XKaapi : a runtime for highly parallel (OpenMP) application

Thierry Gautier
thierry.gautier@**inrialpes.fr**
MOAIS, INRIA, Grenoble

C2S@Exa 10-11/07/2014

Agenda

- context
- objective
- task model and execution
- status

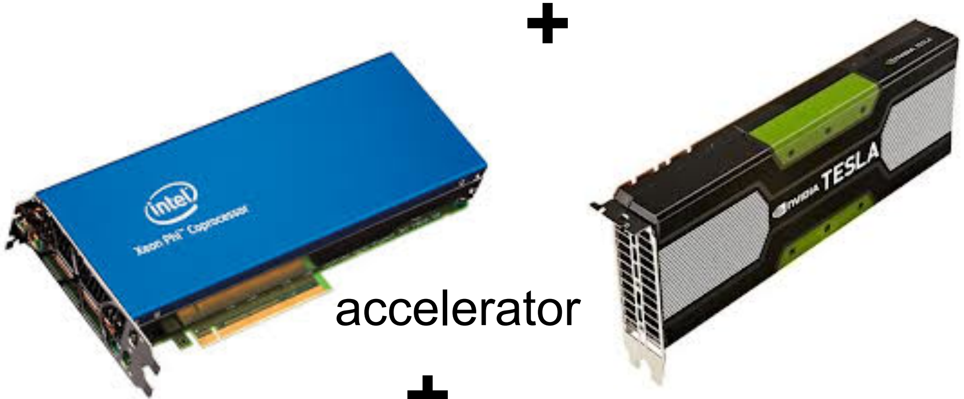
Computer

processor

\+

memory

\+



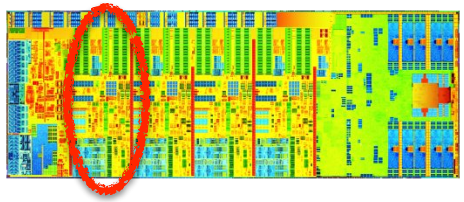
accelerator

\+

Processor architecture

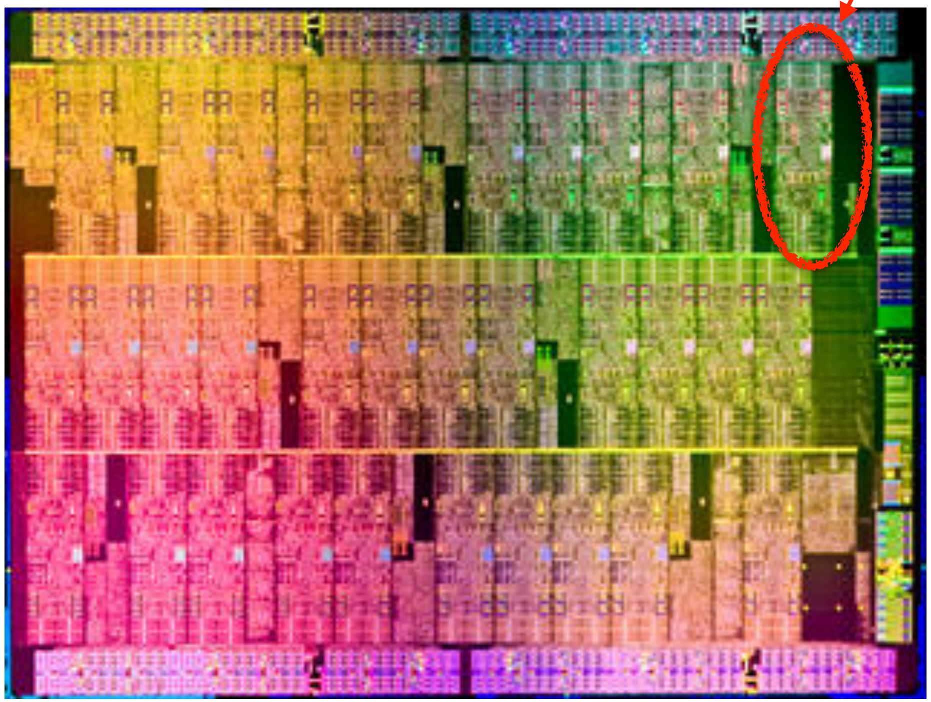
- multicore
- caches

4-cores AMD Barcelona

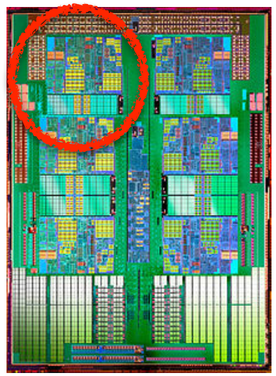


4-cores Haswell

1 core

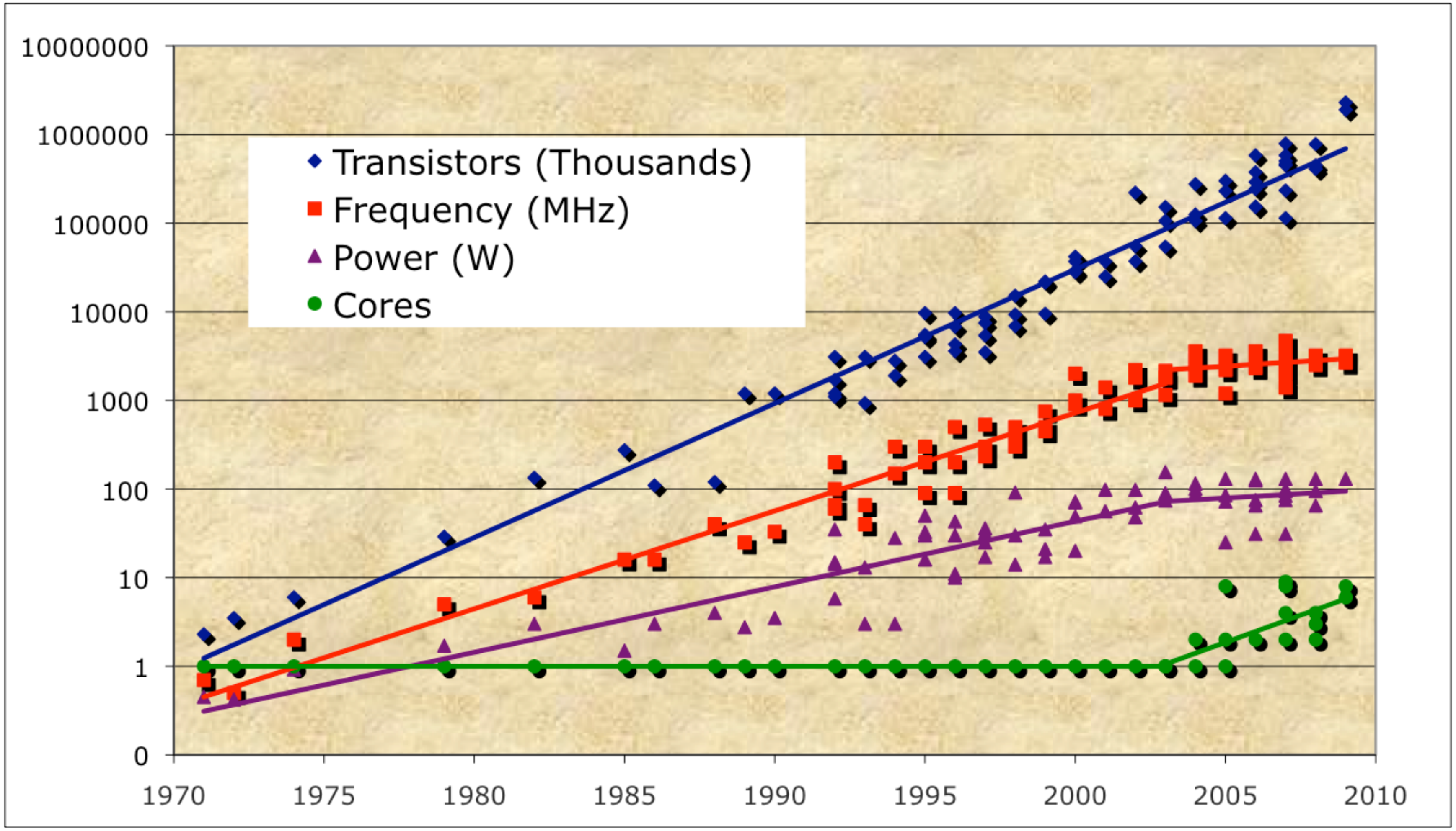


Intel Xeon Phi, 60 cores !



