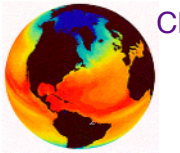
Présentation de l'écosystème HPC français et européen et Retour d'expérience cellule de veille technologique

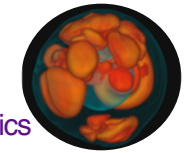
G. Hautreux (GENCI)
gabriel.hautreux@genci.fr

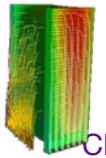
CONTEXTE

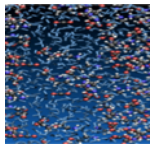
Le calcul haute performance, un outil stratégique

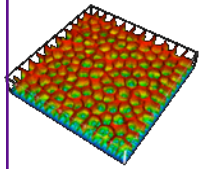
Science

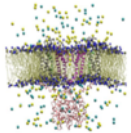

Climate


Astrophysics


Energy

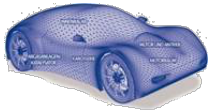

Chemistry

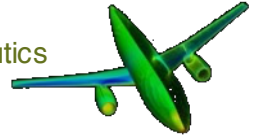

Materials

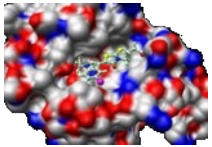

Life Sciences

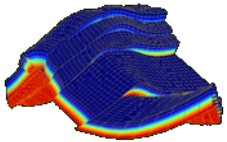
Social sciences & Humanities

Innovation


Automotive

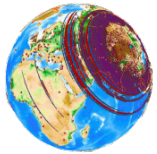

Aeronautics


Pharmacology

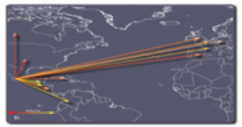

Oil & Gas exploration

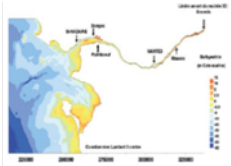

Personalised medicine

Support à la décision

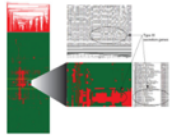

Natural risks

Biological and epidemiological risks




Impact of industrial activities

Security



MISSIONS DE GENCI

2000-2004
France: retard
en HPC

2007
Creation de **GENCI**
pour rattraper le retard

2017
France compétitive,
Europe du HPC

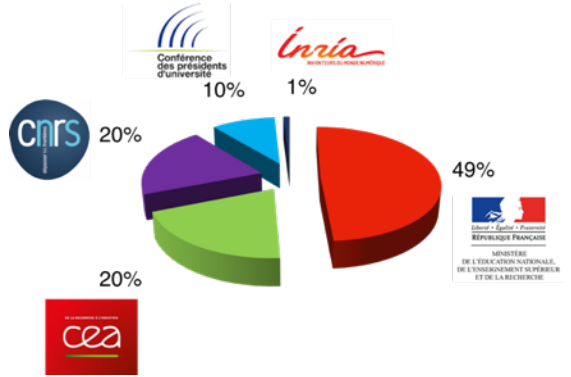
En charge du HPC et du stockage national

Equipement des 3 centres nationaux (TGCC, IDRIS, CINES)

↳ **Dynamique nationale**
↳ **Etendue aux régions**

"Civil society"
5 shareholders
2007-2016 budget = €30M
2017 budget = €30M

EQUIP@MESO



Participation à l'Europe du HPC

GENCI représente la France dans PRACE
↳ **Curie Tier0 européen (80%)**
↳ **Visibilité en Europe**

↳ **Dynamque européenne**
- **Un réseau de 7 supercalculateurs dont Curie 70Pflop/s en 2017**

Promotion du HPC

Dans la recherche et l'industrie
Actions spécifiques sur les PMEs

↳ **Democratisation du HPC pour l'industrie**

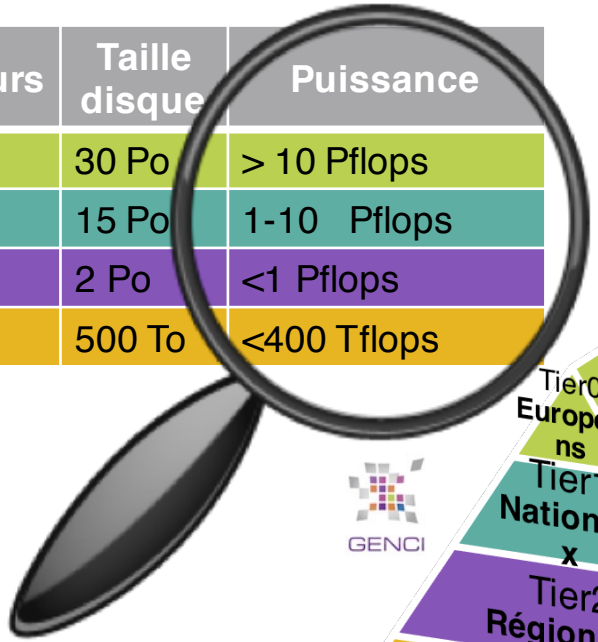
- GENCI et ses partenaires Equip@meso partners
- Sur la base de l'initiative Bpifrance/GENCI/Inria
  = Proof of concept

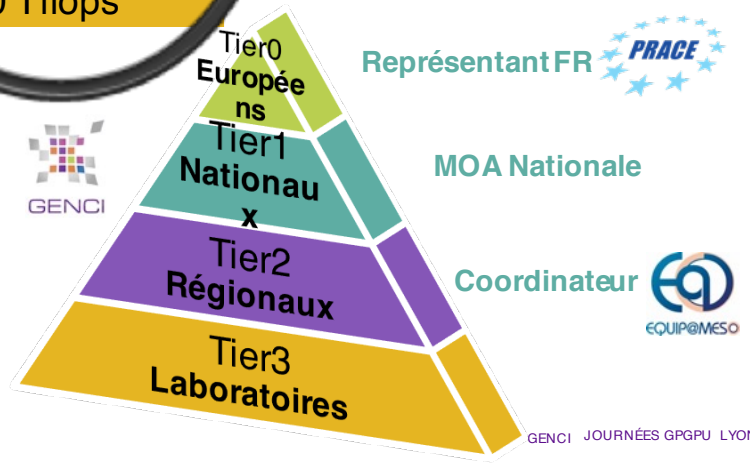
LA PYRAMIDE

Répartition des moyens dans les Tiers

Quels moyens, pour quels projets ?

Tier	Nbre cœurs	Taille disque	Puissance
0	>120 000	30 Po	> 10 Pflops
1	80 000	15 Po	1-10 Pflops
2	10 000	2 Po	<1 Pflops
3	1 000	500 To	<400 Tflops





GENCI

GENCI

Moyens de calcul nationaux pour la production

❑ 3 centres nationaux (CINES, TGCC, IDRIS)

Occigen depuis 2015

Montpellier

Curie (depuis 2012 : 20 % pour le DARI)

Bruyères-le-Châtel

6,8 Pflop/s

Ada et **Turing** (depuis 2013)

Orsay

Irene (S2 2018 50% pour PRACE)

BULL Sequana de 9 Pflops

- 6 cellules Skylake 24 c : 79 488 c
- 3 cellules KNL 68 c, 45 288 c
- 400 To de mémoire,
- Débit disque 500 Go/s.
- Upgrade à 20 Pflops en 2020

❑ Budget pluriannuel sur 10 ans de 30 M€/an*, pour assurer le renouvellement :

- 2018 TGCC / 2020 extension
- 2019 IDRIS / 2021 extension
- 2021 CINES / 2023 extension

→ Une évolution tous les 10 mois

** 28 M€ en réalité (mise en réserve de précaution), majorité du fonctionnent encore assuré par les centres*
→ 50% Capex / 50% Opex

Partnership for Advanced Computing in Europe

PRACE is an international not-for-profit association under Belgian law, with its seat in Brussels.

PRACE counts 25 members and 2 observers.

The **PRACE** Hosting Members are France, Germany, Italy, Spain, and Switzerland.

PRACE is governed by the **PRACE** Council in which each member has a seat. The daily management of the association is delegated to the Board of Directors.

PRACE is funded by its members as well as through a series of implementation projects supported by the European Commission.

