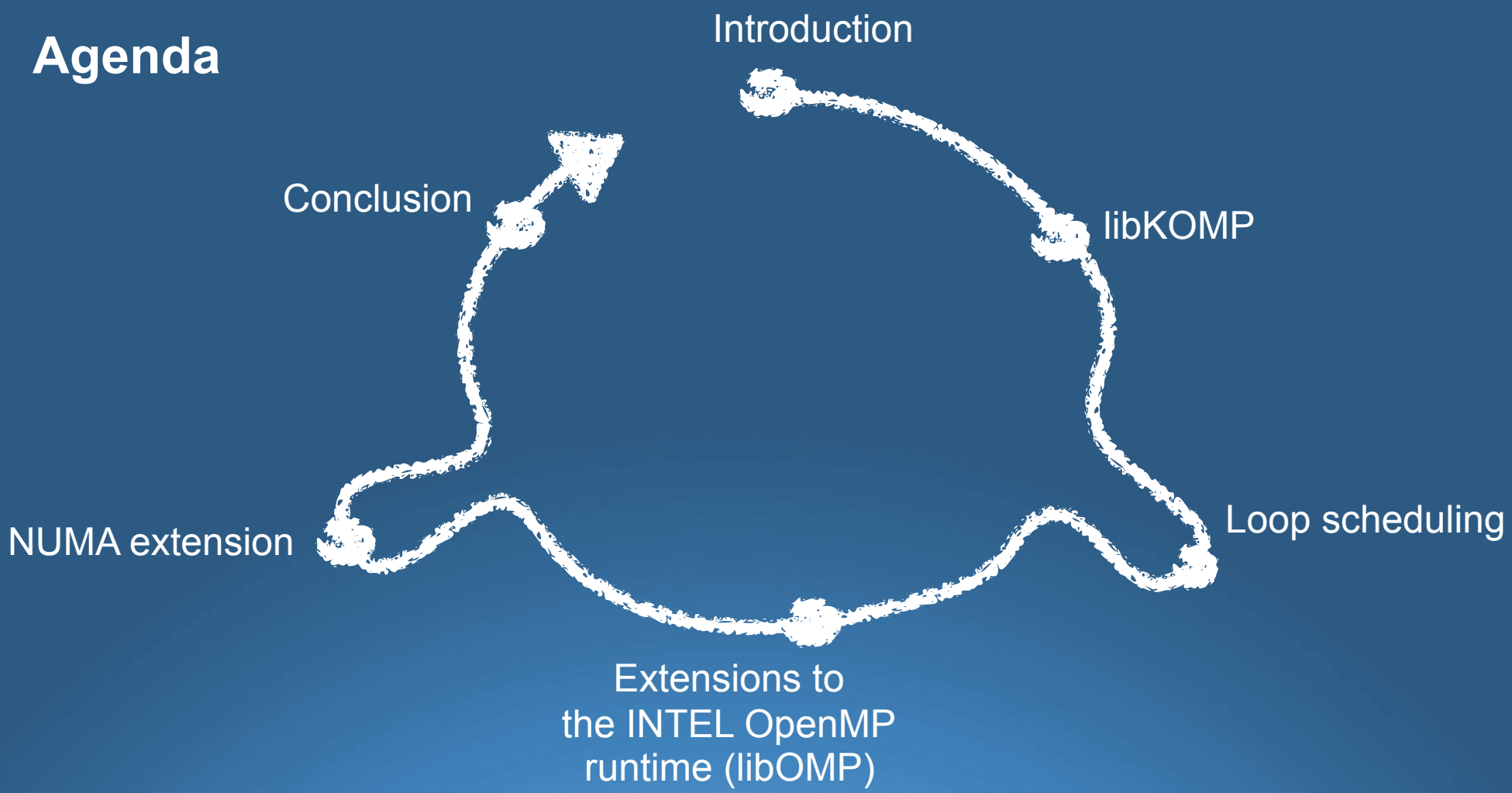
Improving an OpenMP runtime using XKaapi

thierry.gautier@inrialpes.fr
EPI AVALON

philippe virouleau [EPI CORSE]

Agenda


Introduction
libKOMP
Loop scheduling
Extensions to
the INTEL OpenMP
runtime (libOMP)
NUMA extension
Conclusion

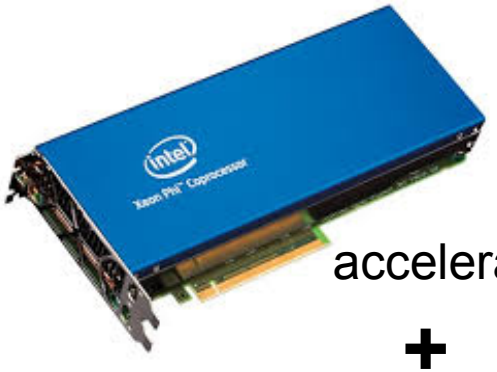
{Many,Multi}-{node,core}

processor

+

memory

+



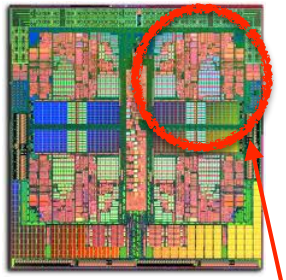
accelerator

+

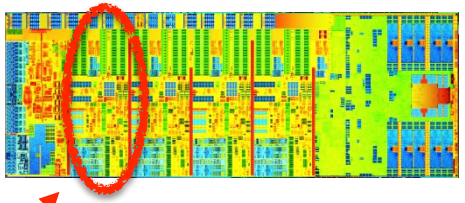
+ network
+ I/O
+ …

Inside Intel :-)

