Retour d'expérience

École IN2P3 d'informatique 2016

« Parallélisme sur matériel hétérogène »

Vincent Lafage
lafage@ipno.in2p3.fr

S2I, Institut de Physique Nucléaire
Université d'Orsay

IPN
INSTITUT DE PHYSIQUE NUCLÉAIRE
ORSAY

Thème

« Découvrir les principales formes de **parallélisme** offertes par les **architectures matérielles** modernes. Explorer la façon d'en tirer parti en combinant plusieurs **technologies logicielles**, à la recherche du bon **compromis entre performance, portabilité et durabilité** du code. La plupart des technologies présentées seront pratiquées à travers un exemple simplifié de simulation de collision de particules. »

https://indico.in2p3.fr/event/13126/

INSTITUT DE PHYSIQUE NUCLÉAIRE
ORSAY

- le code (à la Rosetta Code : F77, F90, Ada, C++, C et Go)
  `https://bitbucket.org/bixente/3photons/src`
- différentes technologies mise en œuvre ou évoquées :
  - `OpenMP`,
  - `C++'11 & HPX`,
  - `OpenCL (+ MPI)`,
  - Intel Threading Building Blocks `TBB (C++)`
  - `DSL-Python`
  - `OpenCL pour FPGA`
- besoin d'abstraction
  - programmation fonctionnelle (*cf.* JI 2014).
  - `C++++`

Intégration numérique

INSTITUT DE PHYSIQUE NUCLÉAIRE ORSAY

⇒ méthode de ***Monte-Carlo*** (pas de raffinement adaptatif)

- générer un événement avec un certain poids,
  - générateur de nombres pseudo-aléatoires
  - traduit en un événement aléatoire

  RAMBO « *A new Monte Carlo treatment of multiparticle phase space at high energies* »
- voir s'il passe les coupures,
- calculer l'élément de matrice,
- pondérer,
- accumuler...

...et recommencer
Mais il y a deux types d'itérations :

1. celles qui dépendent de la précédente...
2. ... et les autres ⇒ Monte-Carlo est ***éhontément parallélisable*** (*embarassingly parallel*)

   ... ou plutôt ***délicieusement parallélisable***

INSTITUT DE PHYSIQUE NUCLÉAIRE ORSAY

Pseudo Random Number Generator

1. Linear congruential, Knuth *« Seminumerical Algorithms »*
2. Lüscher `Ranlux`
3. Marsaglia `xorshift`
4. Mersenne Twister ⇒ parallélisable
5. Counter Based `Random123`

   `https://www.deshawresearch.com/resources_random123.html`

   - *satisfy rigorous statistical testing (BigCrush in TestU01),*
   - *vectorize and parallelize well (each generator can produce at least 2^{64} independent streams),*
   - *have long periods (the period of each stream is at least 2^{128}),*
   - *require little or no memory or state,*
   - *and have excellent performance (a few clock cycles per byte of random output)*

depuis 2011 ! `http://dx.doi.org/10.1145/2063384.2063405`

INSTITUT DE PHYSIQUE NUCLÉAIRE
ORSAY

Boucle principale

```
//!>  Start of integration loop
startTime = getticks ();
int ev, evSelected = 0;
double evWeight;
for (ev=0; ev < run->nbrOfEvents; ev++) {

  // Reset Matrix elements
  resetME2 (&ee3p);

  // Event generator
  evWeight = generateRambo (&rambo, outParticles, 3, run->ETotal);
  evWeight = evWeight * run->cstVolume;

  // Sort outgoing photons by energy
  sortPhotons (outParticles);

  // Spinor inner product, scalar product and
  // center-of-mass frame angles computation
  computeSpinorProducts (&ee3p.spinor, ee3p.momenta); // intermediate computation
  computeScalarProducts (&ee3p);

  if (selectEvent (&pParameters, &ee3p)) {
    computeME2 (&ee3p, &pParameters, run->ETotal);
    updateStatistics (&statistics, &pParameters, &ee3p, evWeight);
    evSelected++;
  }
}
```