PRACE | services for science and industry

Towards End-Users

Access

Tier-0 systems (open R&D)
- Project Access
  1-3 years
- Preparatory Access
  Type A, B, C, D

Tier-1 systems (open R&D)
- DECI Programme

Support

Application Enabling & Support
- Preparatory access
- SHAPE
- HLST support

Training
- Training Portal http://www.training.prace-ri.eu/PATC, PTC
- Seasonal Schools & on demand
- International HPC Summer School
- MOOC

- Code Vault
- Best Practice Guides
- White Papers

Communication, Dissemination, Outreach
- Website
- Public Relations
- Scientific Communication
- Summer of HPC

Events
- PRACEdays
- SC, ISC, ICT, ICRI, DI4R, …

Operation & Coordination of the common PRACE Operational Services
- Service Catalogue
- PRACE MD-VPN network
- Security

HPC Commissioning & Prototyping
- Technology Watch, PCP
- Infrastructure WS
- Best Practices
- UEABS

Towards PRACE Partners

PRACE | services for science and industry

March 2018

- 26 Mar - 28 Mar High Performance Molecular Dynamics@CINECA
- 26 Mar - 29 Mar Advanced Topics in High Performance Computing
- 26 Mar - 28 Mar Advanced Fortran Programming @ CSC
- 22 Mar - 23 Mar Parallel I/O & Libraries @ VSB
- 22 Mar - 23 Mar Petaflop System Administration; Marenostrum 4 @ BSC
- 20 Mar - 21 Mar Efficient Parallel IO on ARCHER @ EPCC at Cambridge
- 15 Mar - 16 Mar Introduction to Parallel Programming with HPX @ HLRS
- 14 Mar - 15 Mar Simulation Environments for Life Sciences @ BSC
- 13 Mar - 16 Mar Spring School in Computational Chemistry 2018 @ CSC
- 12 Mar - 14 Mar Parallel I/O and Portable Data Formats @ JSC
- 12 Mar - 13 Mar OpenMP GPU Directives for Parallel Accelerated Supercomputers - an alternative to CUDA from Cray perspective @ HLRS
- 08 Mar Hands-on Porting and Optimisation Workshop: Making the most of ARCHER @ EPCC at University of Oxford

PRACE | Tier-0 Systems

MareNostrum: IBM
BSC, Barcelona, Spain
#13 Top 500

CURIE: Bull Bullx
GENCI/CEA, Bruyères-le-Châtel, France
#85 Top 500

Piz Daint: Cray XC50
CSCS, Lugano, Switzerland
#3 Top 500

MARCONI: Lenovo
CINECA, Bologna, Italy
#14 Top 500

JUQUEEN: IBM BlueGene/Q
GAUSS @ FZJ, Jülich, Germany
#21 Top 500

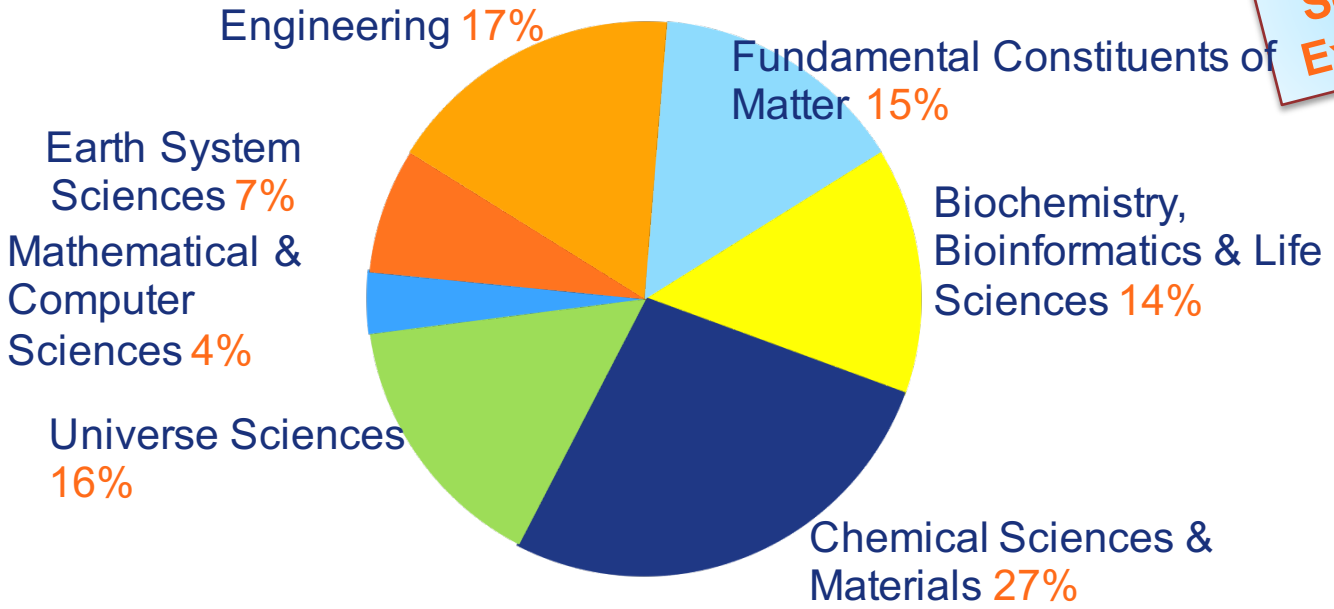
SuperMUC: IBM
GAUSS @ LRZ, Garching, Germany
#40 Top 500

Hazel Hen: Cray
GAUSS/HLRS, Stuttgart, Germany
#17 Top 500

7 Tier0 systems
68 PFlop/s end 2017

PRACE | support to science

Criterion: Scientific Excellence

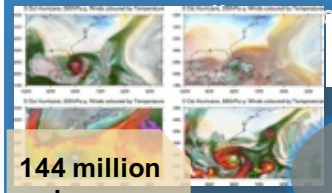


Data up to and including Call 15, % of total core hours awarded (Project Access)

Breakthroughs for the EU research

CLIMATE : FACING THE TEMPEST

A three-year advance in developing



National Centre for Atmospheric Science

Team: **Prof. Pier Luigi Vidale** (NCAS-Climate, Dept of Meteorology, Univ. of Reading and UK Met Office, Exeter, UK)

144 million hours on Hermit (Germany)

ASSESS THE IMPACT OF BIG EARTHQUAKES

3D full-wave tomographic IMAGINg of the Entire ITalian lithosphere

Use of SPECFEM3D

Team: **Prof. D. Komatitisch (CNRS) with researchers from INGV**

cnrs INGV

40 million hours on Curie (France)

QCDpQED project U. Wuppertal/CNRS

Nuclear physics: New outlooks

ueen@Jülich) + **GENCI** (Turing@Idris) ressources = 160 million core hours in 2014

- Confirmation of the mass difference between neutrons and protons, necessary to the stability of nuclei
  - **1st reliable calculation**
  - A step towards **10x** more precise calculations
  - Article in Science, commented in Nature by **Franck Wilczek, Nobel Prize for physics in 2004**

160 million hours on Juqueen (Germany)

Breakthroughs for EU industry

IMPROVING CAR SAFETY

A world premiere in the automotive industry

→ Performing more accurate crash optimization

- **42 million hours** on Curie (France)
- Assess new large scale optimization methods: 200 parameters, 20 million finite elements
- Anticipate new security rules (EuroNCAP6)
- Impossible with internal Renault R&D HPC resources

DEVELOPING NEW DRUGS

Molecular Dynamics Simulations for Computing Kinetic Constants for Drug Discovery

- Improve the process of drug discovery by taking into account for the first time kinetic properties of a drug and a candidate protein
- International collaboration K4DD (Kinetics for Drug Discovery) with NAMD and Gromacs
- **11 million core hours** on Fermi (CINECA)

SANOFI

IMPROVING SHIP SURVIVABILITY UNDER WAVE IMPACT AND AQUAPLANING FOR AUTOMOTIVE

→ Major step: 32k cores simulations – Increased EU visibility and competitiveness for a SME

5 million core hours on Hermit (Germany)

8.2 million hours on CURIE (France)

HydrOcean

PRACE | achievements

- 570 scientific projects enabled
- 16 000 000 000 (thousand million) core hours awarded since 2010
- Of which 63% led by another PI nationality than the HM
- R&D access to industrial users with >50 companies supported
- >10 000 people trained by 6 PRACE Advanced Training Centers, and other events
- 70 Petaflops of peak performance on 7 world-class systems
- 25 PRACE members, including 5 Hosting Members (France, Germany, Italy, Spain and Switzerland)

ACCÈS AUX RESSOURCES GENCI

Conditions d'éligibilité: qui peut y accéder ?