- multicore
- caches

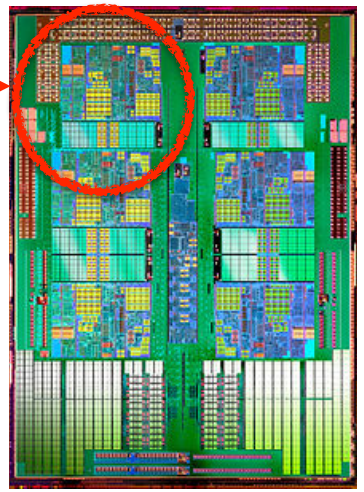


4-cores AMD Barcelona

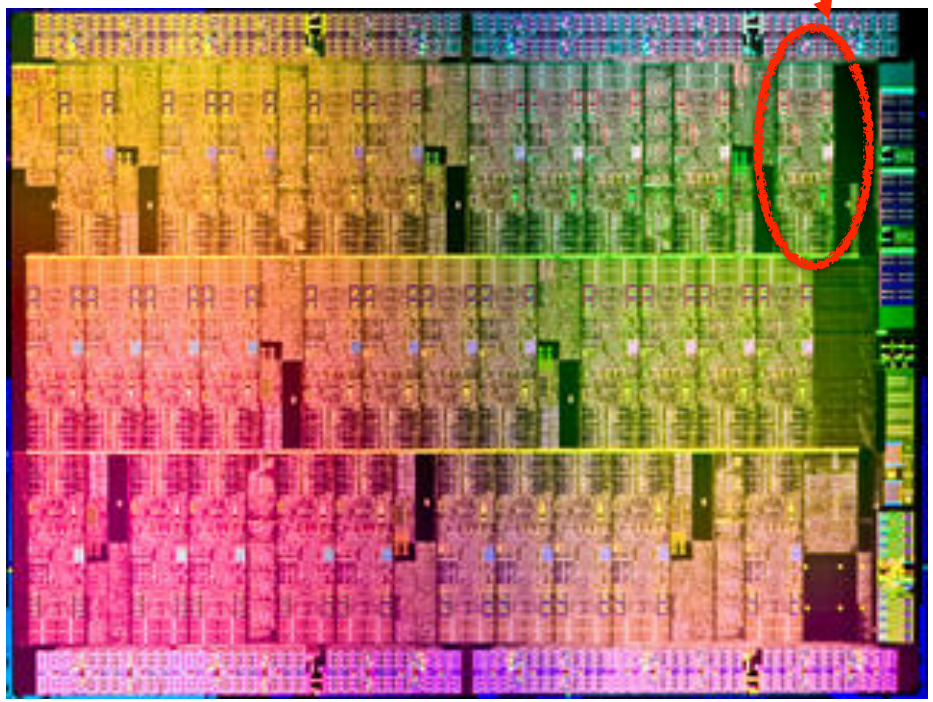


4-cores Haswell

1 core



6-cores AMD Istanbul



Intel Xeon Phi ~ **60** cores (KNC)

- GPU
- MPPA [Kalray]

How to program such computer?

Multi/many core area

Top-down classification

- network
  - MPI, PGAS (UPC), X10
- multi-core
  - OpenMP, Cilk, Intel TBB, XKaapi, StarPU…
- **accelerator**
  - Cuda, OpenCL, OpenACC, OpenMP (4.0)
- vector / SIMD unit
  - compiler, extended type, OpenMP (4.0)

Which parallel programming environment ?

OpenMP 4.x !

Permanent Members of the ARB:
- **AMD** (Greg Stoner)
- **ARM** (Chris Adeniyi-Jones)
- **Cray** (Luiz DeRose)
- **Fujitsu** (Eiji Yamanaka)
- **HP** (Sujoy Saraswati)
- **IBM** (Kelvin Li)
- **Intel** (Xinmin Tian)
- **Micron** (Kirby Collins)
- **NEC** (Kazuhiro Kusano)
- **NVIDIA** (Jeff Larkin)
- **Oracle Corporation** (Nawal Copty)
- **Red Hat** (Matt Newsome)
- **Texas Instruments** (Andy Fritsch)

Auxiliary Members of the ARB:
- **Argonne National Laboratory** (Kalyan Kumaran)
- **ASC/Lawrence Livermore National Laboratory** (Bronis R. de Supinski)
- **Barcelona Supercomputing Center** (Xavier Martorell)
- **cOMPunity** (Barbara Chapman/Yonghong Yan)
- **Edinburgh Parallel Computing Centre (EPCC)** (Mark Bull)
- **Los Alamos National Laboratory** (David Montoya)
- **Lawrence Berkeley National Laboratory** (Alice Koniges/Helen He)
- **NASA** (Henry Jin)
- **Oak Ridge National Laboratory** (Oscar Hernandez)
- **RWTH Aachen University** (Dieter an Mey)
- **Sandia National Laboratory** (Stephen Olivier)
- **Texas Advanced Computing Center** (Kent Milfeld)
- **University of Houston** (Barbara Chapman/Deepak Eachempati)