6-cores AMD Istanbul

Trends

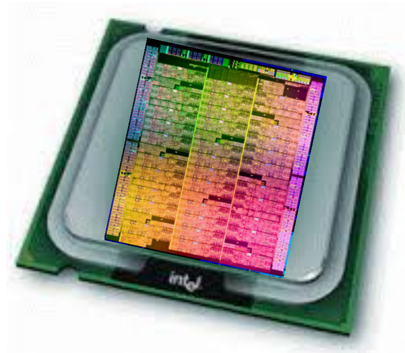


Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

- **many-many cores**
- **cache hierarchies**
- **hybrid**
  - **AMD Fusion, Nvidia**

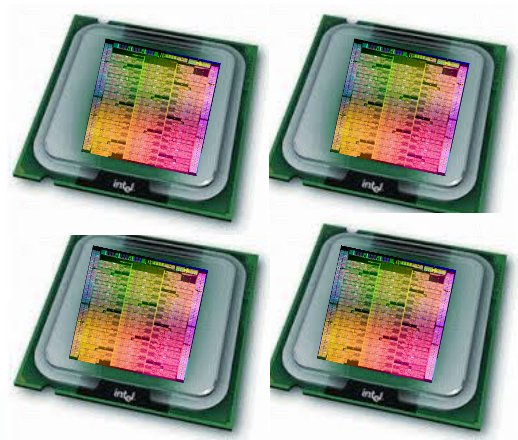
Knights Landing: multi-many cores

~ 2015
socket version



No more offloading
Direct access to main memory

- **4 sockets Intel Xeon Phi machine ?**
  - **~ 4 * 72 = 288 cores**
  - **~ 4*4*72 = 1152 hyperthreads**
  - **~ 4 * 3 Tflop/s**
  - **what about memory coherency… ?**



How to program them ?

- **Top-down classification**
  - **network**
    - MPI, PGAS (UPC), X10
  - **multi-core**
    - OpenMP, X10, Cilk, Intel TBB, XKaapi, StarPU…
  - **accelerator**
    - Cuda, OpenCL, OpenACC, OpenMP (4.0)
  - **vecteur / unité SIMD**
    - compilateur, type étendu, OpenMP (4.0)

} OpenMP 4.0 !

- **Which parallel programming environment ?**
  - **OpenMP-4.0**

    \+ OpenCL?

    \+ Cuda?

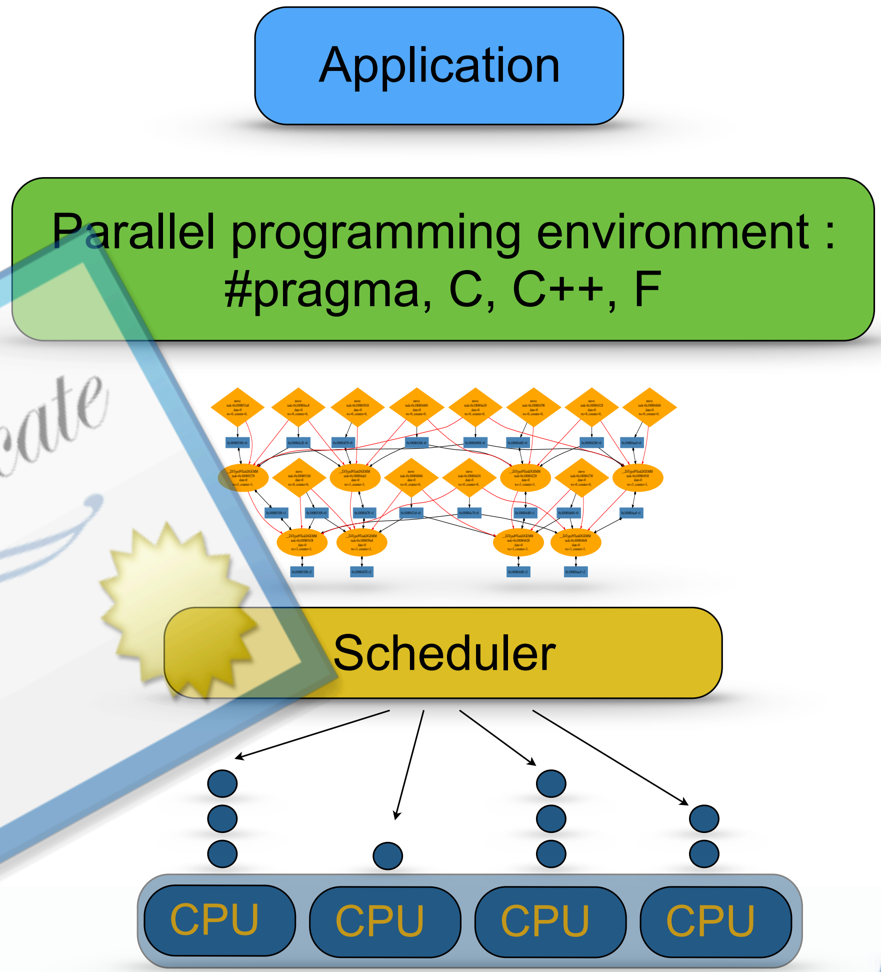
- **Challenge : same programming model, same code.**

Challenge: performance portability!

- **Performance portability**
  - **on several generations of architecture**
  - **with load balancing issues**
    - OS jitter, irregular application

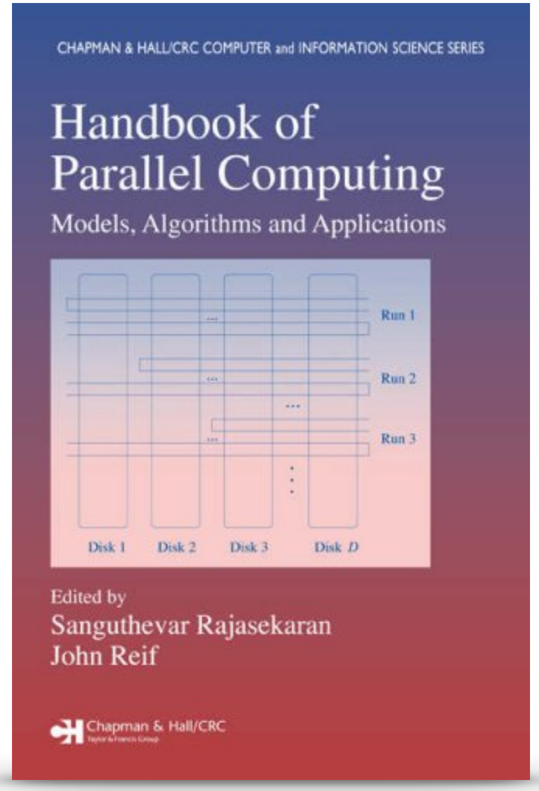
- **MOAIS team promotes a 2 steps approach**
  1. **parallel algorithms**
     - communications !
  2. **scheduling as a plugin**

- **Requirement of high level runtime features**



Which runtime features ?

- **Programming model pressure**
  - **Task (independent and dependent)**
    - extension
  - **Parallel loop**
- **Architecture pressure**
  - **High degree of parallelism**
  - **Memory hierarchy**
- **Performance guarantee**
  - **parallel algorithms**
  - **scheduling**
    - work stealing, HEFT, Dual approximation
    - …



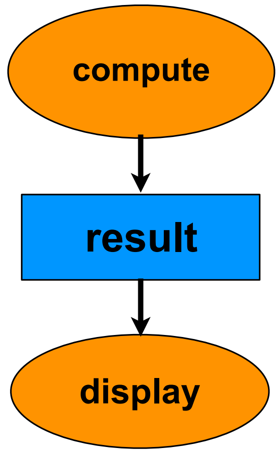
➡ **Management of many-(many) tasks**
  - **parallel slackness**
  - **spatial and temporal locality**
  - **scalable internal algorithms and data structure**

XKaapi task model

- **Athapascan task model**
  - **description of data dependencies (read/write/accumulation)**
- **Using OpenMP code annotation**

```
void main()
{
  /* data result is produced */
  compute( input, &result );

  /* data result is consumed */
  display( &result );

}
```

