  - outil spécifique : `callgrind.xxx` $\rightarrow$ outil `python` $\rightarrow$ `Yed` (gros graphes, `XML`).
  - Filtrage de méthodes : `std::`, `new`, `delete`, appels `libc`, ...

Principes de la correspondance C++ / C99

Projection C++/OpenCL

Démarrage difficile (depuis $<$ 1 an à IN2P3)

- Démarrage en faisant des simplifications dans le C99
- Rapidement perdu après avoir gère une $\simeq$ 20 classes :
- Beaucoup de mal à estimer le volume travail restant (outil base sur Valgrind)
- Perte de la cohérence avec le *framework* $\rightarrow$ application *one-shot* $\rightarrow$ poubelle

Construction pas a pas de la “projection” C++ $\rightarrow$ C99

- Conserver la structure du *framework*
- Établir des règles pour faire une implémentation cohérente
- Se rapprocher de l’écriture du *framework*
- Modifier, restructurer si nécessaire (parallélisme)

Principes de la correspondance C++ / C99

Projection C++/OpenCL

Nos besoins dans le cadre d'un *framework* C++ :

- préserver la hiérarchie des classes (pseudo-class C99)
- surcharge méthodes (fonction)
- héritages
- méthodes virtuelles

Pas de pointeur sur les fonctions en OpenCL.

Ce que l'on ne pourra traiter :

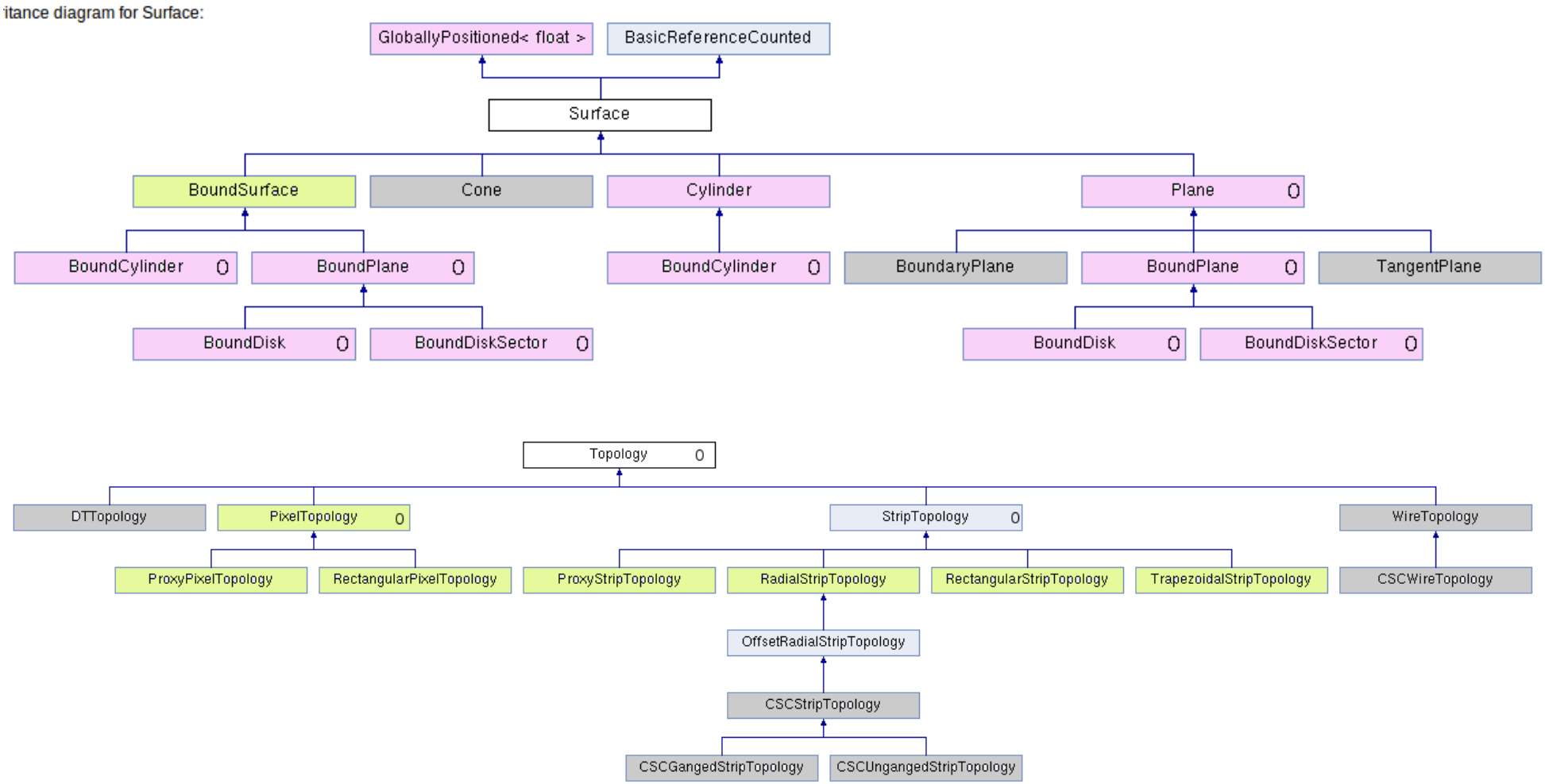
- L'encapsulation
- Les templates (préprocesseur!)