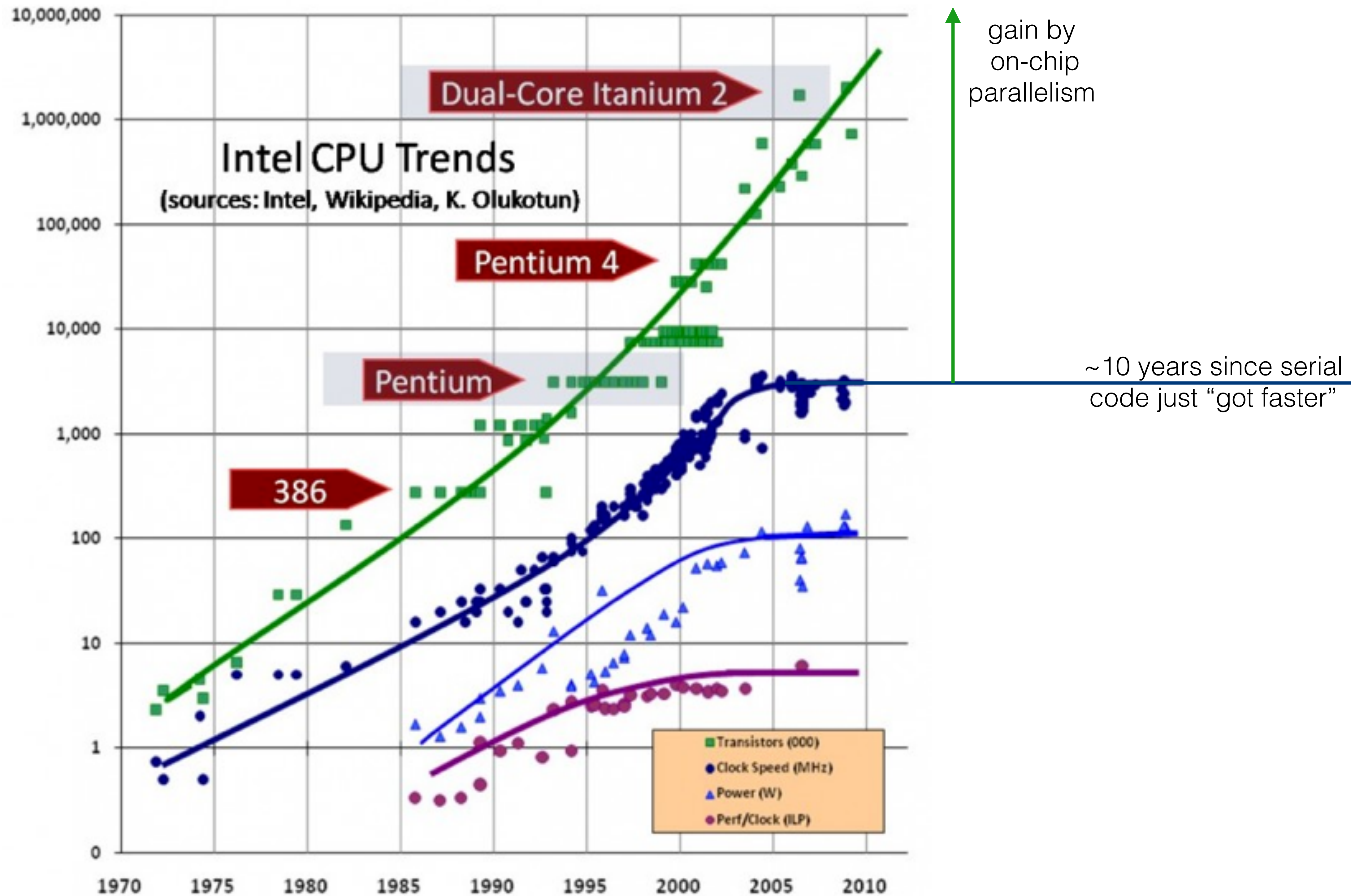
INSTITUT DE PHYSIQUE NUCLÉAIRE
ORSAY

Boucle OpenMPifiée

```
  //!>  Start of integration loop
  startTime = getticks ();
  int ev, evSelected = 0;
  double evWeight;
#pragma omp for reduction…(+:)
  for (ev=0; ev < run->nbrOfEvents; ev++) {

    // Reset Matrix elements
    resetME2 (&ee3p);

    // Event generator
    evWeight = generateRambo (&rambo, outParticles, 3, run->ETotal);
    evWeight = evWeight * run->cstVolume;

    // Sort outgoing photons by energy
    sortPhotons (outParticles);

    // Spinor inner product, scalar product and
    // center-of-mass frame angles computation
    computeSpinorProducts (&ee3p.spinor, ee3p.momenta); // intermediate computation
    computeScalarProducts (&ee3p);

    if (selectEvent (&pParameters, &ee3p)) {
      computeME2 (&ee3p, &pParameters, run->ETotal);
      updateStatistics (&statistics, &pParameters, &ee3p, evWeight);
      evSelected++;
    }
  }
```

Moore’s Law: transition to many-core

There is no escaping parallel computing any more even on a laptop.