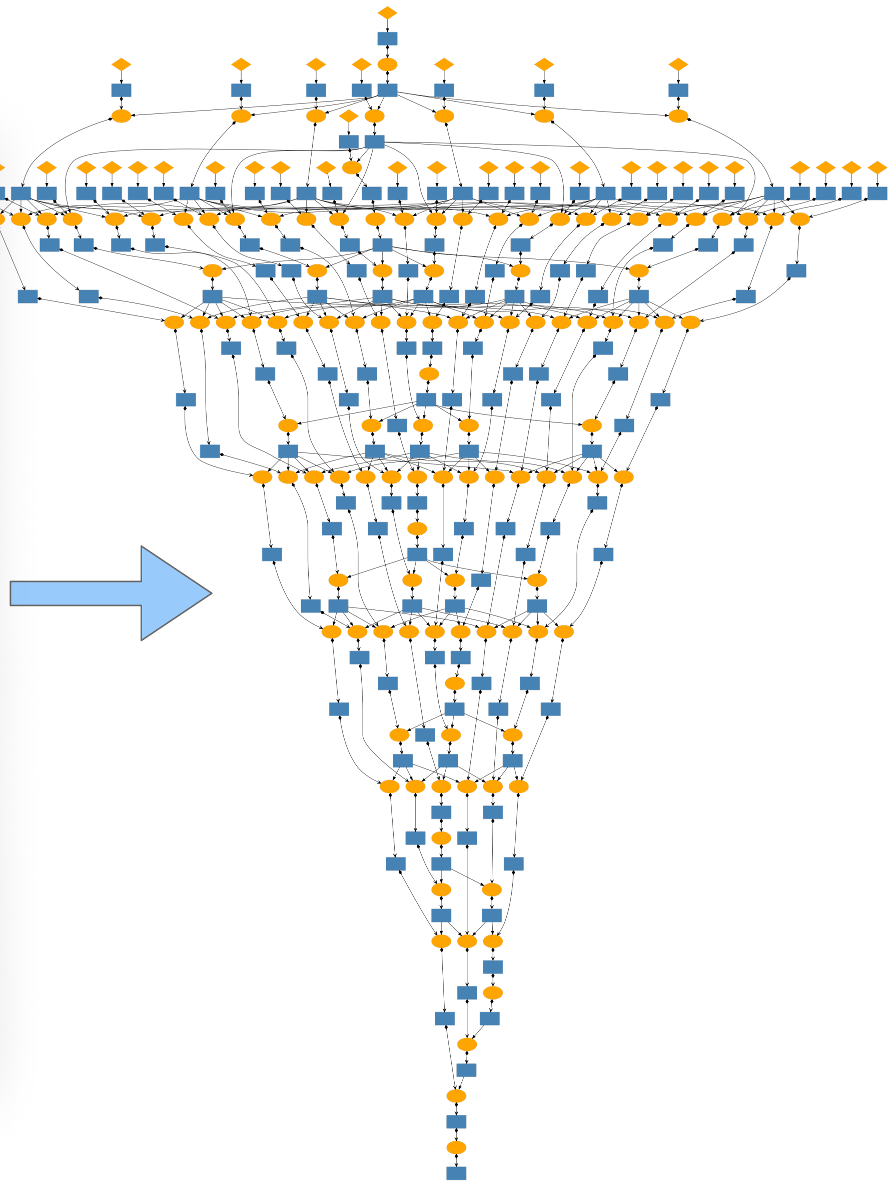
```
void main()
{
#pragma omp task depend(out: result)
  compute( input, &result );

#pragma omp task depend(in: result)
  display( &result );

}
```



- **Task**
  - **OpenMP structured block, function call**
  - **assumption: no side effect, description of access mode**
- **~ Sequential semantics**

Limitation

```
#include <cblas.h>
#include <clapack.h>

void Cholesky( int N, double A[N][N], size_t NB )
{
  for (size_t k=0; k < N; k += NB)
  {
#pragma omp task depend(inout: A[k:NB][k:NB]) shared(A)
    clapack_dpotrf( CblasRowMajor, CblasLower, NB, &A[k*N+k], N );

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: A[k:NB][k:NB]) \
                 depend(inout: A[m:NB][k:NB]) shared(A)
      cblas_dtrsm ( CblasRowMajor, CblasLeft, CblasLower, CblasNoTrans, CblasUnit,
        NB, NB, 1., &A[k*N+k], N, &A[m*N+k], N );
    }

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: A[m:NB][k:NB]) \
                 depend(inout: A[m:NB][m:NB]) shared(A)
     cblas_dsyrk ( CblasRowMajor, CblasLower, CblasNoTrans,
        NB, NB, -1.0, &A[m*N+k], N, 1.0, &A[m*N+m], N );

      for (size_t n=k+NB; n < m; n += NB)
      {
#pragma omp task depend(in: A[m:NB][k:NB], A[n:NB][k:NB])\
                 depend(inout: A[m:NB][n:NB]) shared(A)
        cblas_dgemm ( CblasRowMajor, CblasNoTrans, CblasTrans,
          NB, NB, NB, -1.0, &A[m*N+k], N, &A[n*N+k], N, 1.0, &A[m*N+n], N );
      }
    }
  }
#pragma omp taskwait
}
```

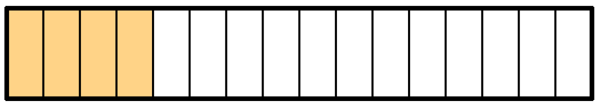


- **Size of the graph !**
- **Two complementary approaches**
  - fine grain task management: XKaapi task creation ~ 10cyles / task
  - compact dag representation for symbolic scheduling (DAGuE [UTK]), lazy task creation

Adaptive Task model

- **On-demand task creation**
  - **Adaptive tasks can be split at run time to create new tasks**
  - **Provide a «splitter» function called by idle cores on running task**
    - steal part of the remaining computation

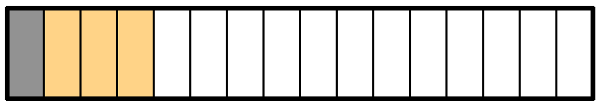
- **Typical example : parallel loop**
  - **A task == a range of iterations**
  - **Initially, one task in charge of the whole range**



Adaptive Task model

- **On-demand task creation**
  - **Adaptive tasks can be split at run time to create new tasks**
  - **Provide a «splitter» function called by idle cores on running task**
    - steal part of the remaining computation

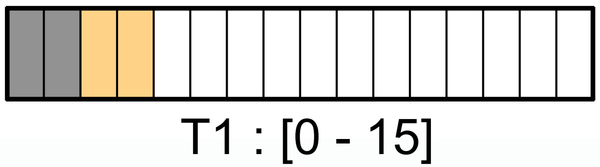
- **Typical example : parallel loop**
  - **A task == a range of iterations**
  - **Initially, one task in charge of the whole range**



Adaptive Task model

- **On-demand task creation**
  - **Adaptive tasks can be split at run time to create new tasks**
  - **Provide a «splitter» function called by idle cores on running task**
    - steal part of the remaining computation

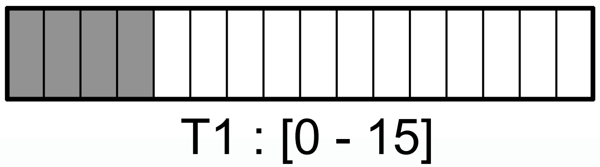
- **Typical example : parallel loop**
  - **A task == a range of iterations**
  - **Initially, one task in charge of the whole range**



Adaptive Task model

- On-demand task creation
  - Adaptive tasks can be split at run time to create new tasks
  - Provide a «splitter» function called by idle cores on running task
    - steal part of the remaining computation

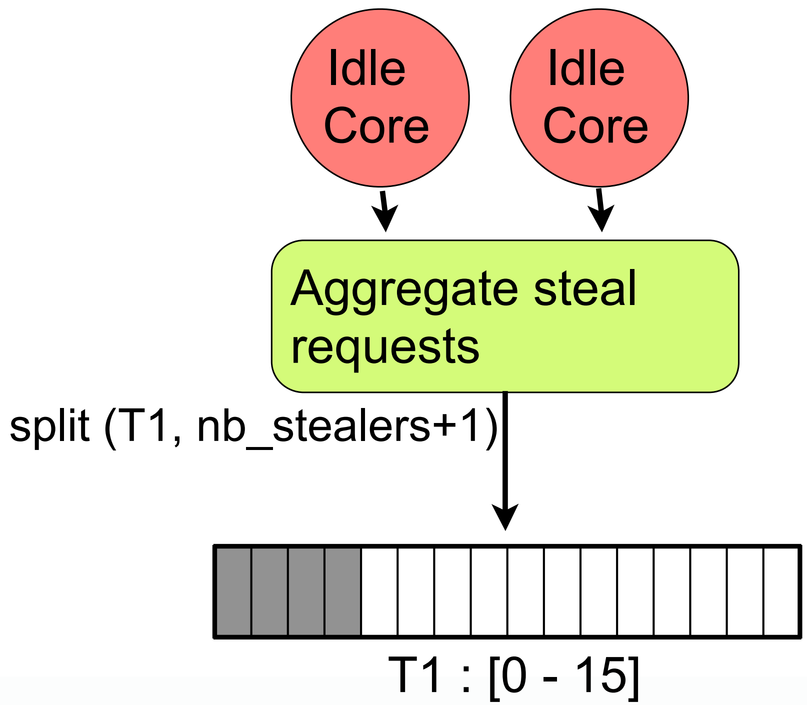
- Typical example : parallel loop
  - A task == a range of iterations
  - Initially, one task in charge of the whole range