Le plus important étant la structure de données : hiérarchie des classes

Exemples d'héritage

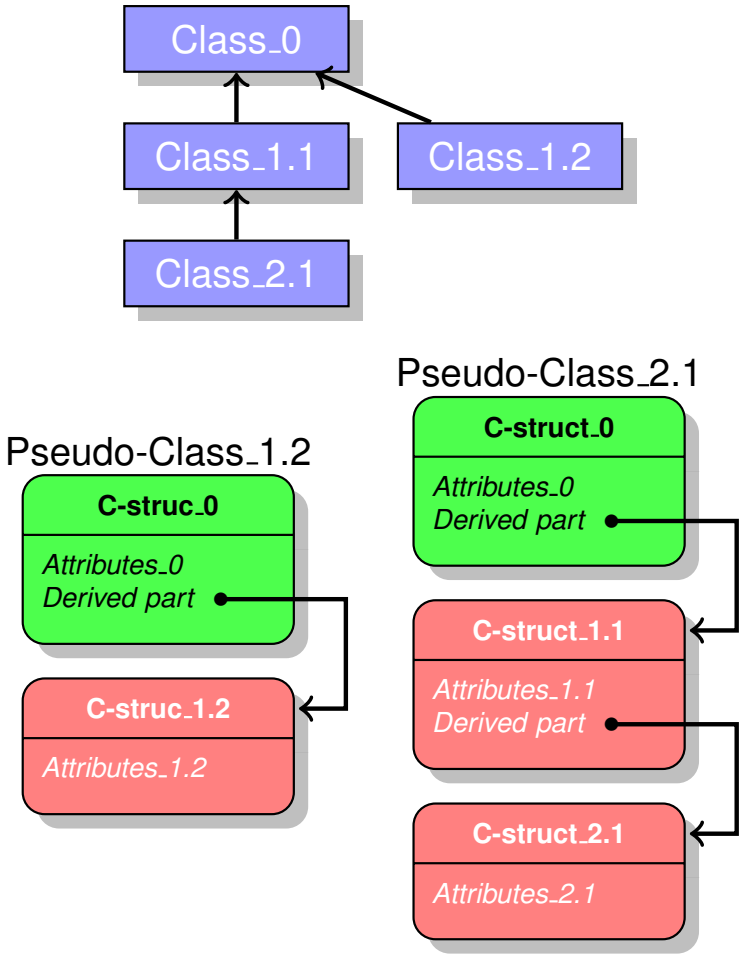
Projection C++/OpenCL

Exemples de structures hiérarchiques de classes (pas trop compliquées !):



Traitement de l'héritage

Projection C++/OpenCL



Avantages :

- gère le polymorphisme de "données"
- à partir d'une seule entité (pointeur *racine*), description de toutes les classes de la hiérarchie C++,

Inconvénients :

- lourd à gérer,
- perte du type pour les classes dérivées (structures pas autonomes) : `Class_x.y` → `C-struct_0`