http://wccftech.com/moores-law-will-be-dead-2020-claim-experts-fab-process-cpug-pus-7-nm-5-nm/

1998-2014 : The memory Wall

Year	Proc	GFlops	GHz	Cores	SIMD	GB/s
1998	Dec α	0.750	0.375	1		0.6
2014	Intel Xeon	500	2.6	2×14	AVX.2	68
		$\times 1333$	$\times 7$	$\times 28$	$\times 4/8$	$\times 100$

Résumé

Depuis 15 ans :

- Accélération des super-ordinateurs :$\times$1000
- Accélération des nœuds :$\times$1000
- Accélération par la fréquence :$\times$10
- Accélération par le // SMP :$\times$10
- Accélération par la vectorisation :$\times$10
- Augmentation de la bande passante :$\times$100

Moving data

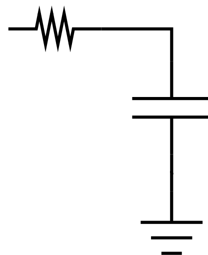
Data is moved through wires.

Wires behave like an RC circuit.

Trade-off:

- Longer response time (“latency”)
- Higher current (more power)

Physics says: *Communication is slow, power-hungry, or both.*



INSTITUT DE PHYSIQUE NUCLÉAIRE
ORSAY

Pour celui qui a un marteau,....

... tous les problèmes ressemblent à un clou

Non seulement il faut paralléliser, mais encore faut-il que **l'intensité arithmétique** du problème s'y prête pour ne pas être **memory bound** : ne pas se laisser aveugler par les benchmarks. Les produits scalaires et convolutions sont limitées, mais les **produit, inversions et diagonalisation de matrices** sont particulièrement bien conçus pour exploiter les architectures vectorielles.

Pour voir les problèmes à travers l'œil de qui a un GPU, il faut identifier des produits de matrices.
Les simulations de théoricien seront potentiellement bien adaptées : peu de données...

- ... initiales à transférer en mémoire GPU
- ... finales (agrégées) à transférer de la mémoire GPU

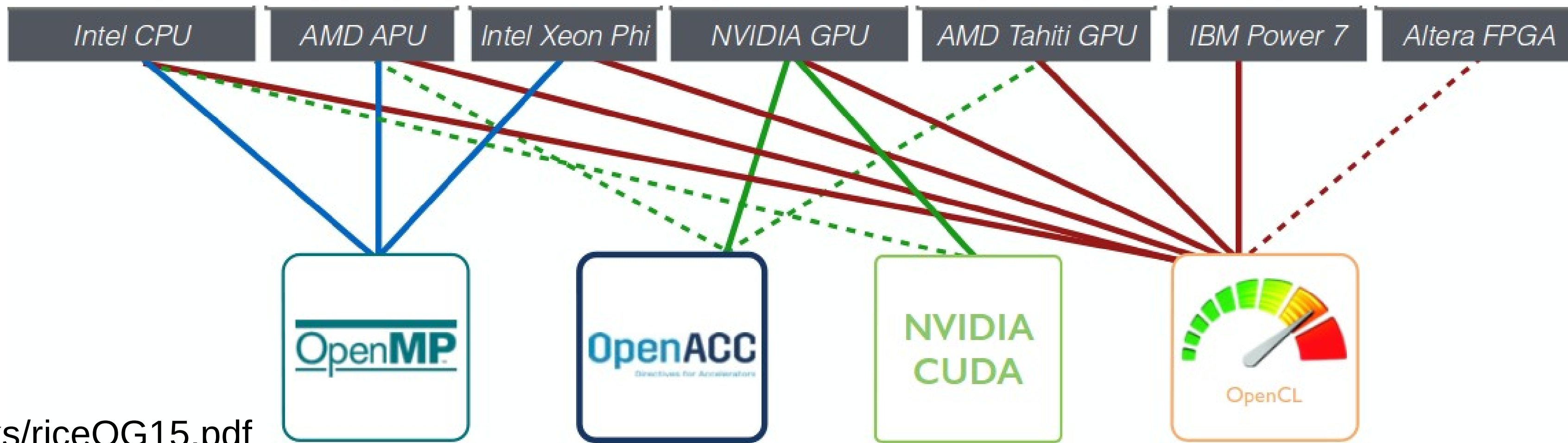
Résumé

Pour vectoriser efficacement il faut :

- Trouver du parallélisme SIMD
- Aligner les ballons (données) en face des joueurs (unités SIMD).
- Avoir un bonne localité des données spatiale et temporelle (intensité arithmétique).
- Choisir son outil :
  - assembleur (+ anti-depresseurs)
  - intrinsics (+ aspirine)
  - compilateur performant (+ boucles triviales)
  - bibliothèque adaptée (en C++ Eigen, boost : :simd,..)