Adaptive Task model

- **On-demand task creation**
  - **Adaptive tasks can be split at run time to create new tasks**
  - **Provide a «splitter» function called by idle cores on running task**
    - steal part of the remaining computation

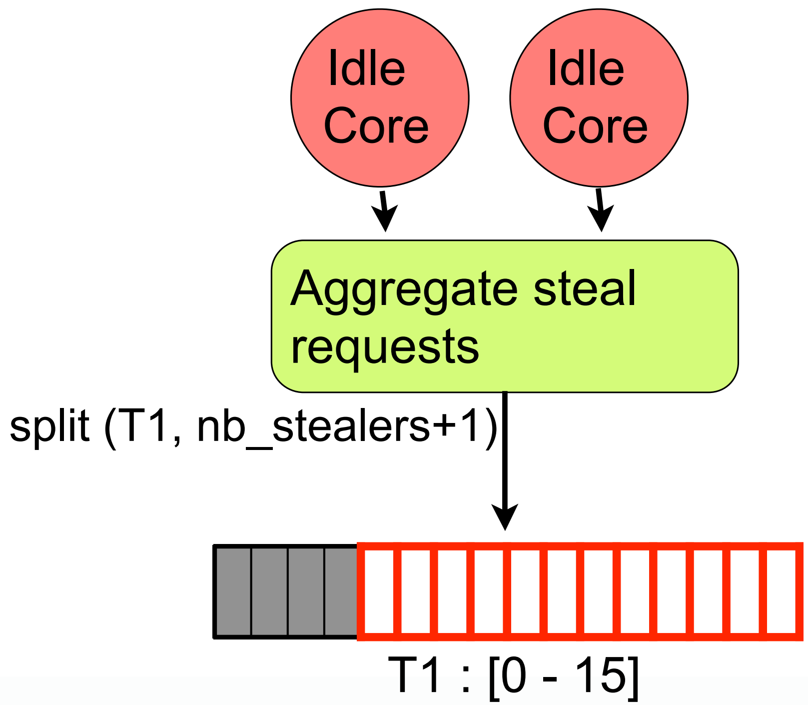
- **Typical example : parallel loop**
  - **A task == a range of iterations**
  - **Initially, one task in charge of the whole range**



Adaptive Task model

- **On-demand task creation**
  - **Adaptive tasks can be split at run time to create new tasks**
  - **Provide a «splitter» function called by idle cores on running task**
    - steal part of the remaining computation

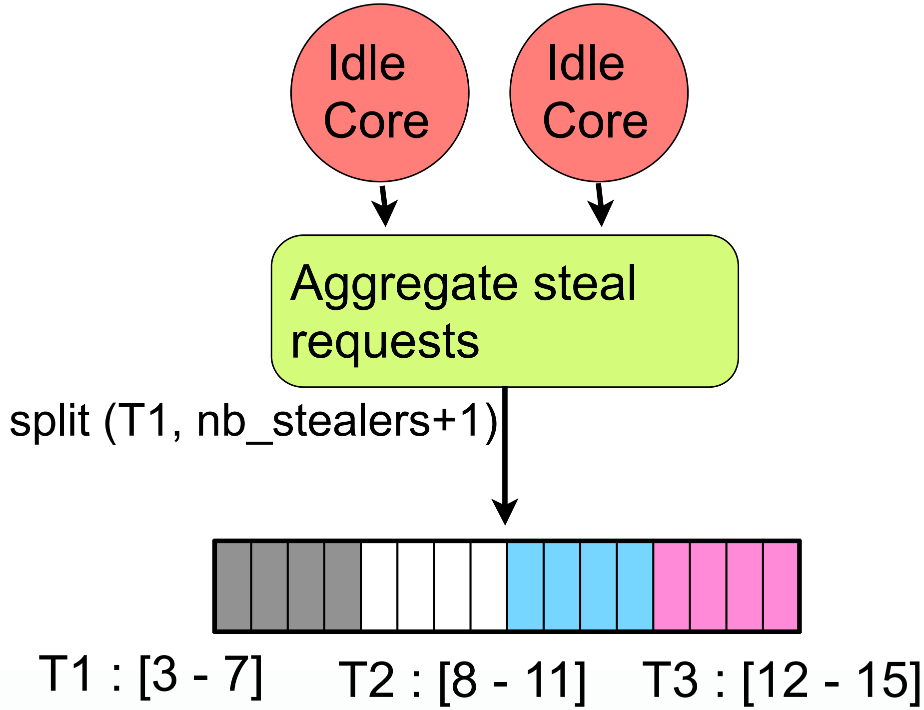
- **Typical example : parallel loop**
  - **A task == a range of iterations**
  - **Initially, one task in charge of the whole range**



Adaptive Task model

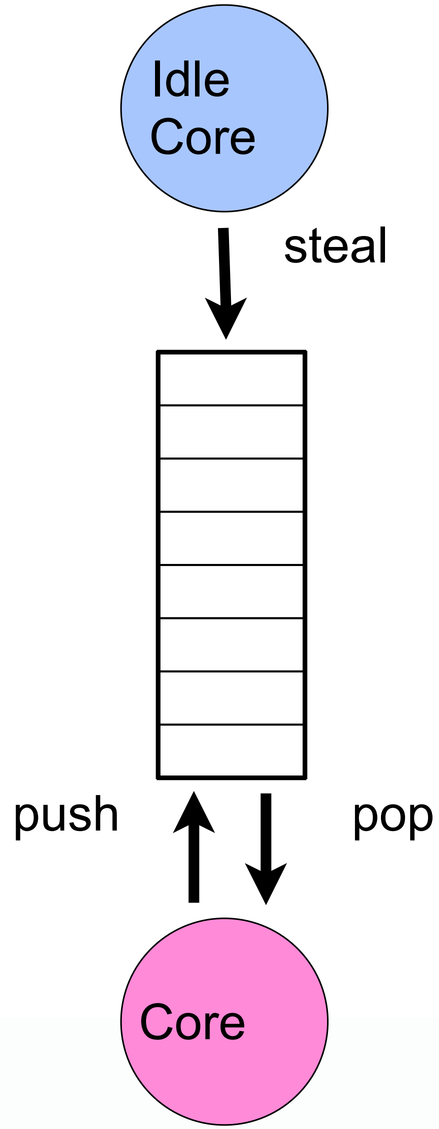
- **On-demand task creation**
  - **Adaptive tasks can be split at run time to create new tasks**
  - **Provide a «splitter» function called by idle cores on running task**
    - steal part of the remaining computation

- **Typical example : parallel loop**
  - **A task == a range of iterations**
  - **Initially, one task in charge of the whole range**



The way XKaapi executes tasks

- **Work-stealing based scheduling**
  - **One "worker thread" per core**
    - Holds a queue of tasks
- **push = Task creation is cheap !**
  - **Reduces to pushing C function pointer + its arguments into the worker thread queue**
    - ~ 10 cycles / tasks on AMD Many Cours processors
  - **Recursive tasks are welcome**
- **pop = Task execution by the owner is cheap !**
  - **pop the next task in the queue without computing dependencies**
    - sequential execution is a valid order
- **steal = the most costly operation**
  - **Compute ready tasks due dependencies**
    - Cilks's work first principle !



Steal operation

- Theoretical basis
  - **Work first principle: if application is highly parallel, steal operations are rare**
  - **Move overhead from the work to the critical path**

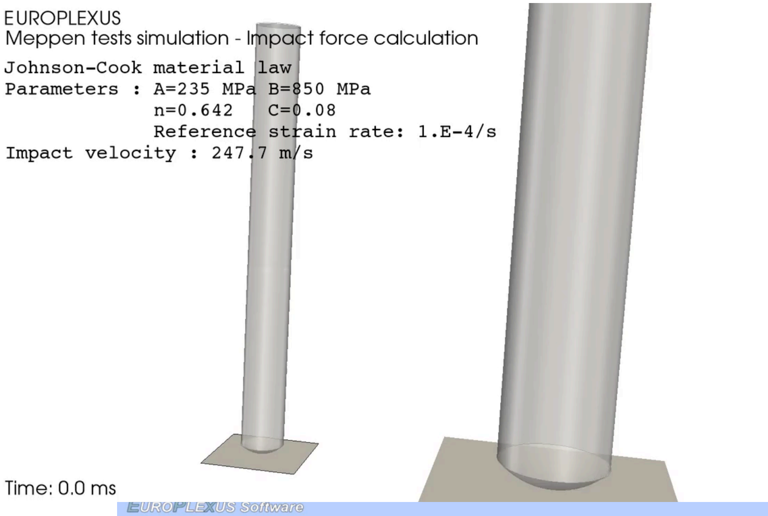
- Adaptive task
- Detection of ready task
  - **visit all tasks in a work queue following sequential order of creation**
    - for each task, visit all data accesses to detect dependencies with previous accesses
  - **detection of dependencies can be cached between steal operations**
    - allows to build the data flow graph for HEFT or GDUAL scheduler
    - new implementation in XKaapi-3.0, on going work