[+2016]
- **INRIA (O. Aumage)**

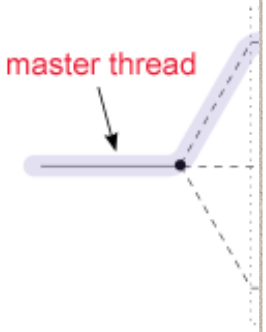
OpenMP

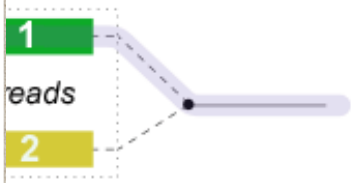
Based on directive

- compiler generat
- C,C++, Fortran

Execution model

- sequence of para



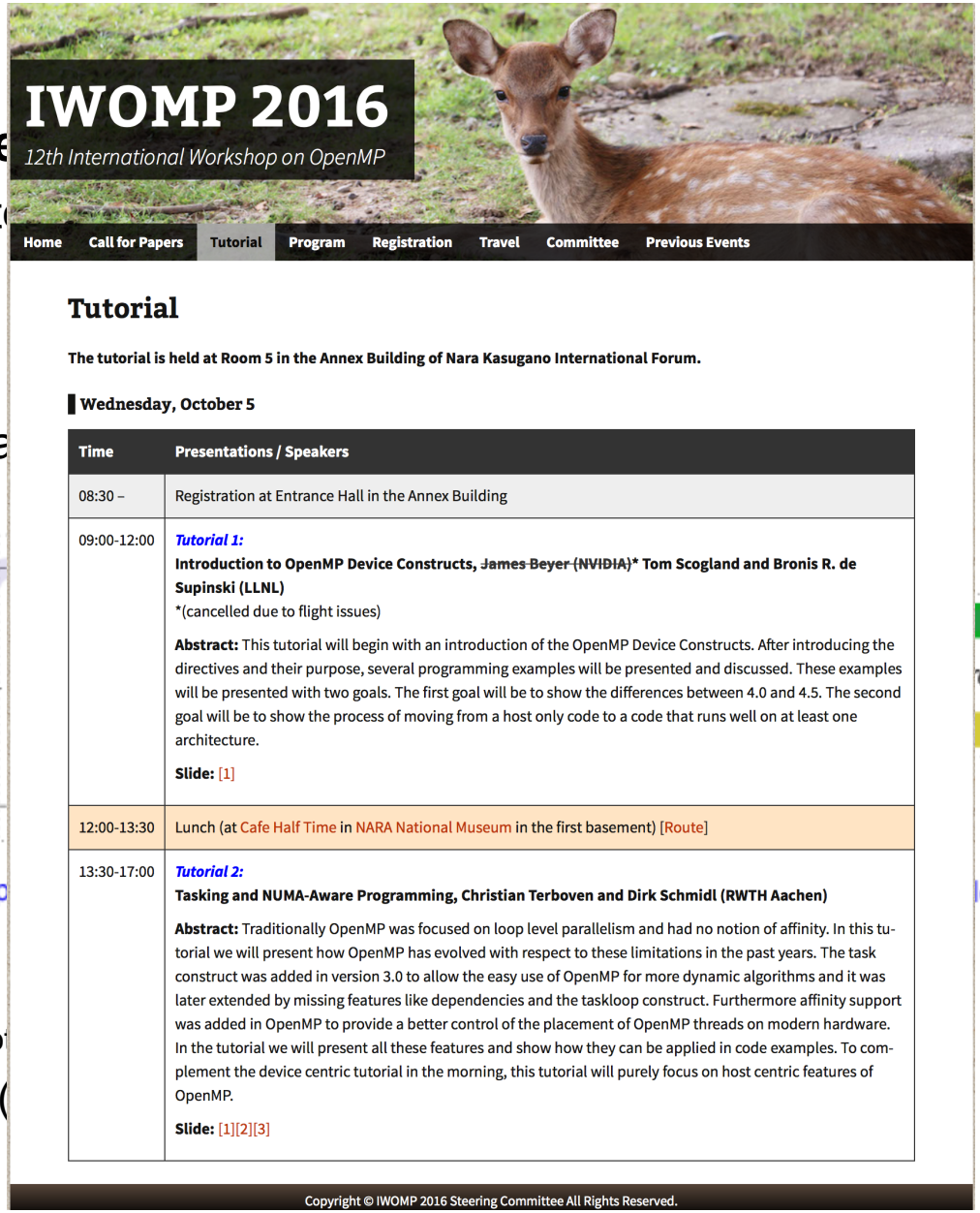


p

lel region

Directives for

- parallel loop and o
- Recursive tasks (
- synchronization



IWOMP 2016

12th International Workshop on OpenMP

Home | Call for Papers | Tutorial | Program | Registration | Travel | Committee | Previous Events

Tutorial

The tutorial is held at Room 5 in the Annex Building of Nara Kasugano International Forum.

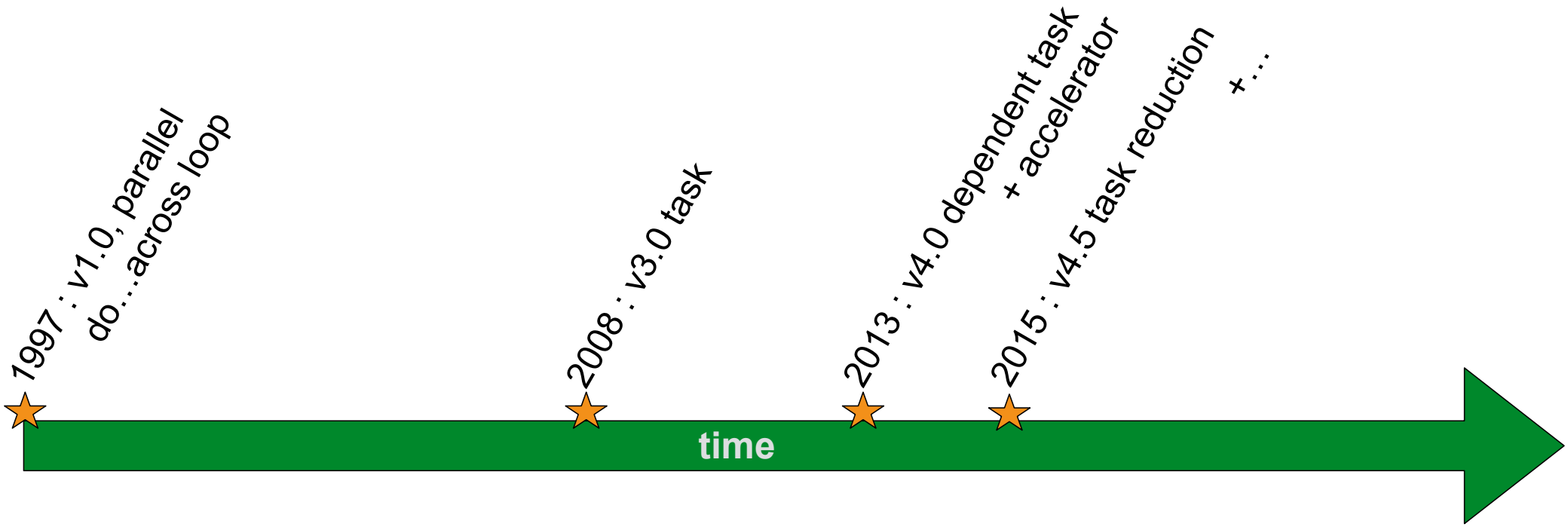
Wednesday, October 5

Time	Presentations / Speakers
08:30 –	Registration at Entrance Hall in the Annex Building
09:00-12:00	***Tutorial 1:*** **Introduction to OpenMP Device Constructs, ~~James Beyer (NVIDIA)~~* Tom Scogland and Bronis R. de Supinski (LLNL)** *(cancelled due to flight issues) **Abstract:** This tutorial will begin with an introduction of the OpenMP Device Constructs. After introducing the directives and their purpose, several programming examples will be presented and discussed. These examples will be presented with two goals. The first goal will be to show the differences between 4.0 and 4.5. The second goal will be to show the process of moving from a host only code to a code that runs well on at least one architecture. **Slide:** [1]
12:00-13:30	Lunch (at Cafe Half Time in NARA National Museum in the first basement) [Route]
13:30-17:00	***Tutorial 2:*** **Tasking and NUMA-Aware Programming, Christian Terboven and Dirk Schmidl (RWTH Aachen)** **Abstract:** Traditionally OpenMP was focused on loop level parallelism and had no notion of affinity. In this tutorial we will present how OpenMP has evolved with respect to these limitations in the past years. The task construct was added in version 3.0 to allow the easy use of OpenMP for more dynamic algorithms and it was later extended by missing features like dependencies and the taskloop construct. Furthermore affinity support was added in OpenMP to provide a better control of the placement of OpenMP threads on modern hardware. In the tutorial we will present all these features and show how they can be applied in code examples. To complement the device centric tutorial in the morning, this tutorial will purely focus on host centric features of OpenMP. **Slide:** [1][2][3]

What's new in OpenMP?