Traitement du polymorphisme

Projection C++/OpenCL

Plusieurs polymorphismes :

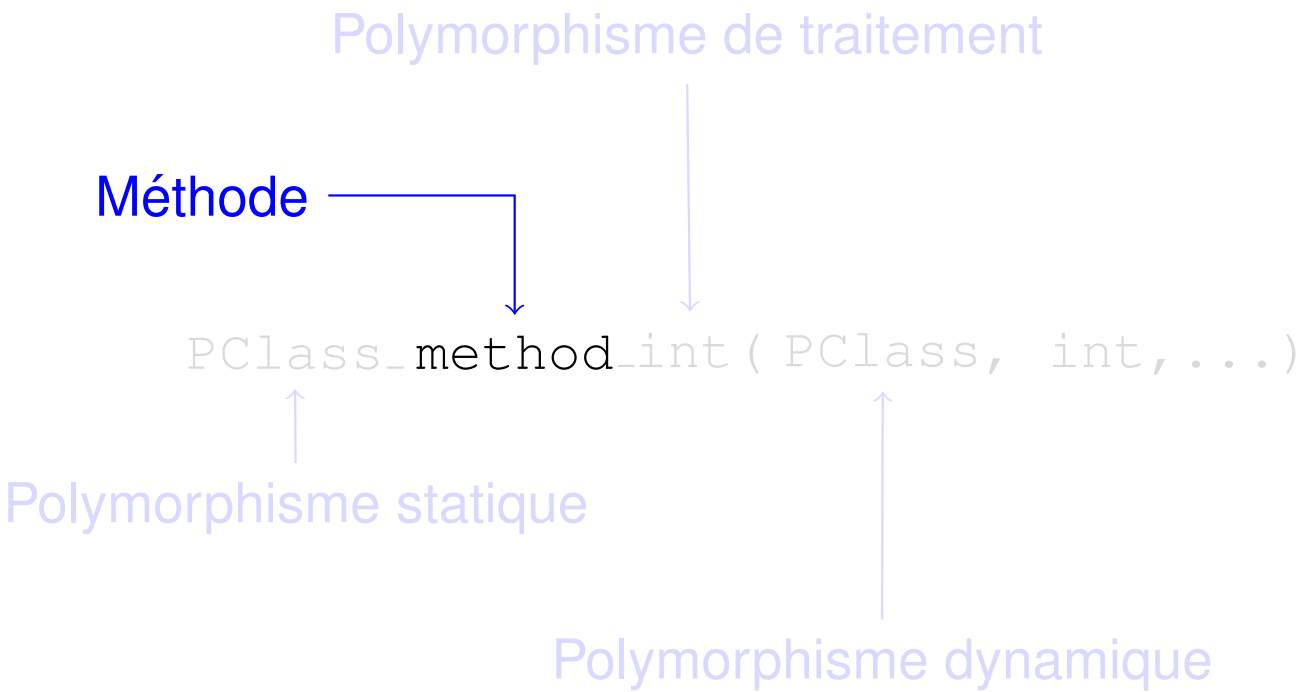
- polymorphisme de traitement : fonction, opérateur (surcharge)
- Polymorphisme d' inclusion (héritage)
  - statique
  - dynamique
- Polymorphisme paramétrique : un même code peut être applique à n'importe quel type $\rightarrow$ généricité (*Templates*)