❑ Tous les chercheurs **académiques** et **industriels**

❑ **3 modes d'accès** possibles

1. Recherche ouverte (88%)
   - Accès **gratuit**, sélection sur critères d'**excellence scientifique** (68% Nat+20% Eur)
   - Un **processus unique** pour candidater sur les 3 centres de calcul nationaux
     - Environ 600 projets / an pour près de 2 800 utilisateurs
     - Depuis 2010, plus de 4 500 dossiers expertisés

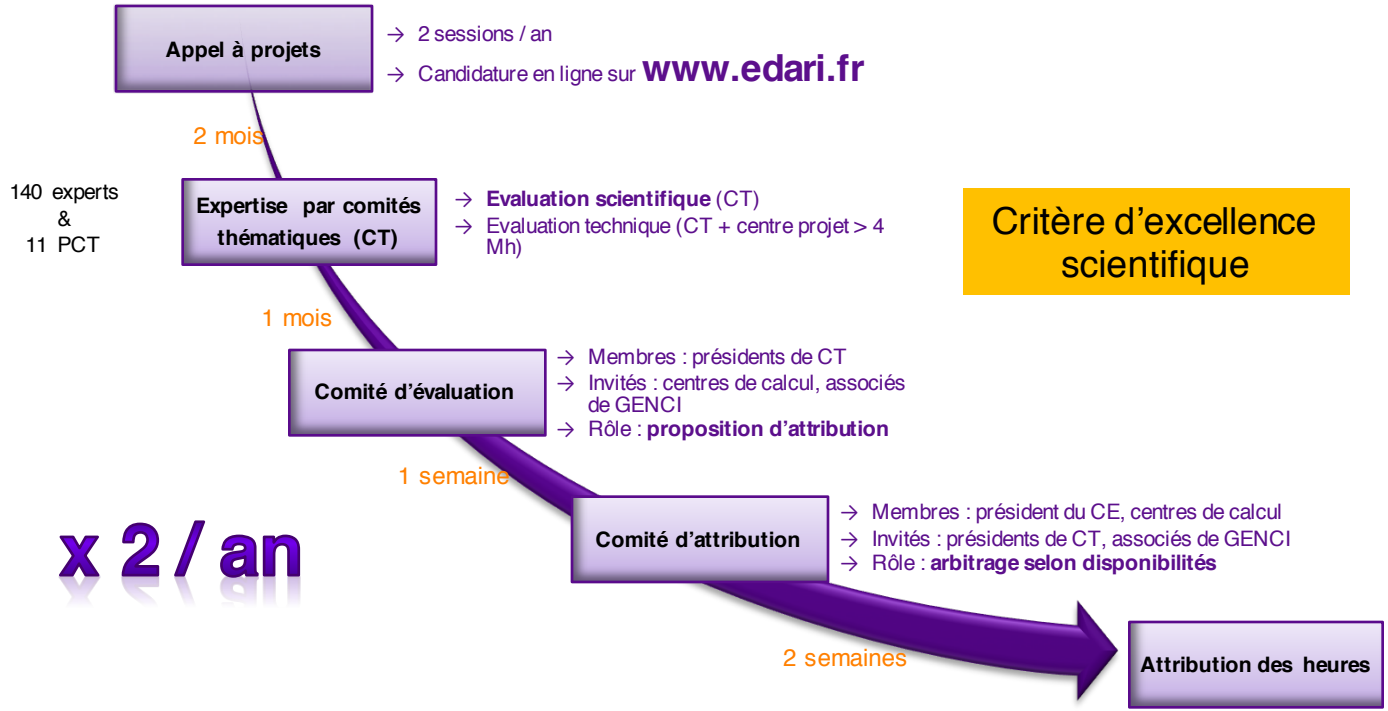
   →**Obligation de publication**

2. Accès communautaire ou stratégique (12%)
   - Accès pluriannuels (ex CMIP6 8%, CEA 4%, IFPEN >1%)

3. Accès ponctuel pour preuves de concept / projets stratégiques
   - Durée limitée dans le temps (one shoot), pas de publication résultats

ACCÈS AUX RESSOURCES GENCI

Attribution d'heures pour le national



Procédure très similaire:
https://prace-peer-review.cines.fr

+ Accès préparatoires 60/an

PROSPECTIVE (CELLULE DE VIELLE TECHNOLOGIQUE)

Moyens de calcul prototype (Puissance d'un Tier2)

- Anticiper l'arrivée des futures architectures …(pré)Exascale
  - Tester les machines en « avant première »
  - Préparer les communautés scientifiques nationales (accès et workshops)
  - Avec 20 experts issus des partenaires de GENCI

CINES

- Plateforme Frioul
  - 48 nœuds Intel KNL 7250 à 68c – IB-EDR **146 Tflop/s**

IDRIS

- Plateforme Ouessant
  - 12 nœuds OpenPOWER Minksy » **245 Tflop/s**
    de 2 x processeurs Power8+ NVLink + 4x NVIDIA P100

>3000 c

Accessibles à tous via AP sur: www.edari.fr

48 P100

TECHNOLOGICAL WATCH GROUP

Led by GENCI and its partners

- **Goals**:
  - anticipate upcoming (pre) exascale architectures
  - deploy prototypes and prepare our users
  - organise code modernization
  - share and mutualise expertise
    - Preserve legacy codes by using standards – **OpenMP**

FRIOUL

KNL based prototype

- Atos Sequana cell @ CINES, Montpellier (France)
  - 2 partitions in 48 Intel Xeon Phi 7250 nodes, 146 Tflops peak
    - 24 Quad+Cache nodes
    - 24 Quad+Flat nodes
  - EDR Interconnect
  - Quadrant mode used for both partitions
    - Poor results with other modes

- Software stack
  - Intel compiler
  - Intel Vtune
  - Intel Advisor

- High level support
  - Thanks to Atos and Intel teams

KNL ARCHITECTURE

Overview

ISA

Intel® Xeon® Processor Binary-Compatible (w/Broadwell)

On-package memory

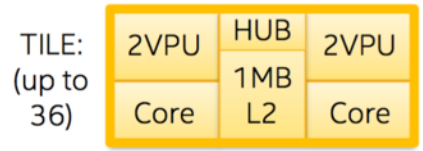
Up to 16GB, ~500 GB/s STREAM at launch

Platform Memory

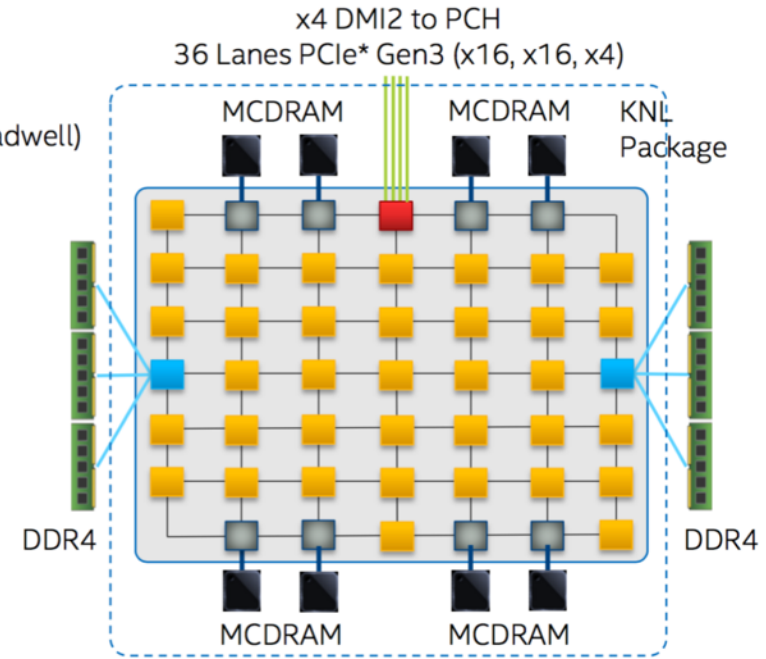
Up to 384GB (6ch DDR4-2400 MHz)

Fixed Bottlenecks

- ✓ 2D Mesh Architecture
- ✓ Out-of-Order Cores
- ✓ 3x single-thread vs. KNC



Enhanced Intel® Atom™ cores based on Silvermont™ Microarchitecture



Tile | EDC (embedded DRAM controller) | IMC (integrated memory controller) | IIO (integrated I/O controller)

OUESSANT

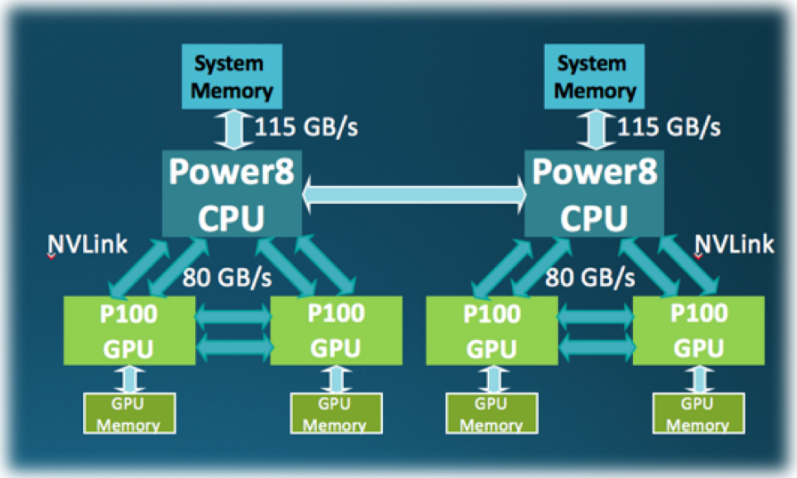
OpenPOWER based prototype

- OpenPOWER platform @ IDRIS, Orsay (France)
  - 12 IBM System S822LC “Minsky”, >250 Tflops peak
    - 2 IBM Power8 10-core processors @ 4.2GHz
    - 128GB of memory per node
    - 2 Nvidia P100 GPUs per socket
    - Connection socket <-> GPU with NVLink 1.0 (80GB/s)
  - IB EDR Interconnect
- Software stack
  - Multiple compilers
    - PGI (main target OpenACC)
    - IBM XL (main target OpenMP)
    - LLVM (fortran troubles in 2016)
  - Power AI within Docker
- High level support
  - Multiple workshops organised
  - Thanks to IBM and Nvidia teams