Parallel code thanks to annotation (directives)

- Fortran, C, C++

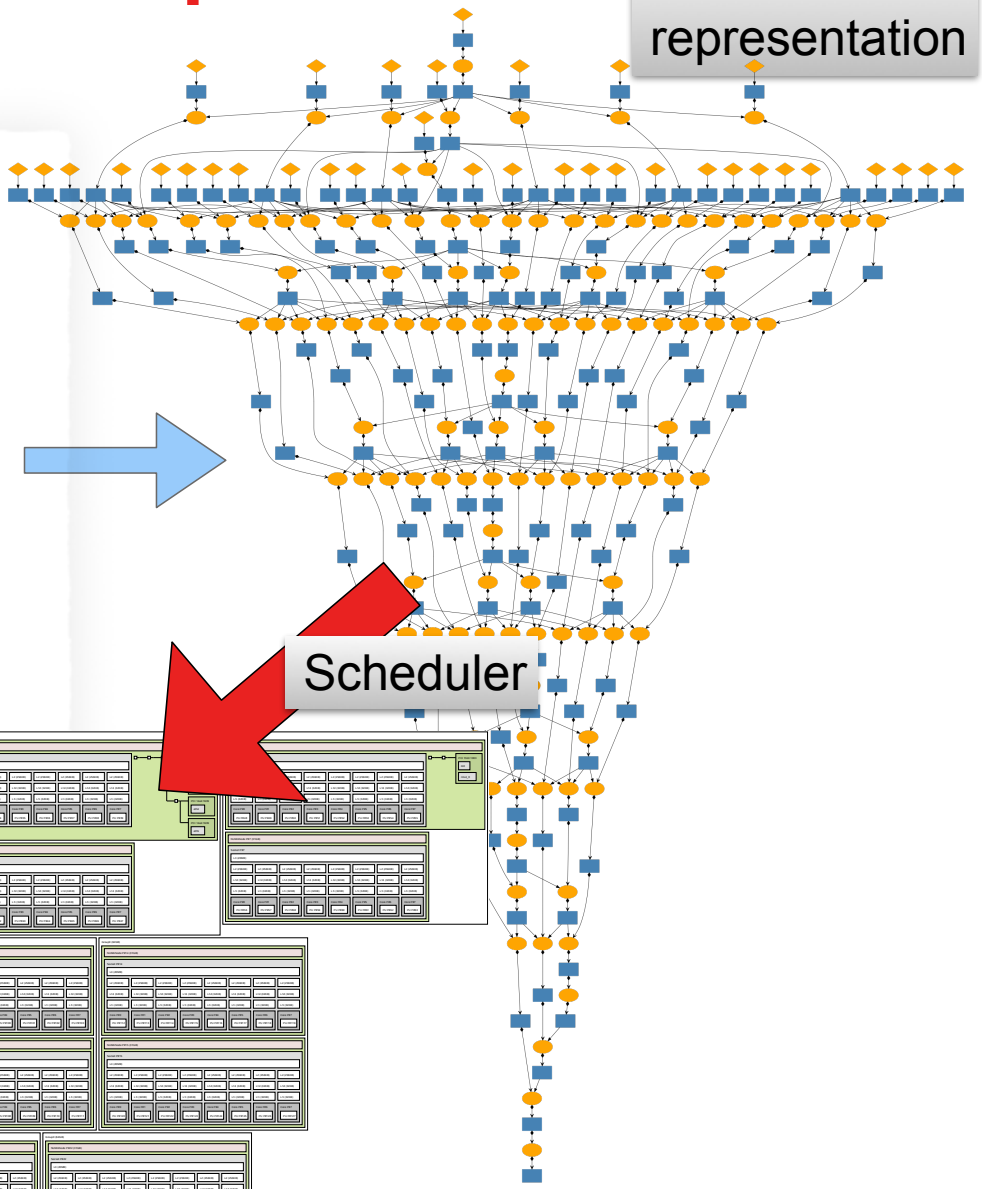


Program & execution with OpenMP

Application

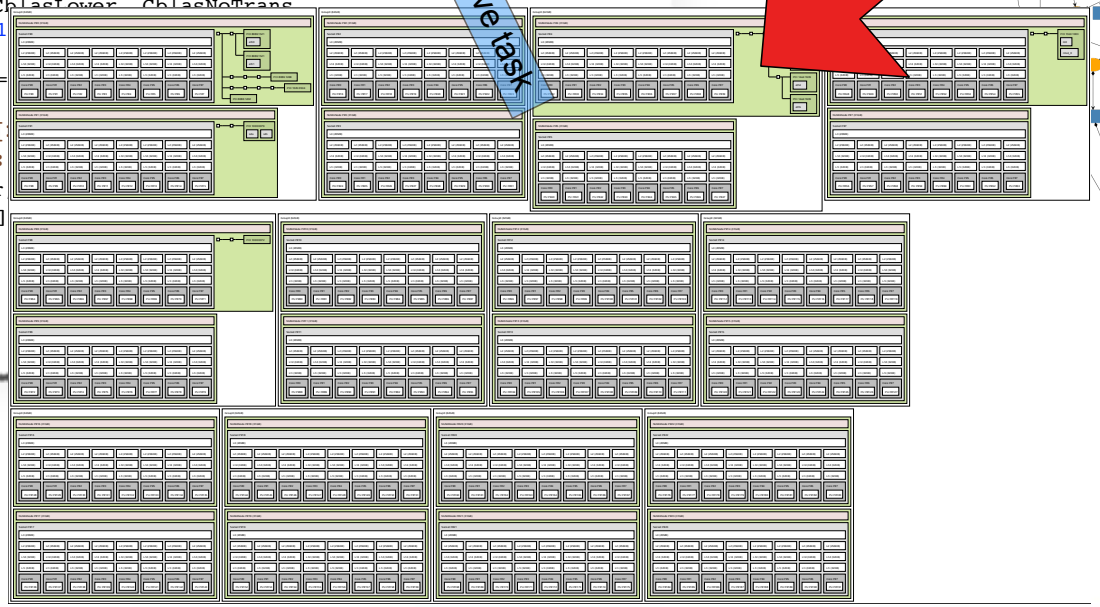
```
#include <cblas.h>
#include <clapack.h>

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

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

Data Flow representation



If recursive task

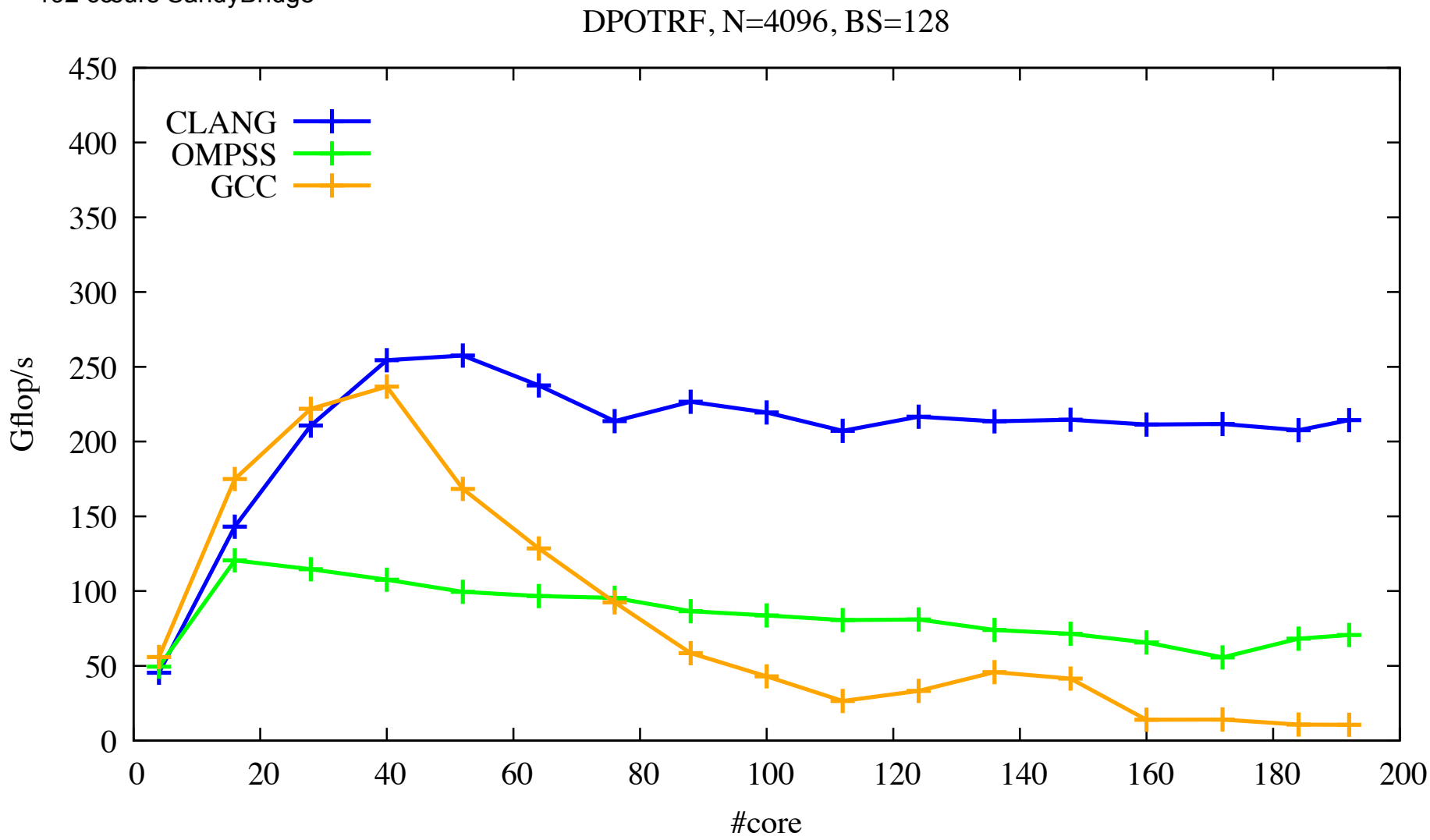
Scheduler



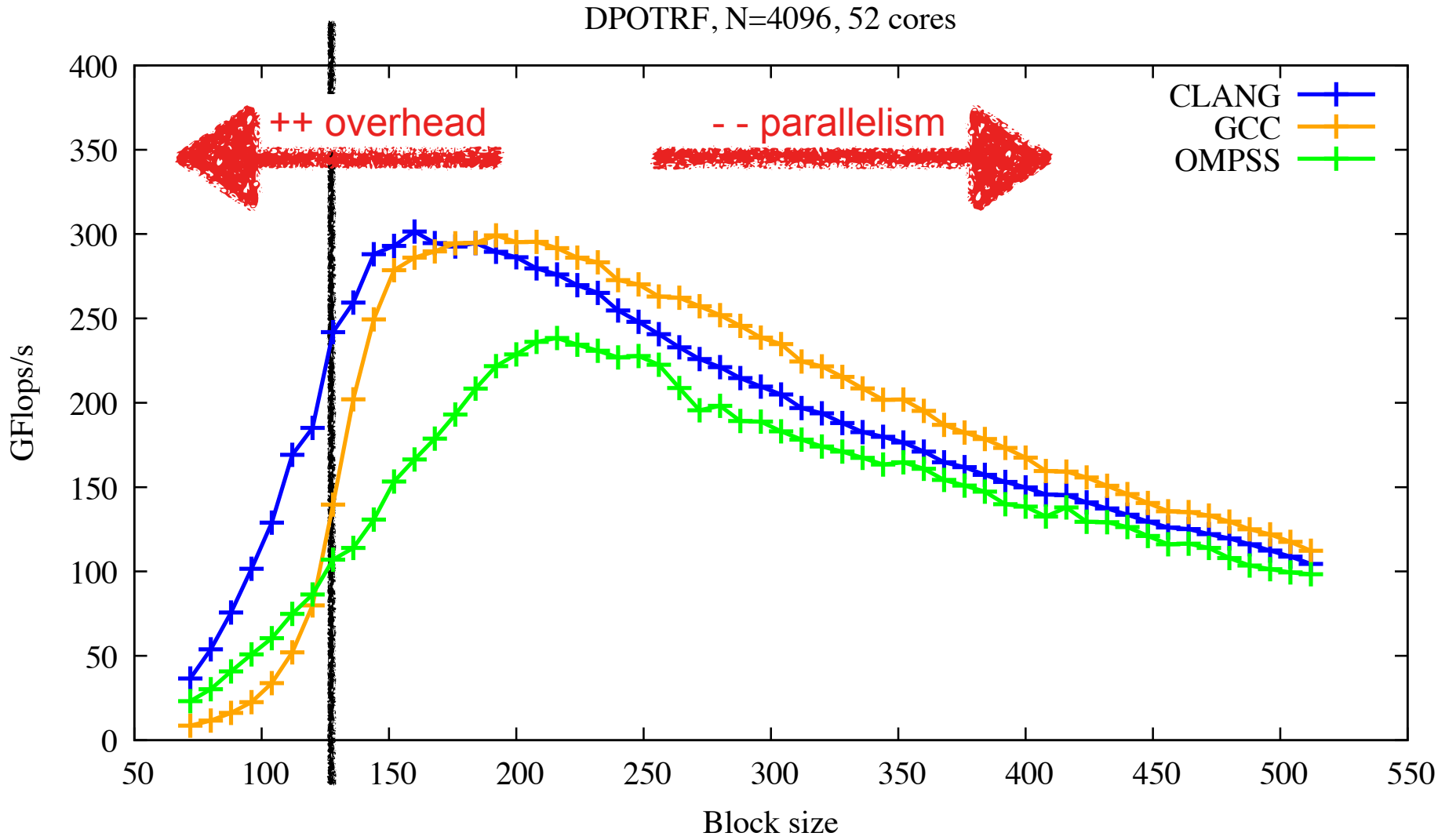
Performance portability ?

Same program (Cholesky factorization), several compilers & runtime libraries

- 192 cœurs SandyBridge



Overhead versus parallelism



How to improve performances ?

Task overhead depends on the runtime implementation

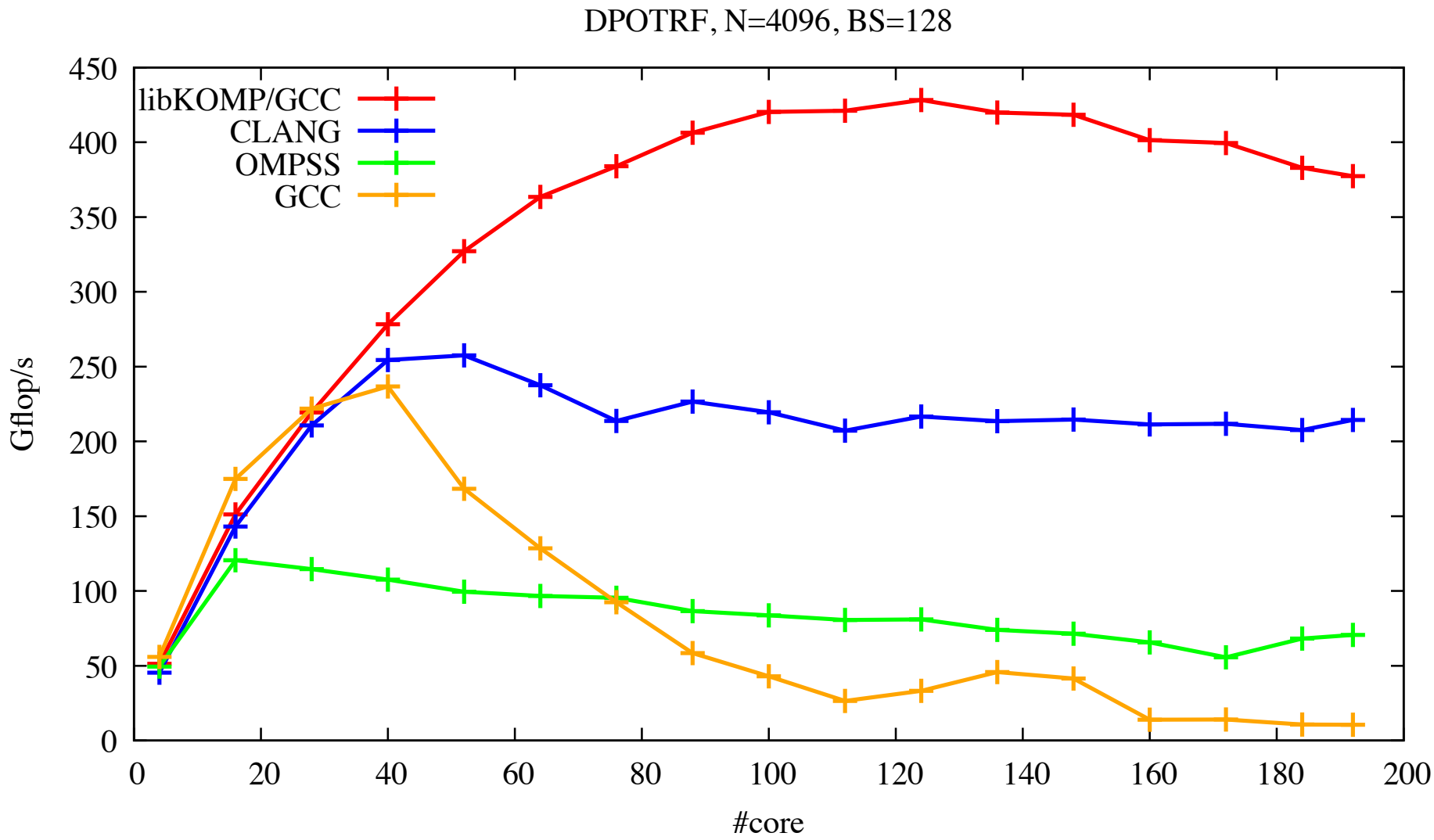
- OMPSS >> GCC >> CLANG