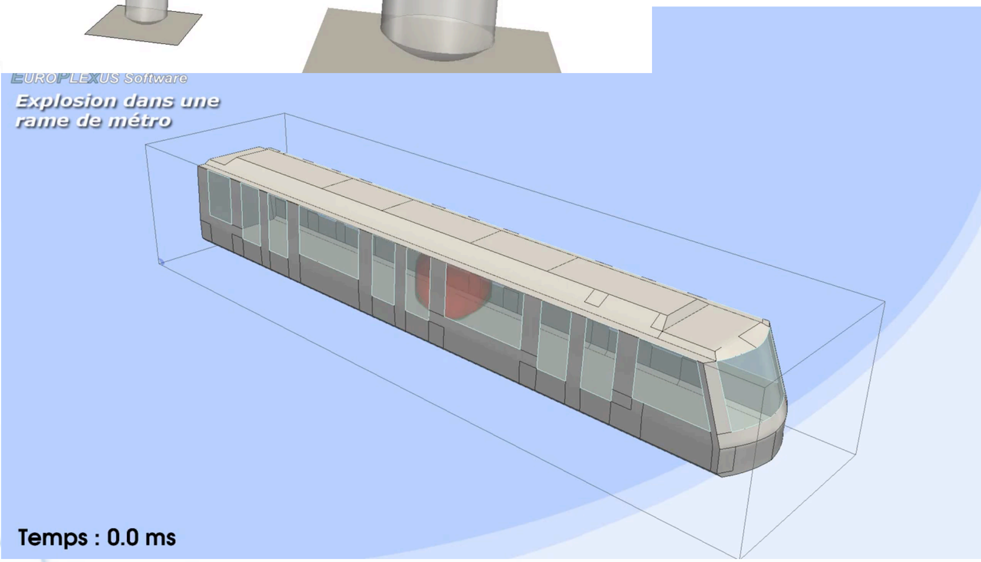
- Basic work stealing protocol
  - **idle core locks the victim queue**

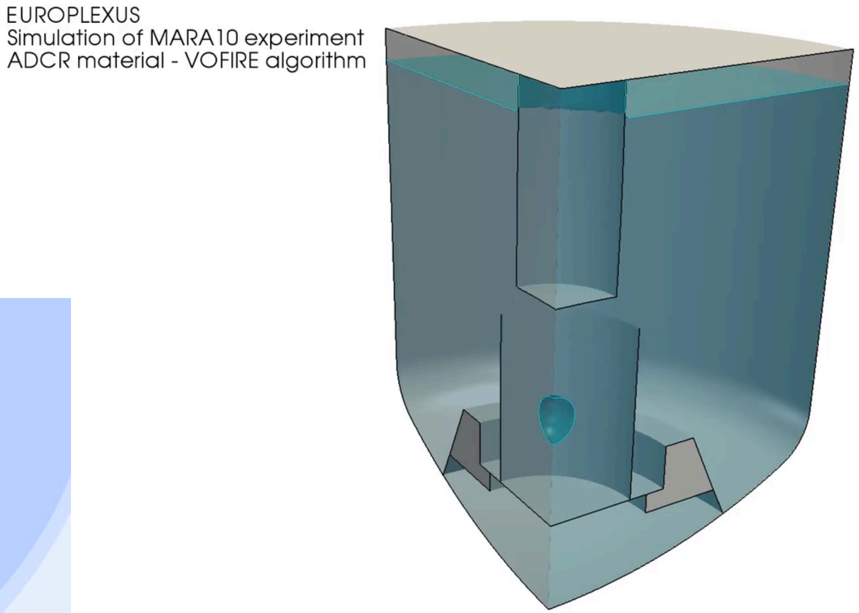
- Extension : work stealing requests aggregation
  - **idle core steals tasks for itself and all waiting cores (on the same victim)**
  - **only one visitor of the victim queue**

Case of study: EPX

- **EPX (EUROPLEXUS) code [CEA - IRC - EDF - ONERA], V. Faucher**
  - **Fluid-Structure systems subjected to fast transient dynamic loading**
- **Grand prix SFEN 2013**
  - **http://energies.sfen.org/emploi/le-grand-prix-sfen**







Multicore version of EPX

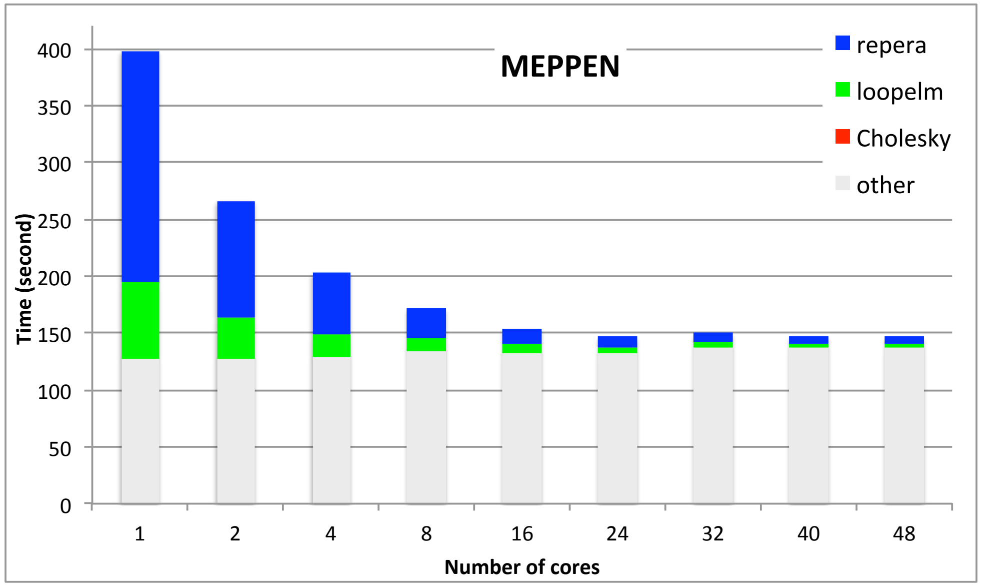
- **Complex code**
  - **600 000 lines of code (Fortran + MPI)**

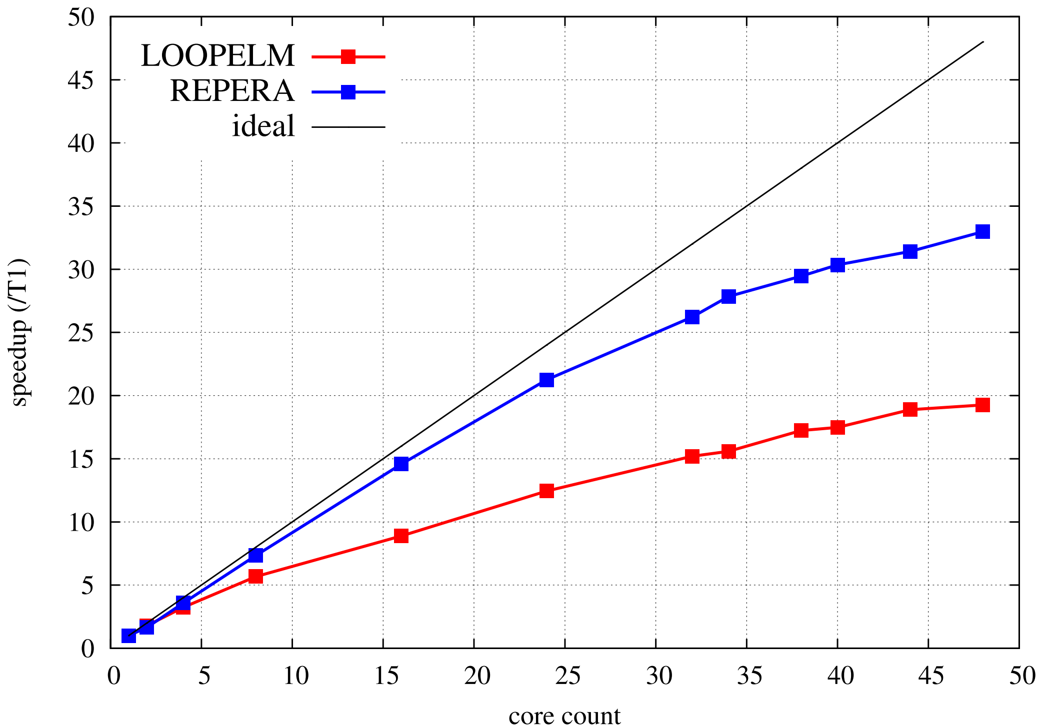
- **Two main sources of parallelization (~70% of the computation)**
  - **Sparse Cholesky factorization**
    - skyline representation
    - dependent tasks with data flow dependencies
  - **2 Independent loops**
    - LOOPELM:
      - iteration over finite elements to compute nodal internal forces
    - REPERA:
      - iteration for kinematic link detection