OUESSANT

Ouessant architecture

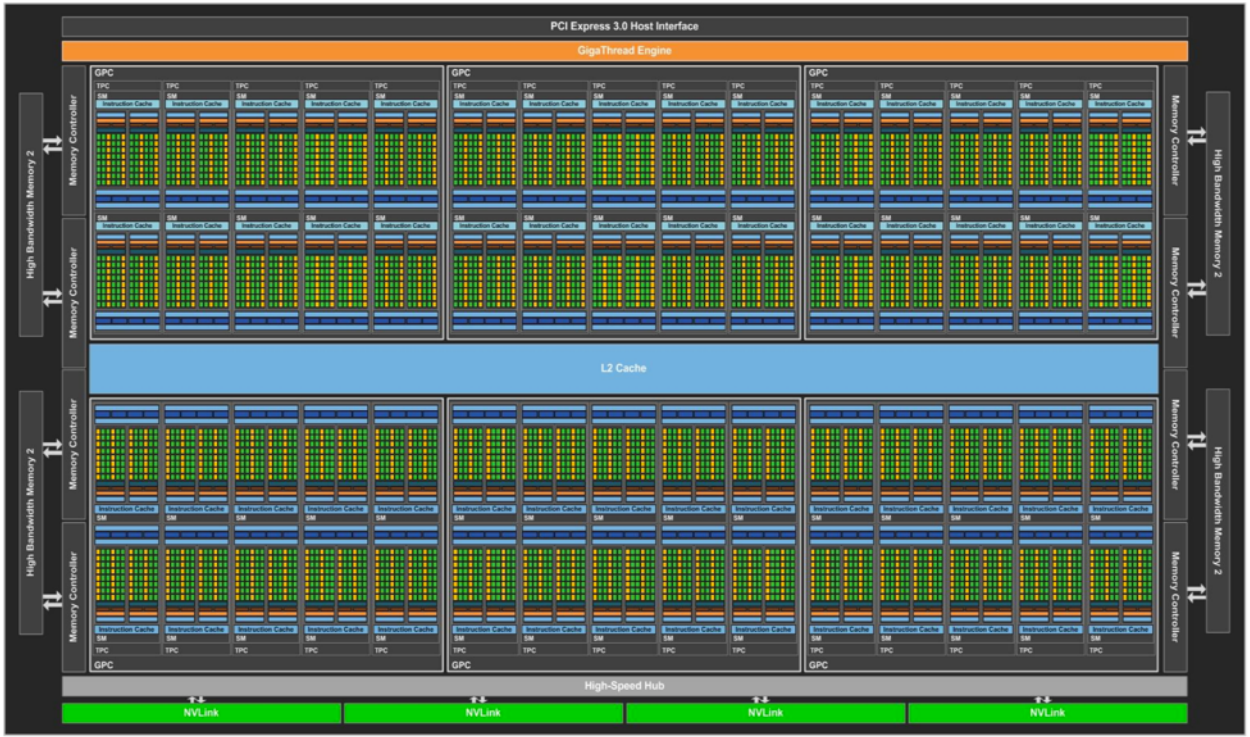
- OpenPOWER platform
  - "Minsky" nodes (21 Tflops)
  - 20 CPU cores/node
  - 4 GPUs/node
  - 12 nodes interconnected (IB)
  - High performance filesystem
  - Storage available



Minsky node

NVIDIA P100 ARCHITECTURE

Overview



SOFTWARE ENVIRONMENT

Architectures usability

- One question: how to write code for those architectures?

	Paradigm	Standard x86 processors	Intel Xeon Phi (many cores as KNL)	NVIDIA GPUs
	MPI	Yes	Yes	No
#1	OpenMP	Yes	Yes	Yes
#2	OpenACC	Yes	Yes	Yes
	CUDA	No	No	Yes

RELEVANT SET OF APPLICATIONS

Represent French research community

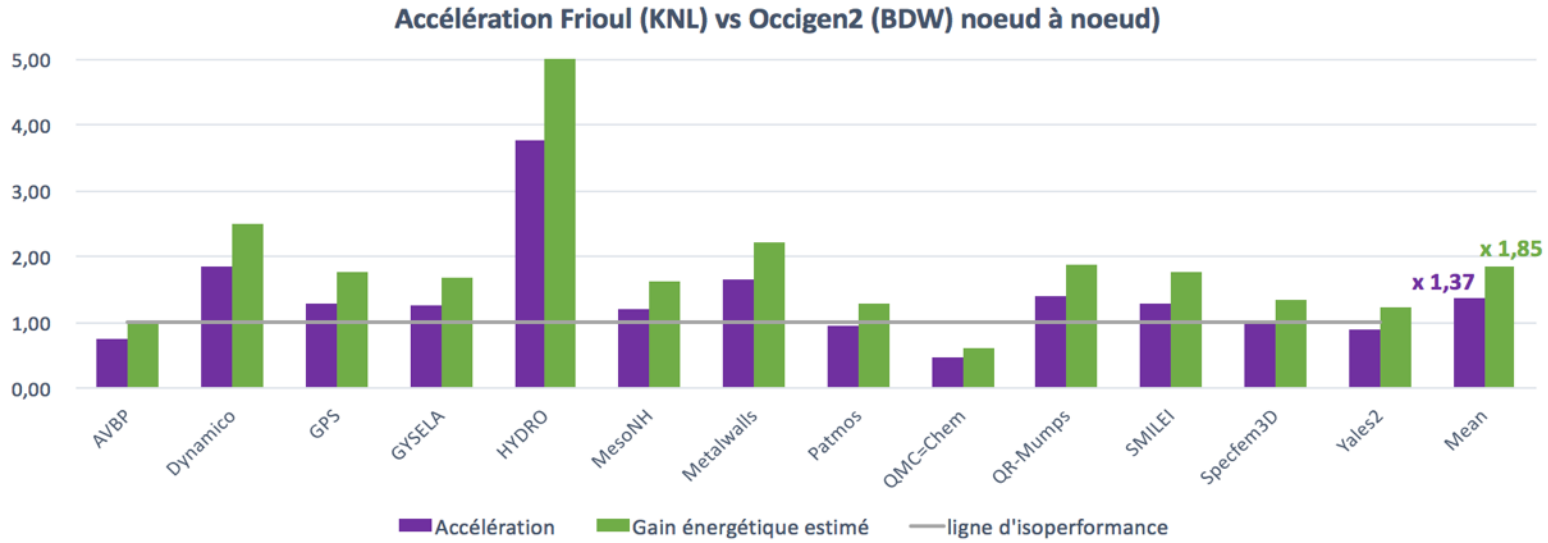
- 18 « real » applications
  - 2 GPU focused (RAMSES end EMMA)
  - 1 OpenCL (CMS-MEM)
    - No official support at the moment
  - 15 « standard applications » coming from various scientific and industrial domains
- 4 withdrawals so far

→ **Work performed on 14 applications**

PERFORMANCE SUMMARY

Frioul results

- The overall performance for those applications at the moment:
  - Node to node comparison
  - Energy efficiency is an estimation using TDP ratios