Traitement du polymorphisme

Projection C++/OpenCL

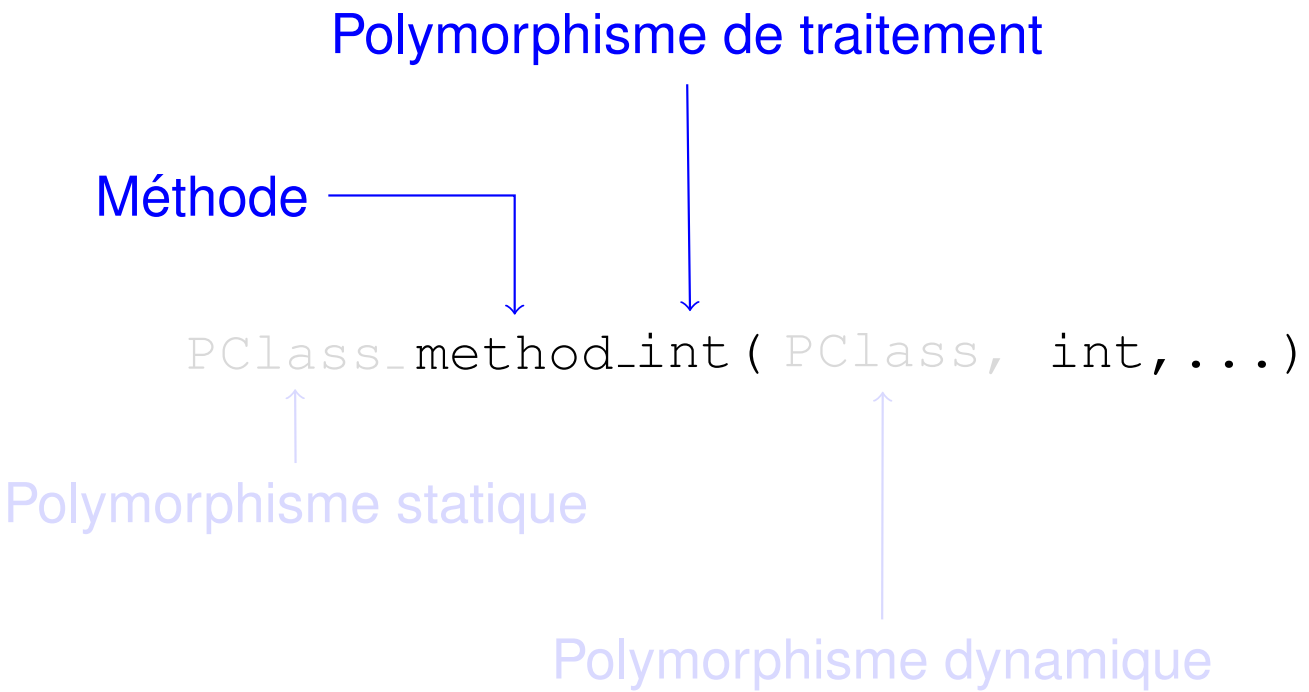
Traitement du polymorphisme

Projection C++/OpenCL



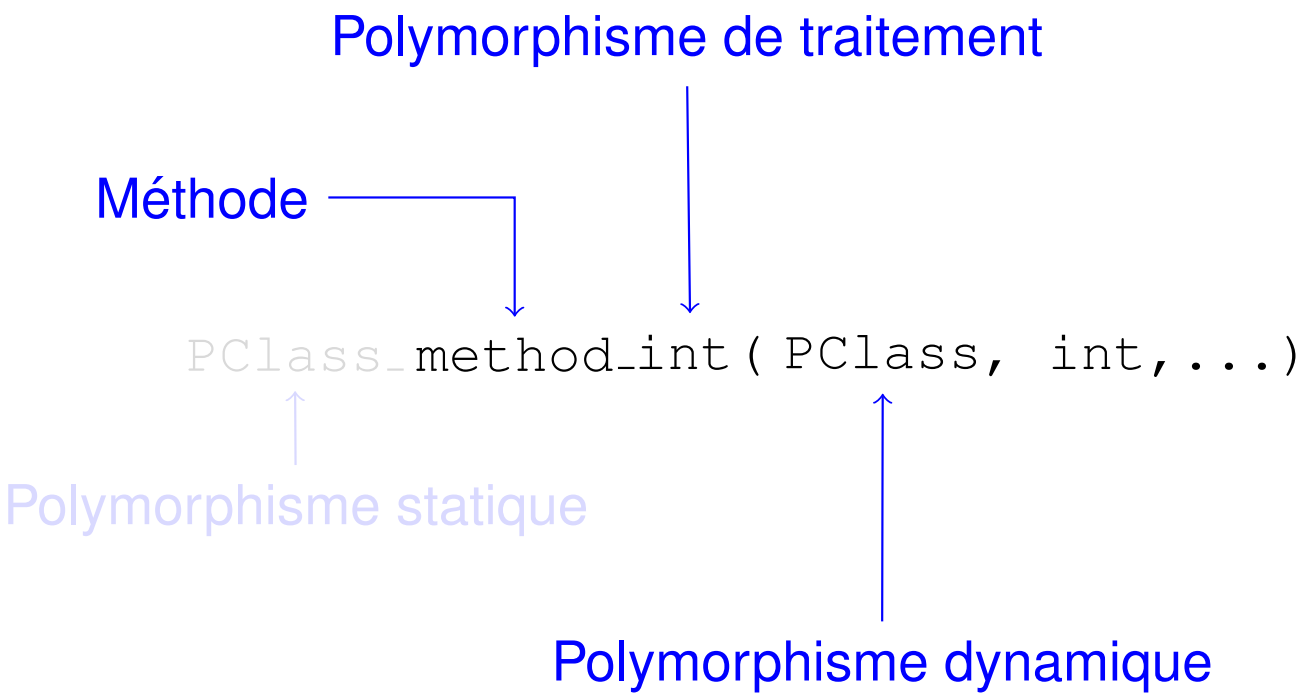
Traitement du polymorphisme

Projection C++/OpenCL



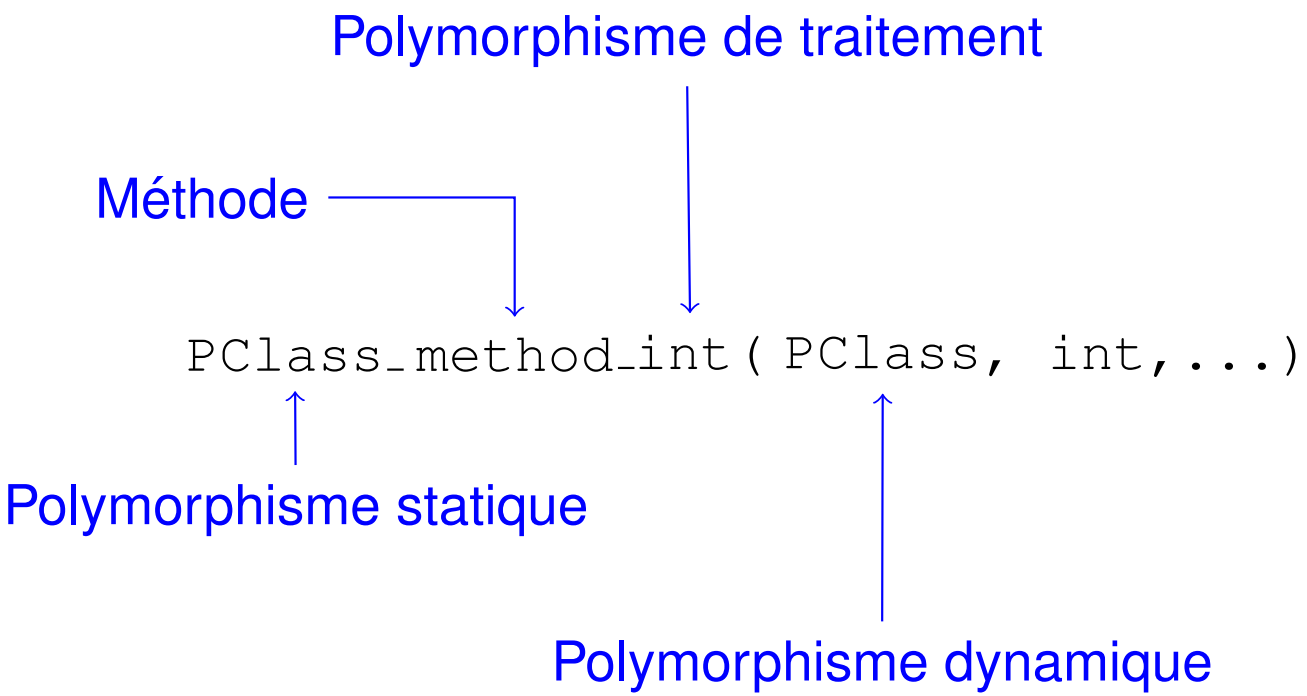
Traitement du polymorphisme

Projection C++/OpenCL



Traitement du polymorphisme

Projection C++/OpenCL



STL, Boost et autres

STL :

`vector` réécrite `CVector`

`auto_ptr` ou *smart pointer* (container pour déléger la libération de la mémoire) on fera tout à la main pendant les phases de recopies.