- **ANR REPDYN**
  - **2009-2012**
  - **parallelization using XKaapi**
    - tasks with data flow dependencies
    - on demand task creation for // loop
    - Fortran API
- **Phd Student [2013-2015]**

Case of study: MEPPEN

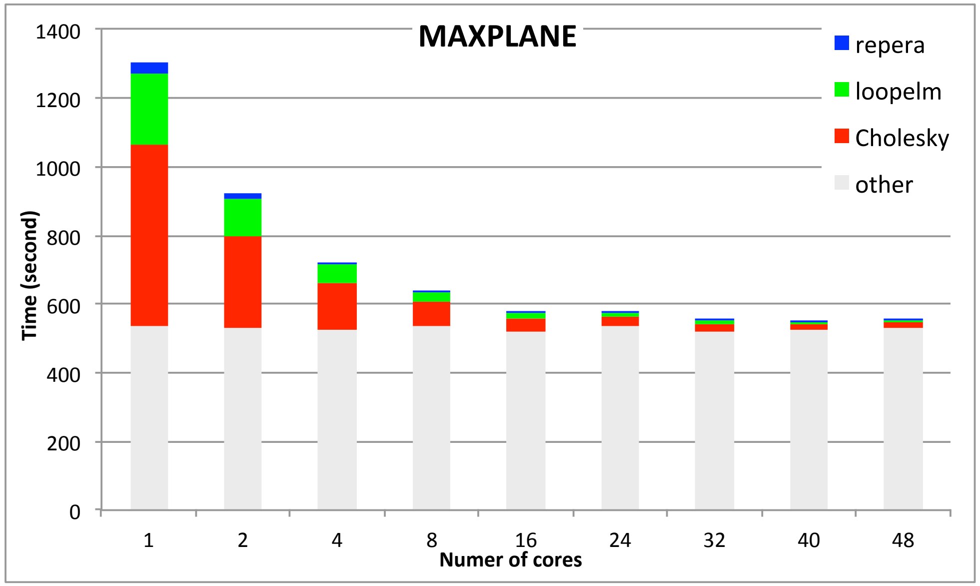
- **Main characteristics**
  - **Most of the time in independent loops LOOPELM and REPERA**
- **AMD Many Cours, 2.2GHz, 48 cores, 256GB main memory**

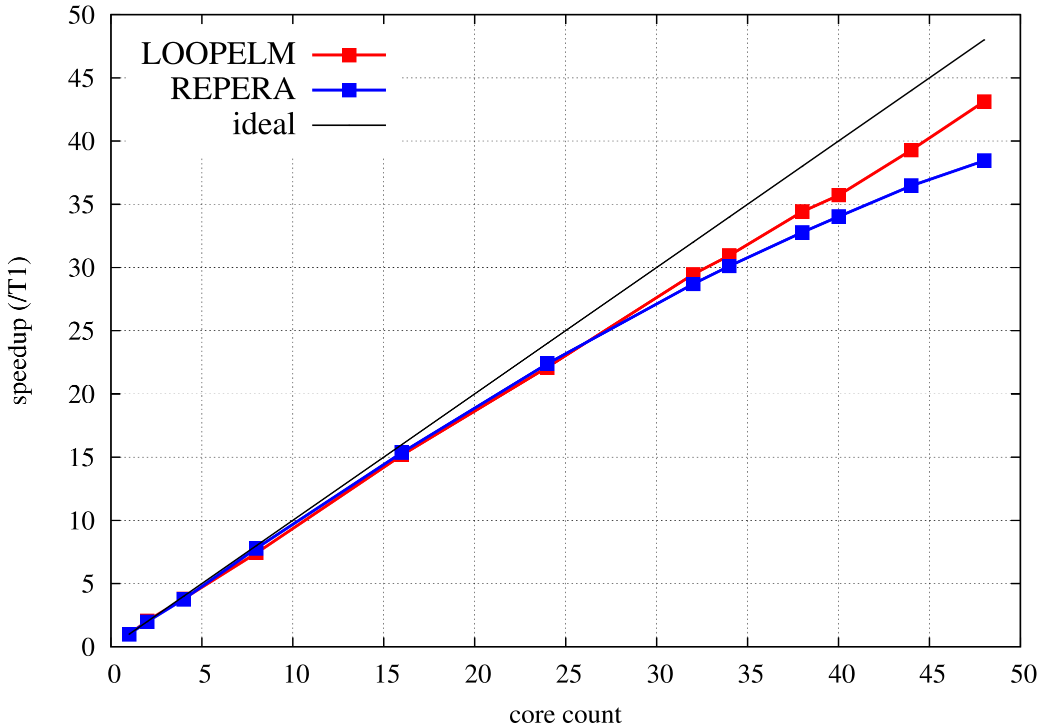




Case of study: MAXPLANE

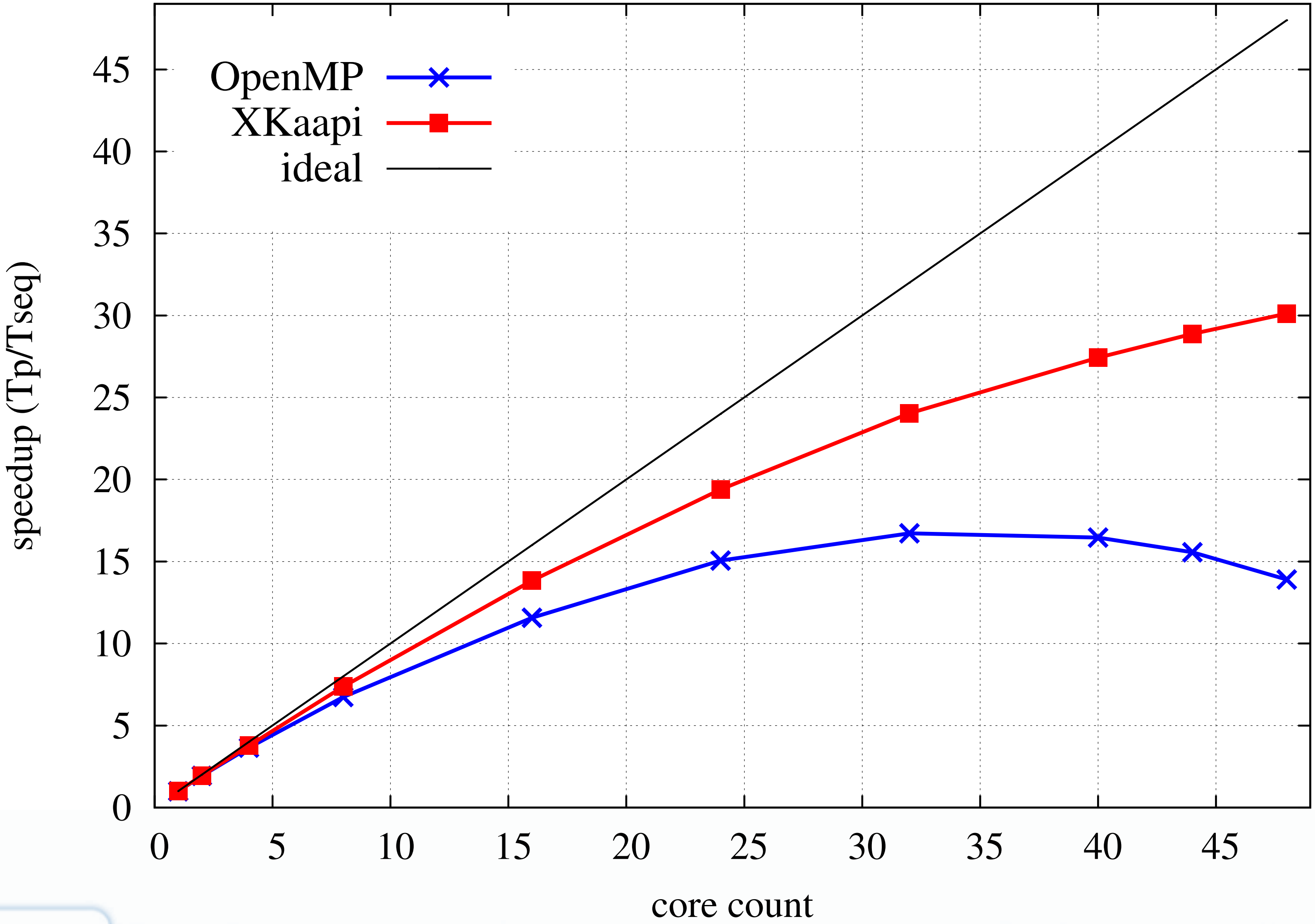
- **Main characteristics**
  - **Most of the time in sparse Cholesky factorization**
- **AMD Many Cours, 2.2GHz, 48 cores, 256GB main memory**