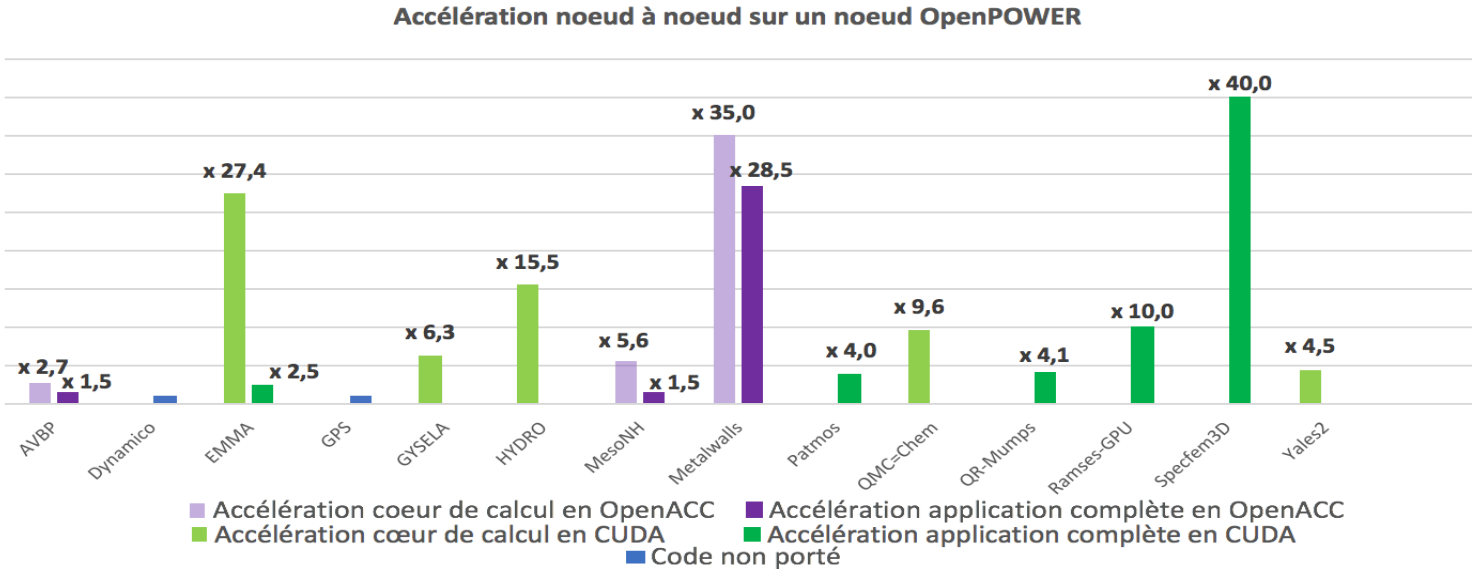


  - Lessons learned:
    - Easy to use (more or less recompile and run, and add some vectorisation)
    - Cache mode is good enough for most cases (and does not need any action from the user)

PERFORMANCE SUMMARY

Ouessant preliminary results

- The overall performance for those applications at the moment:
  - Mainly CUDA and OpenACC
  - No OpenMP results on this graph at the moment



- Impressive speed-up compared to what can been achieved on KNL
  - At the price of a way higher porting effort

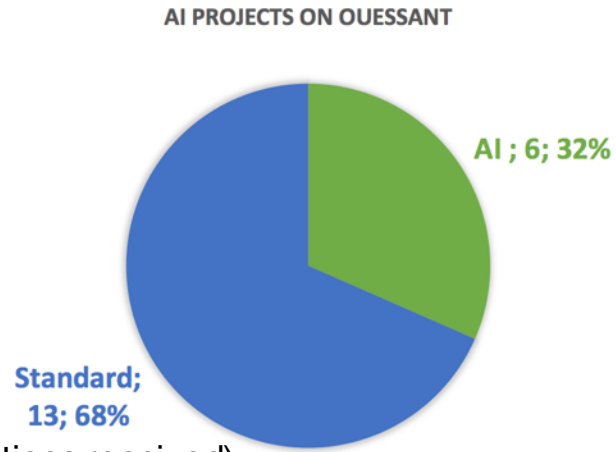
FIRST CONCLUSIONS

Feedback on OpenPower platform

- Power8 processor is easy to use (compile and run)

- Programming models
  - CUDA: very high performances
  - OpenACC: high performances
  - OpenMP 4.5: no global feedback
- Compilers
  - PGI working efficiently (for both Power8 and GPUs with OpenACC)
  - IBM XL is more and more OpenMP for GPU aware

- First results are very promising
  - Opening of the platform to the full French community in April 2017
  - more applications and new focus on AI applications (50% of applications received)

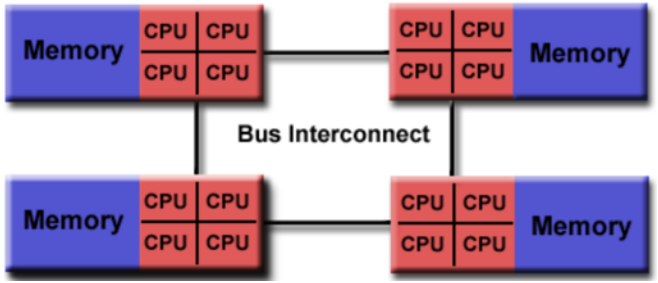


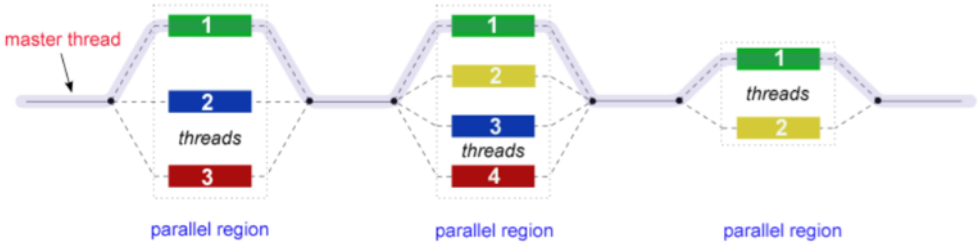
One example:
Porting Metalwalls
on Intel KNL and OpenPOWER platforms

WHAT IS OPENMP ?

Multi-threading standard

- C/C++ and Fortran API
  - portable, scalable model
  - shared memory parallel applications
    - UMA and NUMA architectures
- Based on :
  - compiler directives
  - Environment variables
  - OpenMP routines
- Uses the thread fork/join model





OPENMP IMPLEMENTATION

How to write efficient code?

- **Writing efficient code is a prerequisite for OpenMP**
  - Try to avoid if statements
  - Try to work on large loops
    - Avoid AoS, prefer SoA
  - Work on you vectorisation (and then use SIMD pragmas)
  - Etc… (see: optimisation support CINES)
- **Make sure parallelizing worth it**
  - Compute your theoretical speed-up
    - Amdahl's Law
  - Try to compute the porting effort
  - Try to identify patterns to reduce this effort
- **Implement using the standard**
  - http://www.openmp.org/wp-content/uploads/OpenMP-4.5-1115-CPP-web.pdf
  - https://computing.llnl.gov/tutorials/openMP

OPENMP EXEMPLE

A dot product (source llnl tutorial)

```
      PROGRAM DOT_PRODUCT

      INTEGER N, CHUNKSIZE, CHUNK, I
      PARAMETER (N=100)
      PARAMETER (CHUNKSIZE=10)
      REAL A(N), B(N), RESULT

!     Some initializations
      DO I = 1, N
        A(I) = I * 1.0
        B(I) = I * 2.0
      ENDDO
      RESULT= 0.0
      CHUNK = CHUNKSIZE

!$OMP  PARALLEL DO
!$OMP& DEFAULT(SHARED) PRIVATE(I)
!$OMP& SCHEDULE(STATIC,CHUNK)
!$OMP& REDUCTION(+:RESULT)

      DO I = 1, N
        RESULT = RESULT + (A(I) * B(I))
      ENDDO

!$OMP  END PARALLEL DO

      PRINT *, 'Final Result= ', RESULT
      END
```

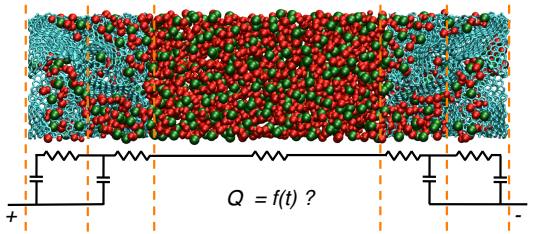
```
#include <omp.h>

main(int argc, char *argv[])  {

int   i, n, chunk;
float a[100], b[100], result;

/* Some initializations */
n = 100;
chunk = 10;
result = 0.0;
for (i=0; i < n; i++) {
  a[i] = i * 1.0;
  b[i] = i * 2.0;
  }

#pragma omp parallel for      \
  default(shared) private(i)  \
  schedule(static,chunk)      \
  reduction(+:result)

  for (i=0; i < n; i++)
    result = result + (a[i] * b[i]);

printf("Final result= %f\n",result);

}
```

MetalWalls for Intel KNL

gabriel.hautreux@genci.fr

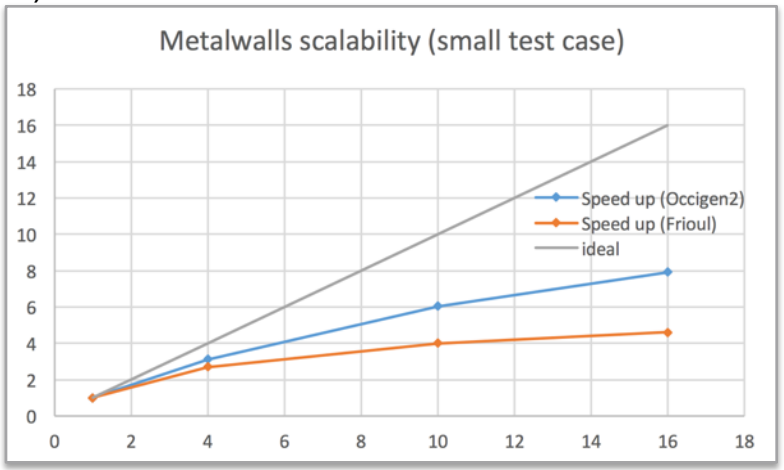
METALWALLS ON INTEL KNL

Molecular dynamic



- Molecular dynamic application
  - Co-developed by “Université Pierre et Marie Curie and Maison de la Simulation” (UPMC and MdS)
- MPI