- Lot of research about task programming framework
  - Independent task model
    - Cilk [1996] MIT => Cilk++ from Intel, TBB
    - Wool [2015], …
  - Dependent task model
    - Athapascan [1998] -> Kaapi [2006]
    - StarPU [2010]
    - StarSS[2007]->OMPSS[2011], Swan [2011], …

KAAPI based OpenMP Runtime !

- MOAIS research team [2005-2015]:
  - research axis about « performance guarantees », « write once, run anywhere »
- Kaapi [2006-2015]
  - Successor of Athapascan [1996]
  - Recursive tasks + data flow dependencies
- Aggressive tasks representation
- Support for libGOMP [GCC] since libKOMP [2012]

Speedup with Kaapi libKOMP

Same program (Cholesky factorization), several compilers & runtime libraries



libKOMP

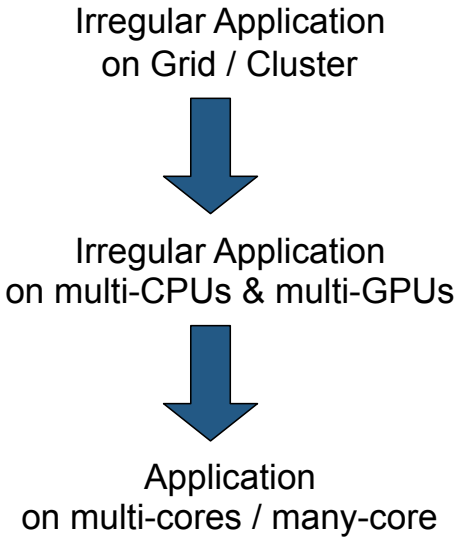
- history
- how to use it