Sparse Cholesky Factorization (EPX)

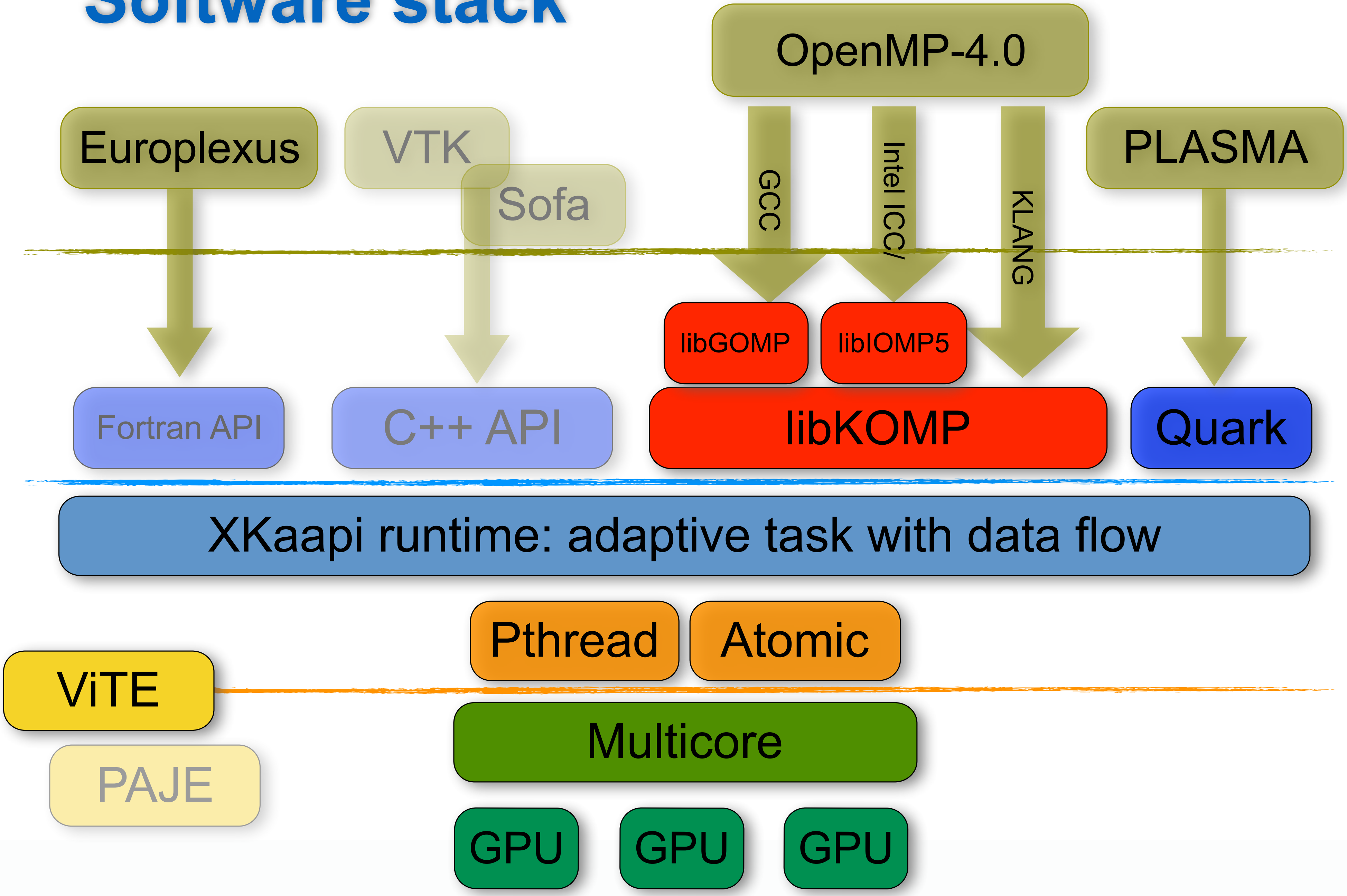
- **OpenMP / XKaapi = independent tasks versus dependent tasks**
  - **59462 with 3.59% of non zero elements**



XKaapi status

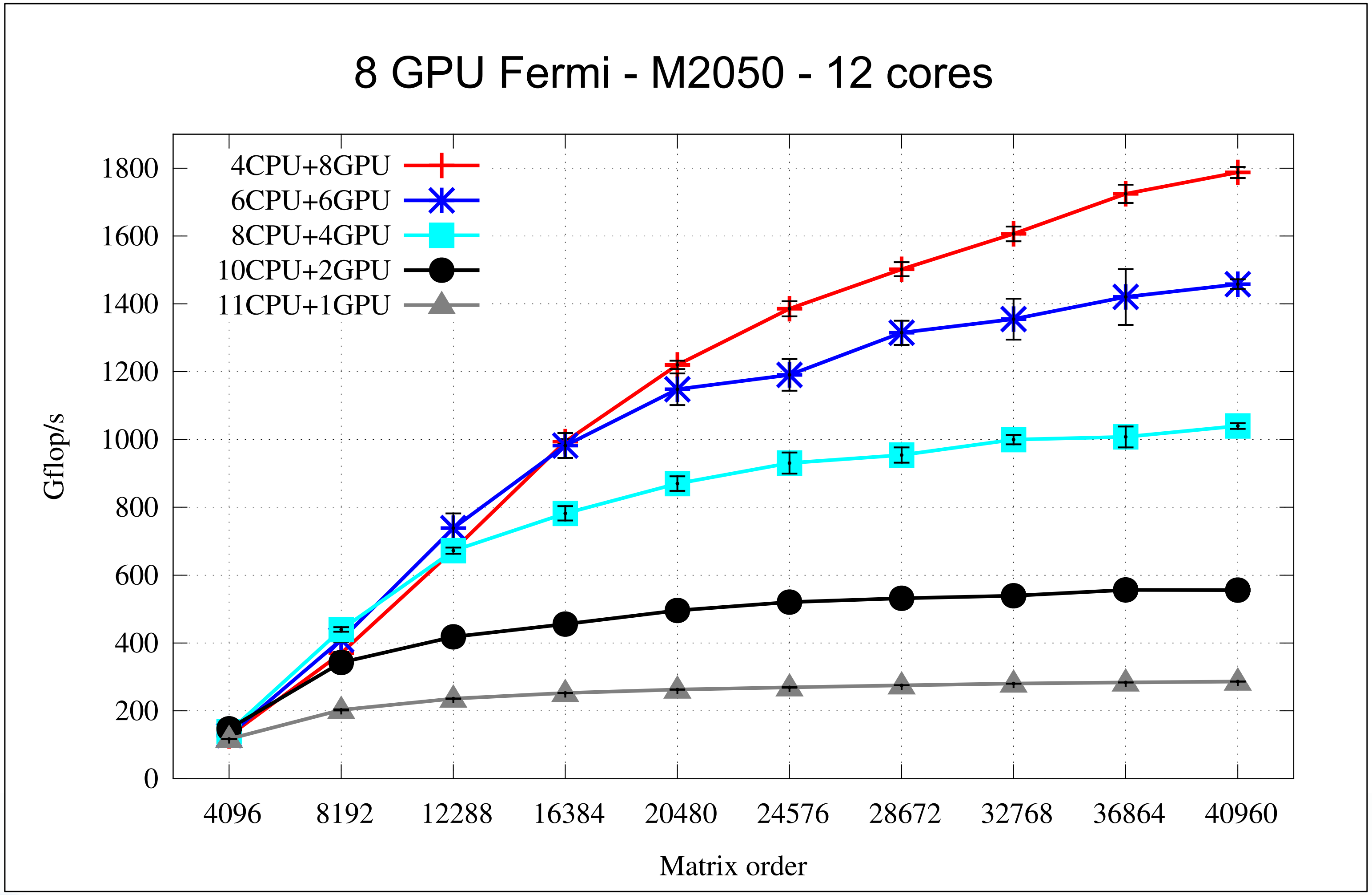
- software stack
- multi-gpu
- Intel Xeon Phi
- gdual scheduling for heterogeneous