Speed up node to node	1,07
Speed up 16 vs 16 nodes	0,62
Vectorization (1 node)	5,97
Flat mode speed up	1,00



- Test case
  - Small test case
- Scalability study
  - Frioul scales badly compared to Occigen
    - The test case may be too small, scalability issues also come from lot of MPI messages
      - An OpenMP implementation should be considered

SINGLE NODE PROFILING

Vtune profile using one MPI process

❑Understand the hotspots

Elapsed Time: 1270.862s

CPU Time:	1188.257s
Total Thread Count:	1
Paused Time:	0s

main	100.0%
conjgradwall	44.5%
wallcg	44.5%
cgwallener	44.5%
cgwallrecipe	41.7%
cgwallreale	2.8%
rattle	41.0%
conjgradwall	40.7%
wallcg	40.7%
cgwallener	40.7%
cgwallrecipe	38.1%
cgwallreale	2.6%
ener	0.3%
ener	13.7%
rgdreale	7.3%
sr_energy	3.5%
rgdrecipe	2.9%
[Unknown stack frame(s)]	0.0%
distrib_pairs	0.5%

- Lot of time spent in a few functions
- Good candidate for a quick OpenMP implementation

THEORETICAL SCALABILITY

Amdahl's law

- Focus on 5 functions:
  - Theoretical speed-up is computed for the first ported function
    - The theoretical speed-up is then increased for each other ported function

Function	Time spent in this function	Summed Theo. Speed-up
cgwallrecipE	79,8%	5
rgdrealE	7,3%	7,8
cgwallrealE	5,4%	13,3
Sr_energy	3,5%	25
rgdrecipe	2,9%	91

  - As we can get a very interesting theoretical speed-up, we can go for an OpenMP implementation

OPENMP IMPLEMENTATION

recipE functions

- cgwallrecipE and rgdrecipE
  - Functions have more or less the same structure

```
!inital array for *recipE functions
do ll = kx_beg, kx_end
  do mm = ky_beg(ll), ky_end(ll)
    do nn = kz_beg(ll, mm), kz_end(ll, mm)
      do i=1,num
        !temp array alpha
        alpha(i)+=foo(ll,mm,nn,i)
        beta(i) +=bar(ll,mm,nn,i)
      enddo
      do i=1,num
        !compute global variables
        pot(i)=pot(i)+f(alpha(i)^2)+g(beta(i)^2)
      enddo
    enddo
  enddo
enddo
```

Parallelise inner loop?
No way: huge overhead

Private temporary arrays, then reduction?
Num is huge, overhead too important for duplicating the arrays

Better if we could parallelize on « i » without threads creation overhead

OPENMP IMPLEMENTATION

recipE functions

- cgwallrecipE and rgdrecipE
  - Functions have more or less the same structure

```
!inital array for *recipE functions
do ll = kx_beg, kx_end
  do mm = ky_beg(ll), ky_end(ll)
    do nn = kz_beg(ll, mm), kz_end(ll, mm)
      do i=1,num
        !temp array alpha
        alpha(i)+=foo(ll,mm,nn,i)
        beta(i) +=bar(ll,mm,nn,i)
      enddo
      do i=1,num
        !compute global variables
        pot(i)=pot(i)+f(alpha(i)^2)+g(beta(i)^2)
      enddo
    enddo
  enddo
enddo
```

Cache blocking: helps to parallelize on « i » loop

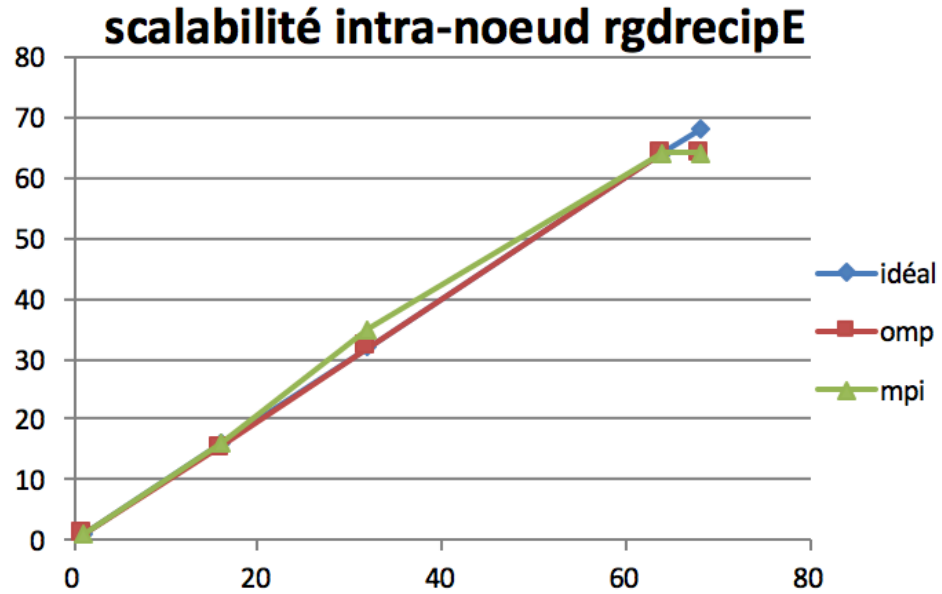
```
!optimized array with cache_blocking
!omp parallel do reduction(+:alpha,beta)
do b = 1, nb_block
  do ll,mm,nn
    do i = iblock_start, iblock_end
    alpha(ll,mm,nn)+=foo(ll,mm,nn,i)
    beta(ll,mm,nn) +=bar(ll,mm,nn,i)
    enddo
  enddo
enddo
!omp parallel do
do b = 1, nb_block
  do ll,mm,nn
    do i = iblock_start, iblock_end
      pot(i)=pot(i)+f(alpha(ll,mm,nn)^2)+g(beta(ll,mm,nn)^2)
    enddo
  enddo
enddo
```

Loop decomposition or pot will not be correctly computed

OPENMP IMPLEMENTATION

recipE functions scalability

- Scalability results: OpenMP vs MPI



OPENMP IMPLEMENTATION

realE functions

- cgwallrealE and rgdrealE
  - Functions have more or less the same structure
  - Sr_energy is also the same

```
!inital array for *realE (and sr_energy..) functions
do j = jbeg_ww2, jend_ww2
  do i = ibeg_ww2(j), iend_ww2(j)
  !check the neighborhood
  compute_distance()
  if(distance>const) CYCLE
  !compute global variables
  pot(i)=...
  elec(i)=...
  ...
  enddo
enddo
MPI_ALL_REDUCE()
```

Parallelise inner loop?
No way: huge overhead

CYCLE: if we parallelize, we will have a load imbalance due to the condition

→ parallelize the outer-loop
→ take care of the load balancing

OPENMP IMPLEMENTATION

realE functions

- cgwallrealE and rgdrealE
  - Functions have more or less the same structure
  - Sr_energy is also the same

```
!inital array for *realE (and sr_energy..) functions
do j = jbeg_ww2, jend_ww2
  do i = ibeg_ww2(j), iend_ww2(j)
  !check the neighborhood
  compute_distance()
  if(distance>const) CYCLE
  !compute global variables
  pot(i)=...
  elec(i)=...
  ...
  enddo
enddo
MPI_ALL_REDUCE()
```

Reduction on pot and elec is a solution to parallelize the outer loop

Dynamic scheduling prevent load imbalance

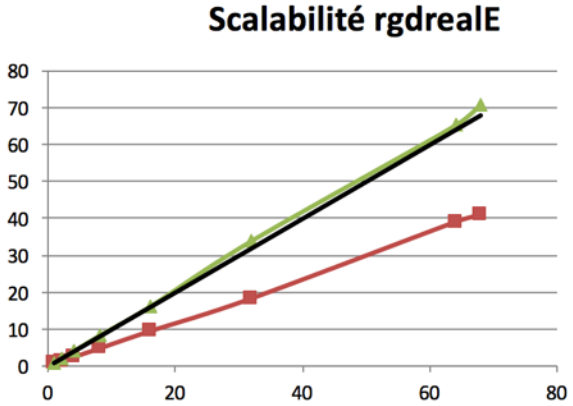
```
!$omp parallel do reduction(+:pot,elec) schedule(dynamic)
do j = jbeg_ww2, jend_ww2
  do i = ibeg_ww2(j), iend_ww2(j)
  !check the neighborhood,
  !dynamic scheduling is used for a better balance
  !between threads right here
  compute_distance()
  if(distance>const) CYCLE
  !compute global variables and reduce
  pot(i)=...
  elec(i)=...
  ...
  enddo
enddo
MPI_ALL_REDUCE()
```