- task scheduling
- loop scheduler

…

Kaapi runtime library

Kaapi [2006] http://kaapi.gforge.inria.fr

- **Recursive** tasks based with data flow dependencies
  - Athapascan [1996]
- Several interfaces for C, C++ and Fortran
  - INRIA RT-0418, INRIA RT-0417, INRIA RT-0429
- Cluster & Grid version [2006-2010]
  - Fault tolerance protocols for data flow program
- Multi-CPUs & Multi-GPUs [2008-2015]
  - model agnostic scheduling (work stealing + heuristic)
- Work stealing scheduler + heuristics
  - for homogeneous architecture = multi core, many core
  - for heterogeneous architecture
    - up to **8 GPUs** [IPDPS 2013, PARCO 2015], NUMA architecture
  - request combining protocol
- Low overhead in task creation
  - Stack based allocation
  - Lazy graph representation, ready tasks computed during steal operation
- Adaptive task model
  - Used to implement parallel loop with very small overhead
- Tracing framework to capture events during runtime
  - software & hardware (PAPI) performance counters
  - generation of gantt for Paje or Vite



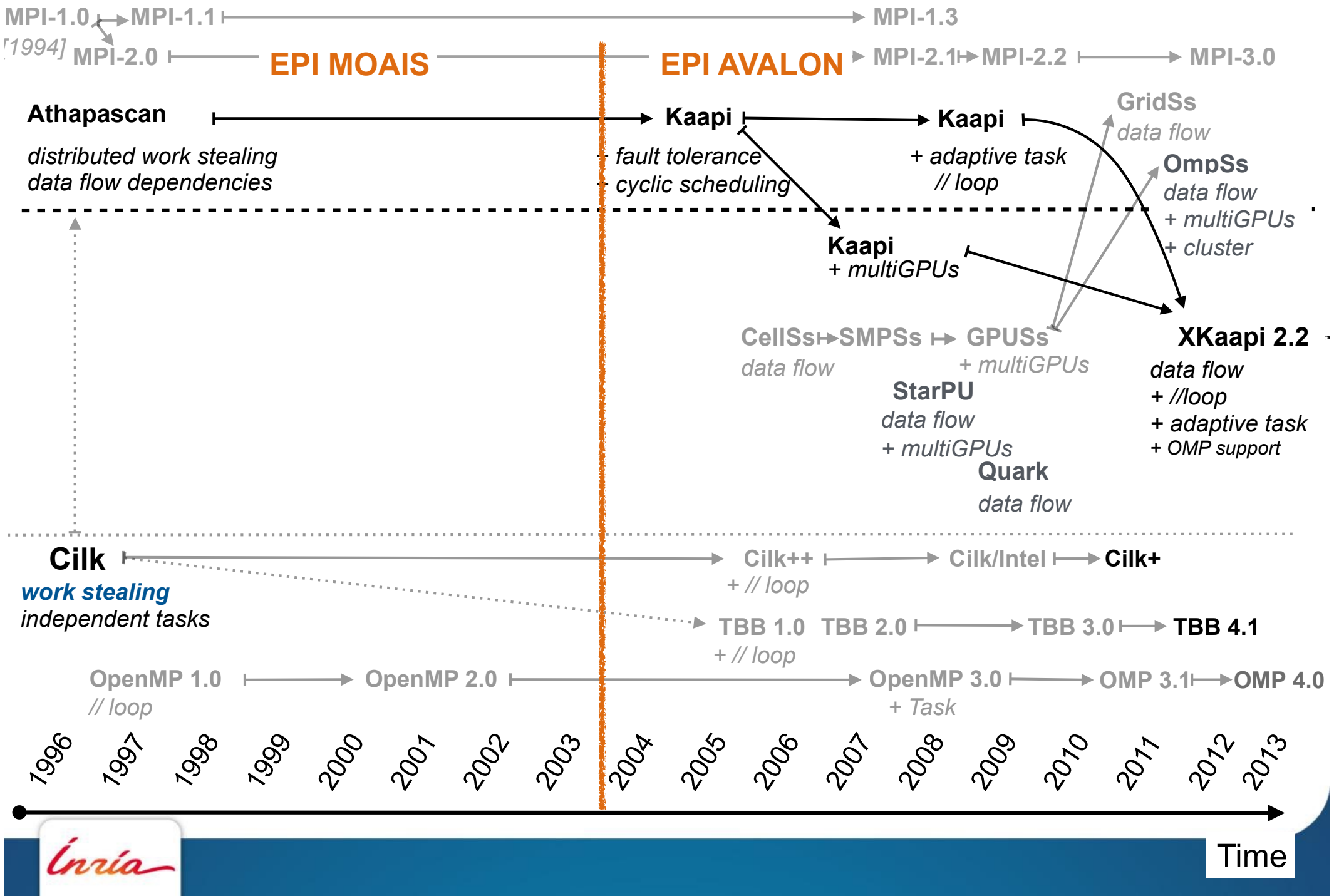
➡ Good performance

- Task: Linear Algebra, Clément Pernet talk
- Loop: EPX (Europlexus) experience, Vincent Faucher talk

BUT non standard interfaces

- Specific Kaapi Fortran API required to be maintained

Related work



libKOMP [IWOMP 2012->]

libKOMP = a binary compatible OpenMP runtime on top of KAAPI

- with libGOMP (GCC) and libOMP (Intel/Clang)
- available with Kaapi distribution
  - autotools: configure; make; make install
- once your program is compiled with gcc, run it with libKOMP !

• **With GCC**