BOOST :

`shared_ptr` gère la désalocation mémoire par un *compteur de référence* (évite les redondances mémoires)

Exemple implémentation

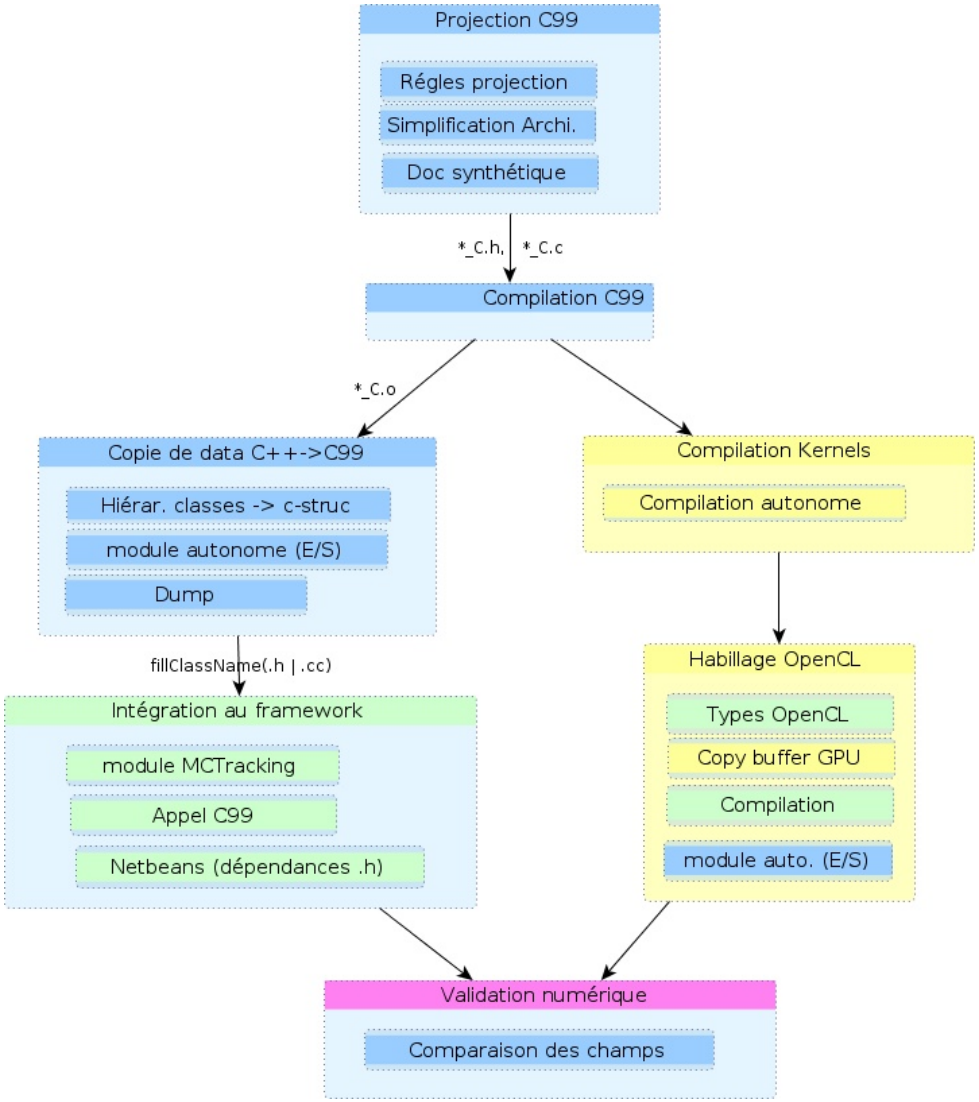
Projection C++/OpenCL

C++

```
SiStripRecHitMatcher::StripPosition
SiStripRecHitMatcher::project( const GeomDetUnit *det,
             const GluedGeomDet* glueddet,
             StripPosition strip, LocalVector trackdir
           ) const {

  GlobalPoint gp_ini =
               (det->surface()).toGlobal(strip.first);
  GlobalPoint gp_end =
               (det->surface()).toGlobal(strip.second);

  LocalPoint lp_ini =
              (glueddet->surface()).toLocal(gp_ini);
  LocalPoint lp_end =
              (glueddet->surface()).toLocal(gp_end);

  // correct the position with the track direction

  float scale = -lp_ini.z()/trackdir.z();

  LocalPoint lp_projini = lp_ini + scale*trackdir;
  LocalPoint lp_projend = lp_end + scale*trackdir;

  return StripPosition( lp_projini, lp_projend );
}
```