Software stack



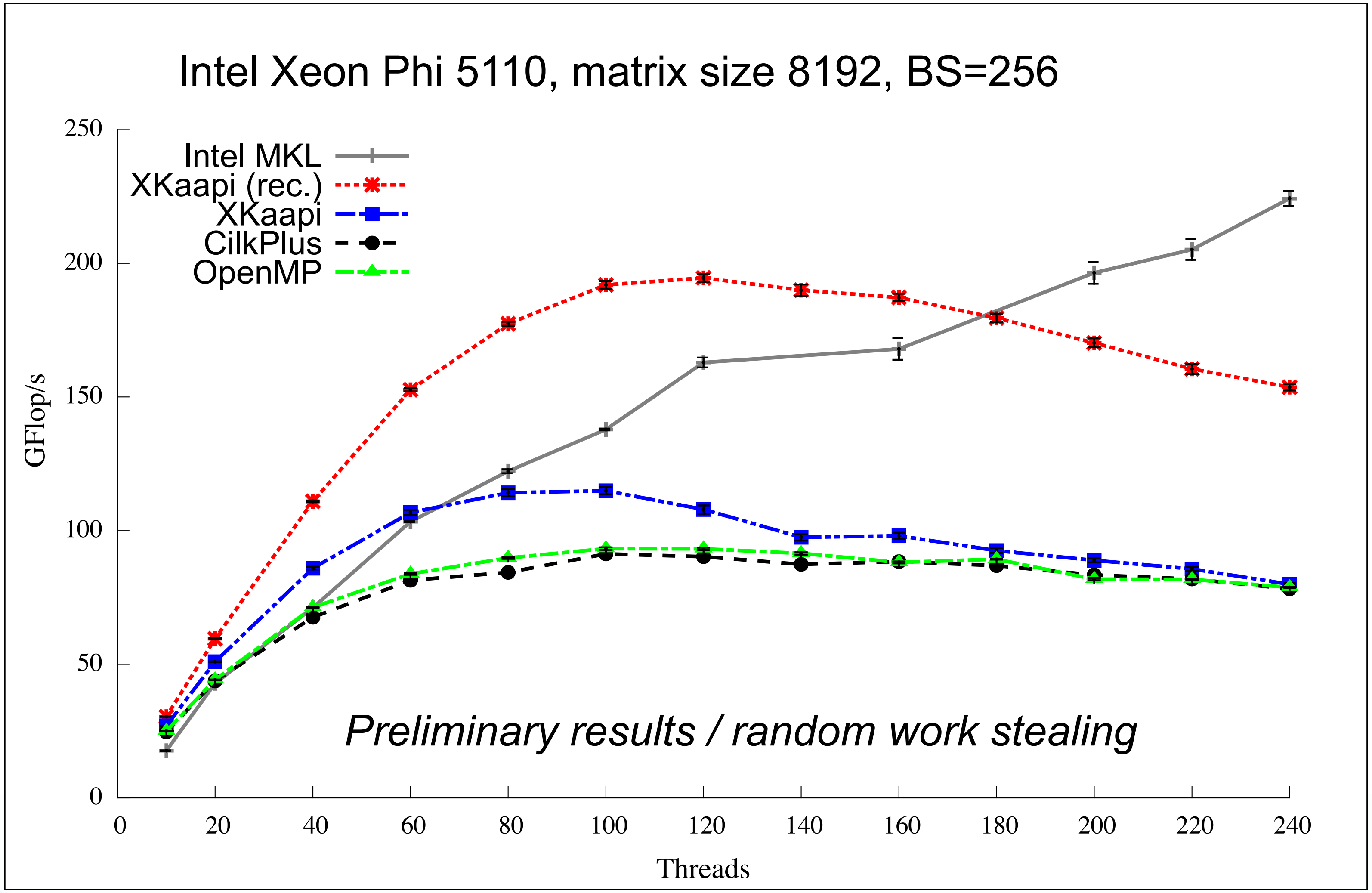
Multi-GPUs Support

- [IPDPS 2013] work with J. Lima, N. Maillard, B. Raffin
- DPOTRF
  - using recursive tasks (on panel factorization) to reduce critical path



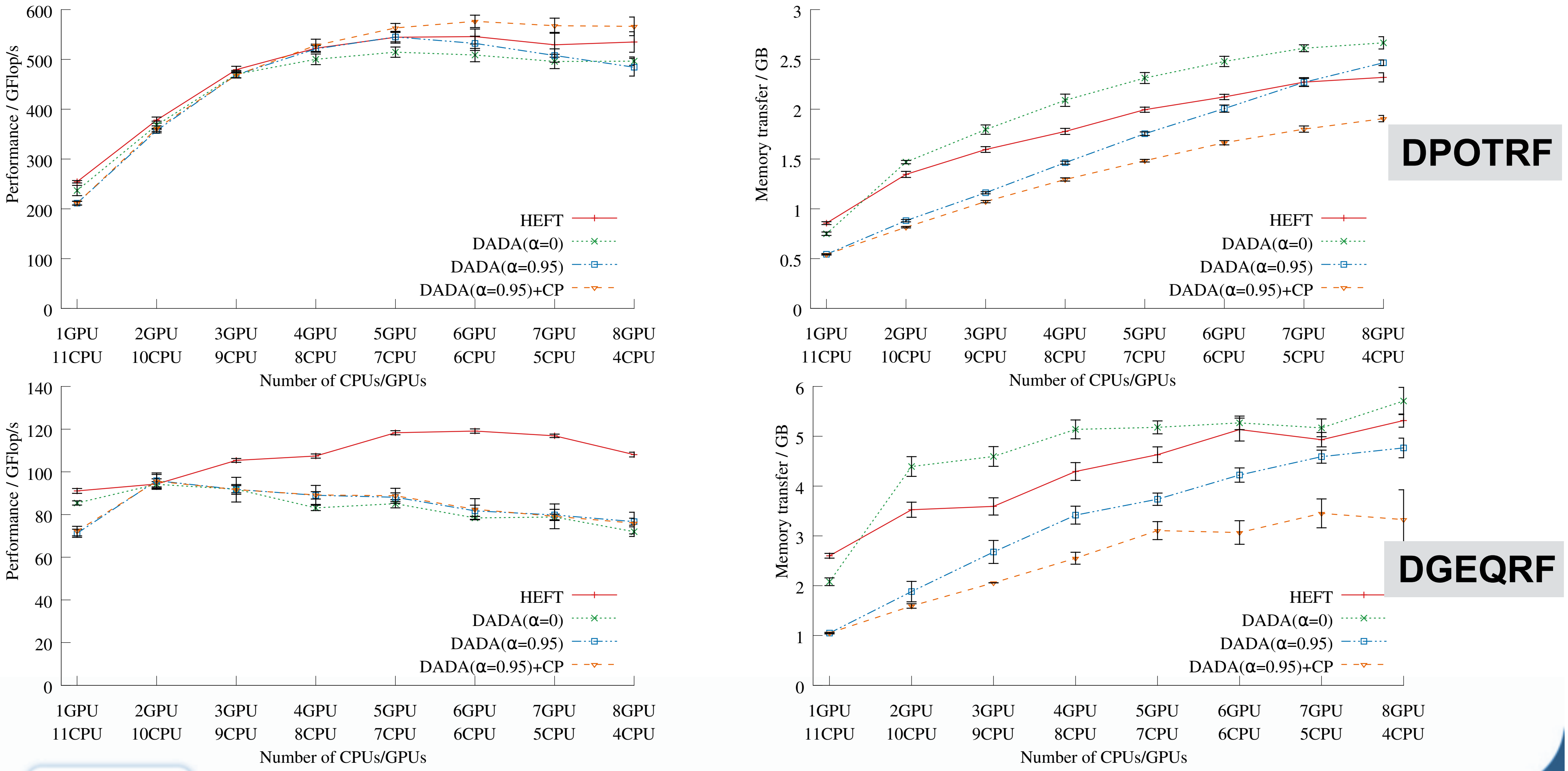
MIC Support

- [SBAC-PAD 2013] work with F. Broquedis, J. Lima, B. Raffin
  - Experiments with XKaapi on Intel Xeon Phi Coprocessor
- DPOTRF



Dual Approximation

- **[Europar 2014], work with D. Trystram, R. Bleuse, J. Lima**
  - Scheduling data flow program in XKaapi : A new affinity-based algorithm for heterogeneous architectures.
- **Optimization : communication and performances**



GFLOPS ⟷ Transfert

Conclusions

- **XKaapi**
  - **runtime of OpenMP + extensions**
    - libGOMP/GCC : very good support
    - libIOMP5/Intel or Clang-omp from Intel
  - **adaptive task**
  - **scheduling**
    - with / without « performance model »

- **Next steps**
  - **programming adaptive algorithms**
    - compiler extension for OpenMP
  - **improving scheduling of OpenMP task' based program**
    - temporal locality
  - **scalable algorithms**
    - many-(many) core
  - **reduce the size of the library**

INVENTEURS DU MONDE NUMÉRIQUE

XKaapi History