OPENMP IMPLEMENTATION

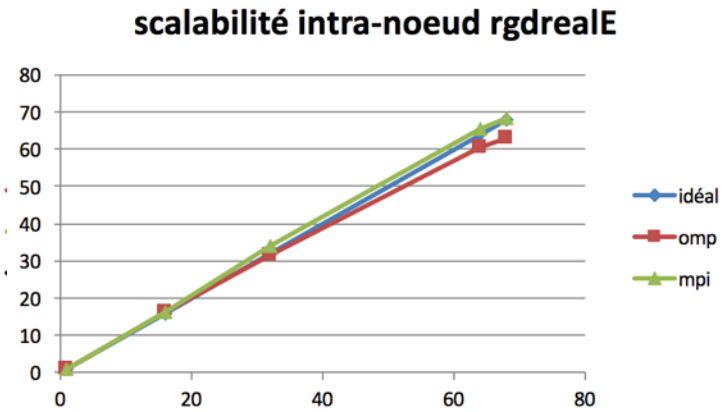
realE functions scalability

- Scalability results: OpenMP vs MPI

Without Dynamic scheduling



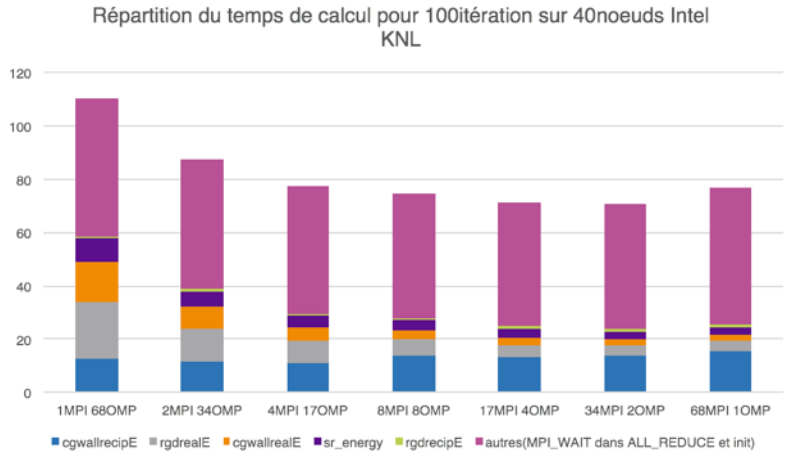
With Dynamic scheduling



OPENMP IMPLEMENTATION

Full application results

- 68MPI vs 68OMP (1 node)
  - OpenMP is 5% faster than MPI
    - Reduction of the communications inside one node
- On 10 nodes
  - About 10% better performances vs full MPI
  - Pink bar is too large
    - Work has to be done on MPI part to scale-up
- Porting effort
  - 2 weeks
- Problem
  - Code imbalance!!
  - Bottleneck identified:
    - Development team is fixing it

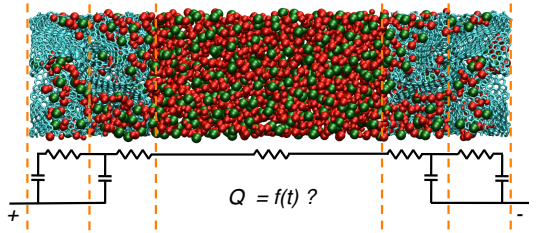


MetalWalls for GPU

gabriel.hautreux@genci.fr

FOCUS ON METALWALLS

First OpenMP results



❑MPI + OpenACC

Abel Marin-Lafleche, Matthieu Haefele (MdS)

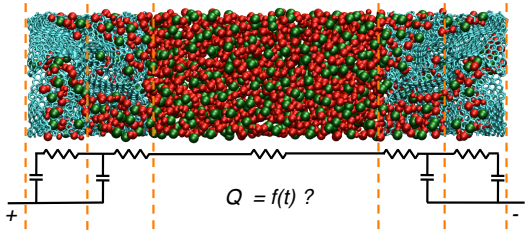
- OpenACC development started in Q1 2017
- 3500 lines of Fortran 90 code (computationnal part)
  - 2 kernels : Energy minimization via conjugate gradient and n-body computations (N^2 algorithm complexity)
- First results availabe after one month
- More or less 90% of the app ported
- Porting effort: 2 months

❑MPI + OpenMP

- Development started in Q3 2017
- First results available after a week (thanks to OpenACC existing implementation)

FOCUS ON METALWALLS

Application study



- 2 main types of functions
  - One takes more than 80% of the computational time
  - 2 main loops, look more or less like the following:

Loop #1

```
!cache_blocking
   do b = 0, nb_chunk
   |  do l, m, n
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   enddo
```

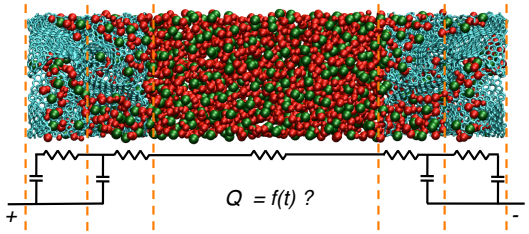
Then →

Loop #2

```
!cache_blocking
   do b = 0, nb_chunk
   |  do l, m, n
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  |  POT(i) += Tmp * BUFFER(n,m,l)
   |  |  |
   |  |  end do
   |  enddo ll, mm, nn
   enddo
```

FOCUS ON METALWALLS

Application study



- Function study
  - l, m, n loop express enough parallelism
  - SIMD is expressed through the inner i loop

Loop #1 (the same apply to loop #2)

```
!cache_blocking
   do b = 0, nb_chunk
   |  do l, m, n
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   enddo
```

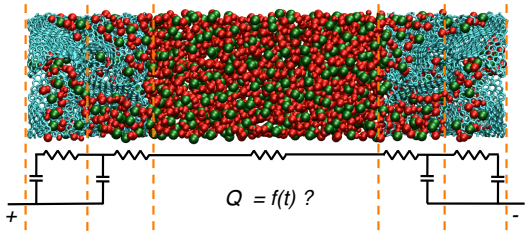
High enough parallelism

SIMD

Conditions for efficient use of the GPU

FOCUS ON METALWALLS

Memory management



- OpenACC pragmas added to both loops

Loop #1

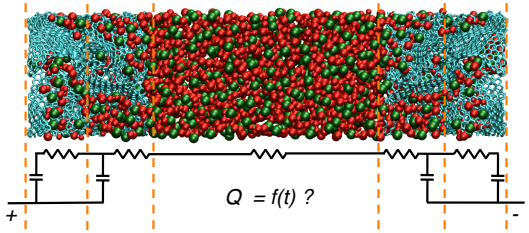
```
!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector reduction(+:Tmp)
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   |  !$acc end parallel
   enddo
```

Loop #2

```
!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  | !$acc atomic update
   |  |  |  POT(i) += Tmp * BUFFER(n,m,l)
   |  |  |
   |  |  end do
   |  enddo ll, mm, nn
   enddo
```

FOCUS ON METALWALLS

OpenACC vs OpenMP implementations



❑Easy to port OpenACC to OpenMP

Loop #1

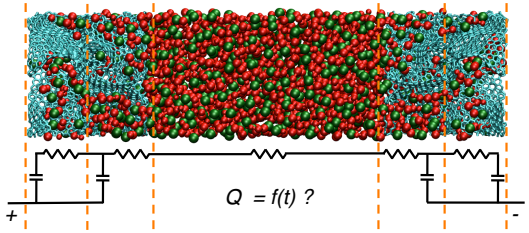
```
!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector reduction(+:Tmp)
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   |  !$acc end parallel
   enddo
```

```
!cache_blocking
   do b = 0, nb_chunk
   |  !$omp target
   |  !$omp teams distribute collapse(3)
   |  do l, m, n
   |  |  !$omp parallel do private(GREEN) reduction(+:Tmp)
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   |  !$omp end target
   enddo
```

- **Private (or firstprivate,…) MUST be used as OpenMP is an explicit model**

FOCUS ON METALWALLS

OpenACC vs OpenMP implementations



- Easy to port OpenACC to OpenMP

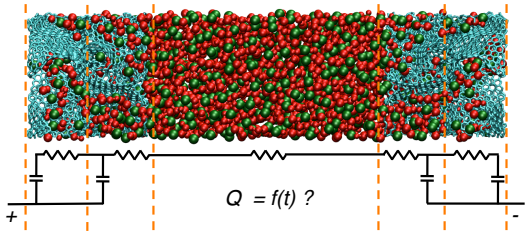
Loop #2

```
!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  | !$acc atomic update
   |  |  |  POT(i) += Tmp * BUFFER(n,m,l)
   |  |  |
   |  |  end do
   |  enddo ll, mm, nn
   enddo
```

```
!cache_blocking
   do b = 0, nb_chunk
   |  !$omp target
   |  !$omp teams distribute collapse(3)
   |  do l, m, n
   |  |  !$omp parallel do private(GREEN)
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  | !$omp atomic update
   |  |  |  POT(i) += Tmp * BUFFER(n,m,l)
   |  |  |
   |  |  end do
   |  enddo ll, mm, nn
   |  !$omp end target
   enddo
```