Projection C99

```
StripPosition
SSRHM_project( SiStripRecHitMatcher *ssrhm,
        const GeomDet *det,
        const GeomDet* glueddet,
        StripPosition strip, LocalVector trackdir ) {

  Surface*     surf = GD_surface( det );
  GlobalPoint gp_ini = toGlobal_Point3D( surf->theRot,
                          surf->thePos, strip.first  );
  GlobalPoint gp_end = toGlobal_Point3D( surf->theRot,
                          surf->thePos, strip.second );

  surf = GD_surface( glueddet );
  LocalPoint lp_ini = toLocal_Point3D( surf->theRot,
                                surf->thePos, gp_ini );
  LocalPoint lp_end = toLocal_Point3D( surf->theRot,
                                surf->thePos, gp_end);

  // correct the position with the track direction

  float scale = - lp_ini.z / trackdir.z;

  StripPosition lp_proj;
  addSVector( castPoint3DToArray( &lp_ini ), 3,
              scale, castVector3DToArray( &trackdir ),
              castPoint3DToArray( &lp_proj.first )  );

  addSVector( castPoint3DToArray(&lp_end), 3,
              scale, castVector3DToArray(&trackdir),
              castPoint3DToArray(&lp_proj.second)  );

  return lp_proj;
}
```



Plusieurs étapes :

- 75% de la projection est faite `SiStripRecHitMatcher::match` ($\simeq$ 100 méthodes), taille critique favorable au parallélisme
- Validation projection C++/C99 sur un sous-module ($\simeq$ 80 classes) `SiStripRecHitMatcher::project`

Autres activités

Conclusion

Autres activités “optimisation” (parallélisation, vectorisation) :

- RooFit (`GridCL`) : bibliothèque de *fitting* (S. Binet - LAL) `OpenMP`→ `OpenCL`
- BDT (`GridCL`): parallélisation avec `OpenMP` (A. Sartirana - LLR) étude sur un noeud (8 coeurs) → `GPGPU`. Nécessite une réorganisation de données.
- HESS (M. De Naurois) : code déjà optimisé (AVX) → amélioration, → `GPGPU`

Enjeux / Perspectives

Conclusion

Même si le projet “pilote” n’est pas idéal (beaucoup de complications) pour cette première tentative de parallélisation :

Enjeux / perspectives :

- Virage du *many-core* à ne pas rater (Marché)
- Pression pour optimiser `CMSSW` en prevision de la Haute Luminosité (*pile-up*)
- Sensibilisation de la communauté au *many-core*
- Projet au niveau du laboratoire, qui sert de référence et de première d’expérience au LLR
- Un début “expertise” sur le *tracking* pour le Laboratoire
- Mettre de l’automatisation dans le mécanisme de la projection (LRI, . . .)

Bibliographie

OpenCL et GPGPU :

- Standard OpenCL : http://www.khronos.org/opencl
- *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2* - B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, D. Schaa - Morgan Kauffman ed., 2012.
- *OpenCL Programming Guide for the CUDA Architecture* http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Programming_Guide.pdf
- *OpenCL Static C++ Kernel Language Extension* http://developer.amd.com/Assets/CPP_kernel_language.pdf

Parallélisme :

- *Parallelism in Experimental High Energy Physics Applications* - V.Innocente, Palaiseau, juin 2012. http://polywww.in2p3.fr/spip.php?page=view_semin&seminID=115

Remerciements

- Pour leur soutien : D. Chamont, P. Busson
- Pour les discussions : les membres de `GridCL`, V. Innocente, M. Nguyen, C. Charlot
- Pour votre attention