EDF

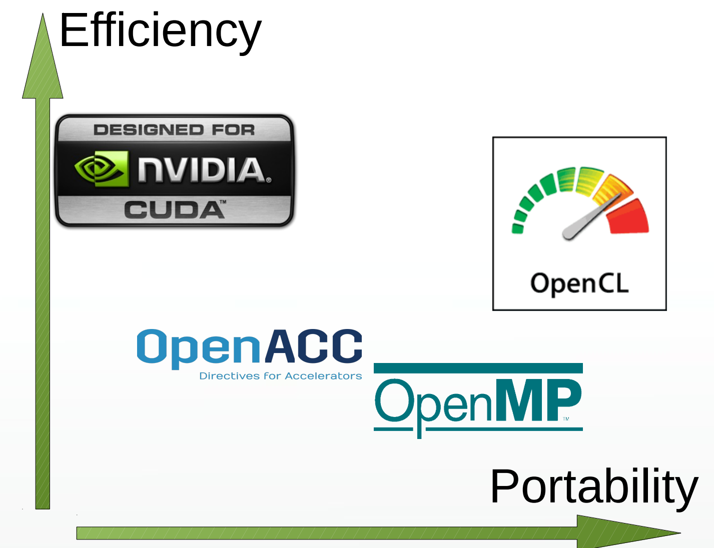
Introduction
Choosing a SW technology



http://libocca.org/talks/riceOG15.pdf

Key points

- Efficient & Portable application (changing technologies)
- Standard free (cheap) technology
- Easy development (//ism not easy task)
- Development tools (debugger, performance analysis)



The C++ Standard

- C++11 introduced lower level abstractions
  - std::thread, std::mutex, std::future, etc.
  - Fairly limited (low level), more is needed
  - C++ needs stronger support for higher-level parallelism
- New standard: C++17:
  - Parallel versions of STL algorithms (P0024R2)
- Several proposals to the Standardization Committee are accepted or under consideration
  - Technical Specification: Concurrency (N4577)
  - Other proposals: Coroutines (P0057R2), task blocks (N4411), executors (P0058R1)

HPX 101 – API Overview

R f(p...)	Synchronous (returns R)	Asynchronous (returns future<R>)	Fire & Forget (returns void)
Functions (direct)	f(p...) C++	async(f, p...)	apply(f, p...)
Functions (lazy)	bind(f, p...)(...)	async(bind(f, p...), ...) C++ Standard Library	apply(bind(f, p...), ...)
Actions (direct)	HPX_ACTION(f, a) a()(id, p...)	HPX_ACTION(f, a) async(a(), id, p...)	HPX_ACTION(f, a) apply(a(), id, p...)
Actions (lazy)	HPX_ACTION(f, a) bind(a(), id, p...) (...)	HPX_ACTION(f, a) async(bind(a(), id, p...), ...)	HPX_ACTION(f, a) apply(bind(a(), id, p...), ...) HPX

In Addition: `dataflow(func, f1, f2);`

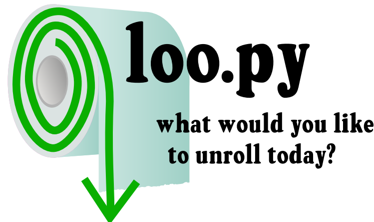
Setting the Stage

Idea:

- Start with math-y statement of the operation
- “Push a few buttons” (transformations) to optimize for the target device
- Strongly separate these two parts

Philosophy:

- Avoid “intelligence”
- User can assume partial responsibility for correctness
- Embedding in Python provides generation/transform flexibility



INSTITUT DE PHYSIQUE NUCLÉAIRE ORSAY

Conclusion

+ Performance
  - `OpenMP` : × 27 sur 32 cœurs
  - `OpenCL + MPI` : × 148 sur 3 nodes GPU

– Obfuscation :
  `OpenMP` : 1 kSLOC
  `C99` : 1,2+1,2 kSLOC
  `OpenCL + CPU` : 2,5 kSLOC
  `HPX + CPU` : 2,5 kSLOC
  `HPX + OpenCL` : 6,7 kSLOC
  `HPX` : grosse bibliothèque (lourdeur proto)

± besoin de couches d'abstraction supplémentaire

- pétrissage du code initial :
  - intérêt du fonctionnel (*a minima*, purifier les fonctions)
  - changer les générateurs aléatoires. (`Random123`)
  - réexprimer certains problèmes ⇒ produits de matrice.

! recourir aux bibliothèques autant que possible