FOCUS ON METALWALLS

Memory management



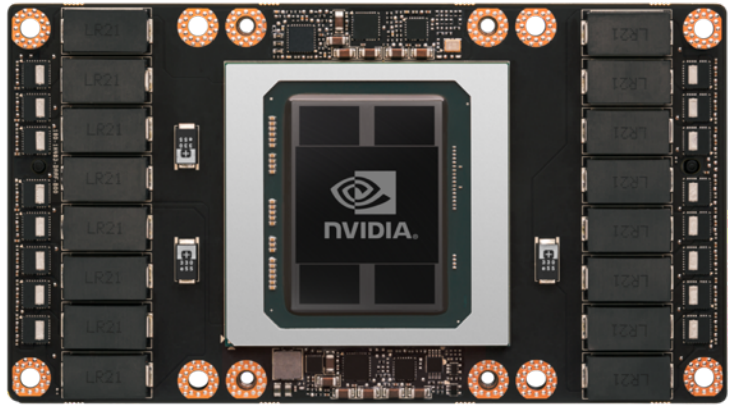
❑Managed memory: how does it work?

```
!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector reduction(+:Tmp)
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   |  !$acc end parallel
   enddo

!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  |  !$acc atomic update
   |  |  |  POT(i) += Tmp * BUFFER(n,m,l)
   |  |  |
   |  |  end do
   |  enddo ll, mm, nn
   enddo
```

What we could expect :

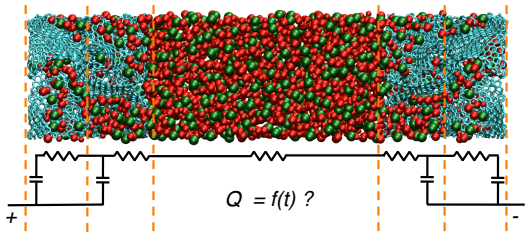
CPU

GPU



FOCUS ON METALWALLS

Memory management

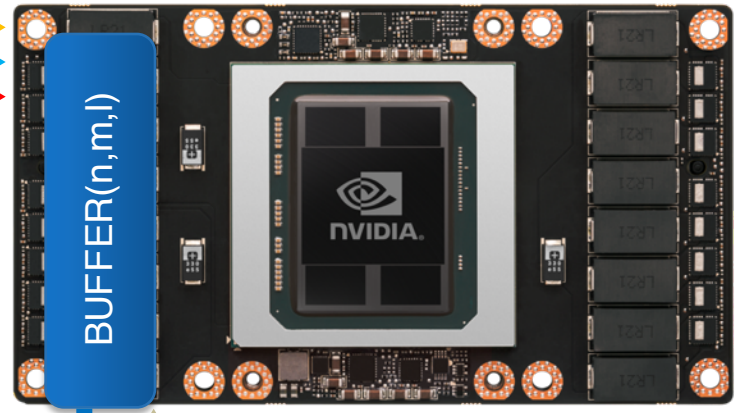


❑Managed memory: how does it work?

```
!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector reduction(+:Tmp)
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   |  !$acc end parallel
   enddo

!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  |  !$acc atomic update
   |  |  |  POT(i) += Tmp * BUFFER(n,m,l)
   |  |  |
   |  |  end do
   |  enddo ll, mm, nn
   enddo
```

What is done :

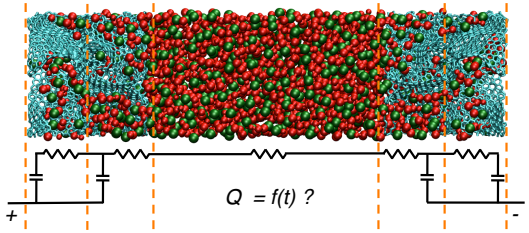
CPU **GPU**



Smart data management → less memory transfers

FOCUS ON METALWALLS

Memory management



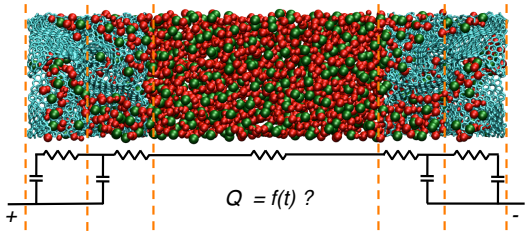
- Explicit data management
  - OpenACC declarations added to the function had no perf impact

```
!$acc data create(BUFF(:,:,:)) &
!$acc copyout(POT(nummove+1:num)) &
!$acc copyin(YELLOW(1:num,0:kmaxx),BLUE(1:num,0:kmaxy),RED(1:num,0:kmaxz)

!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector reduction(+:Tmp)
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  end do
   |  |  BUFFER(n,m,l) += Tmp
   |  enddo ll, mm, nn
   |  !$acc end parallel
   enddo
```

```
!cache_blocking
   do b = 0, nb_chunk
   |  !$acc parallel
   |  !$acc loop gang collapse(3)
   |  do l, m, n
   |  |  !$acc loop vector
   |  |  do i = start_chunk , end_chunk
   |  |  |
   |  |  |  GREEN = YELLOW(i,l) * BLUE(i,m)
   |  |  |
   |  |  |  Tmp += GREEN * RED(i,n)
   |  |  |
   |  |  | !$acc atomic update
   |  |  |  POT(i) += Tmp * BUFFER(n,m,l)
   |  |  |
   |  |  end do
   |  enddo ll, mm, nn
   enddo
!$acc end data
```

FOCUS ON METALWALLS

OpenACC vs OpenMP implementations



❑Easy to port OpenACC to OpenMP

You HAVE TO manage data with OpenMP

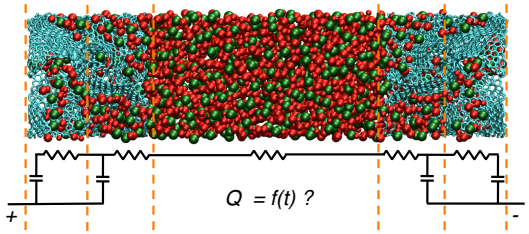
```
!$acc data create(BUFF(:,:,:)) &
!$acc copyout(POT(nummove+1:num)) &
!$acc copyin(YELLOW(1:num,0:kmaxx),BLUE(1:num,0:kmaxy),RED(1:num,0:kmaxz)
```

```
!$omp target data map(alloc:BUFF(1:x,1:y,1:z)) &
!$omp map(from:POT(nummove+1:num)) &
!$omp map(to:YELLOW(1:num,0:kmaxx),BLUE(1:num,0:kmaxy),RED(1:num,0:kmaxz)
```

- **Using OpenMP, you HAVE TO be explicit for the array lengths, otherwise you get wrong results**
- **OpenMP kernel gives first interesting results**

FOCUS ON METALWALLS

Preliminary results



❑Results on a very small testcase:

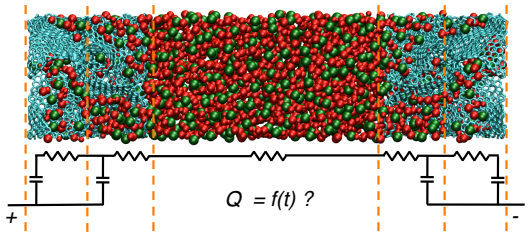
Kernel / Speed-up	Fortran MPI Power8 (20 cores)	OpenMP -O2 compilation 1 GPU (P100)	OpenMP -O3 compilation 1 GPU (P100)	OpenACC managed memory 1 GPU (P100)
cg_kernel	1	x1.1	x0.92	x5

❑Speed-ups

- Better speed-up using -O2 than using –O3
- OpenACC is 5 times faster on this case than OpenMP
- No other preliminary test showed better performances using OpenMP than OpenACC

CONCLUSIONS

Overall feedback



❑Porting to GPU with OpenACC

- Work has to be done on the way you express parallelism
  - Here we work on large arrays in 3 dimensions
- Compiler helps a lot for memory management
  - Automatic management using -ta=tesla:managed compilation option
- Very good performances

❑Porting to GPU with OpenMP

- Easy if you start from OpenACC
- Difficult to have a feedback from the XL compiler (compared to OpenACC)
- Performances are still low at the moment
  - Has to be improved on the compiler side
- OpenMP for GPU is not understood by CPUs

❑OpenMP-GPU is a more and more serious candidate for using GPUs

Rejoignez-nous!

www.genci.fr

www.prace-ri.eu