```
thierry$ gcc  -fopenmp ex1.c
thierry$ ./a.out
Result is: 0.000000 1.000000 4.000000 9.000000 16.000000
thierry$
```

• **With GCC + libKOMP**

```
thierry$ gcc  -fopenmp ex1.c
thierry$ ~/kaapi/bin/komp-run ./a.out
Result is: 0.000000 1.000000 4.000000 9.000000 16.000000
thierry$
```

Ready for OpenMP-4.0 dependent task [2013]

- with dependent task libKOMP is very similar to Kaapi programming model

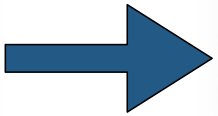
OpenMP performances ~ KAAPI performances

- except at very fine task grain
- [Clément Pernet talk]

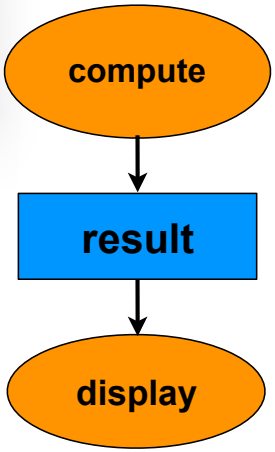
Task creation

Task creation == OpenMP code annotation = Tasks with data dependencies

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

   /* data result is consumed */
  display( size, result );
}
```



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

#pragma omp task depend(in: result)
  display( size, result );
}
```



- Meta data shared by same type of tasks
  - access to memory for each parameters, number & type of parameters,…

Compile with GCC and run with Kaapi !

Illustration

```c
#include <stdio.h>
#include <stdlib.h>

/* write result[0..size-1] */
void compute(int size, double* result)
{
  int i;
  for (i=0; i<size; ++i)
   result[i] = (double)(i*i);
}

/* read result[0..size-1] */
void display( int size, const double* result )
{
  int i;
  printf("Result is:");
  for (i=0; i<size; ++i)
    printf(" %f", result[i]);
  printf("\n");
}

int main()
{
  int size = 5;
  double* result = malloc( sizeof(double)*size );

  #pragma omp parallel
  #pragma omp master
  {
    #pragma omp task depend(out: result[0:size])
    compute( size, result );

    #pragma omp task depend(in: result[0:size])
    display( size, result );
  }

 return 0;
}
```

- With GCC

```
thierry$ gcc  -fopenmp ex1.c
thierry$ ./a.out
Result is: 0.000000 1.000000 4.000000 9.000000 16.000000
thierry$
```

- With GCC + libKOMP

```
thierry$ gcc  -fopenmp ex1.c
thierry$ ~/kaapi/bin/komp-run ./a.out
Result is: 0.000000 1.000000 4.000000 9.000000 16.000000
thierry$
```

- With clang or icc + libKOMP

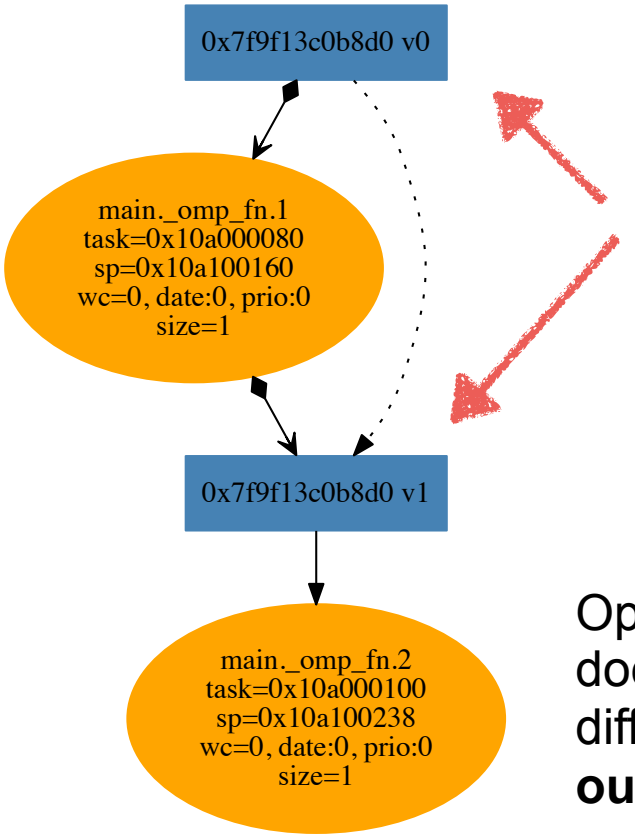
```
thierry$ clang -fopenmp ex1.c
thierry$ LD_LIBRARY_PATH=~/kaapi/bin/lib/xkaapi ./a.out
Result is: 0.000000 1.000000 4.000000 9.000000 16.000000
thierry$
```

Illustration

```c
#include <stdio.h>
#include <stdlib.h>

/* write result[0..size-1] */
void compute(int size, double* result)
{
  int i;
  for (i=0; i<size; ++i)
   result[i] = (double)(i*i);
}

/* read result[0..size-1] */
void display( int size, const double* result )
{
  int i;
  printf("Result is:");
  for (i=0; i<size; ++i)
    printf(" %f", result[i]);
  printf("\n");
}

int main()
{
  int size = 5;
  double* result = malloc( sizeof(double)*size );

  #pragma omp parallel
  #pragma omp master
  {
    #pragma omp task depend(out: result[0:size])
    compute( size, result );

    #pragma omp task depend(in: result[0:size])
    display( size, result );
  }

 return 0;
}
```

- **Dump of the graph**

```
thierry$ gcc  -fopenmp ex1.c
thierry$ KAAPI_DUMP_GRAPH=1 ~/kaapi/bin/komp-run ./a.out
Result is: 0.000000 1.000000 4.000000 9.000000 16.000000
thierry$ dot -Tpdf /tmp/graph_sync_rdlist.thierry.0.dot
> graph.pdf
thierry$
```



OpenMP
does not really make
difference between
out and **inout** depend

KAAPI programming = OpenMP programming

KAAPI task model

- data with data flow dependencies ➡ OpenMP dependent task
- adaptive task ➡ OpenMP parallel loop and task loop

What is missing ?

- C++ interface
  - multi-CPUs / multi-GPUs
- Ultra low overhead in task creation
  - overhead in OpenMP task: ICVs + protocol in GCC
- More rich access modes & simpler semantics
  - Enforced sequential semantics
    read always returns value written by the last task or instruction
  - Concurrent write access mode
    reduction, concurrent write
  - Postponed access mode
    parallel graph construction with sequential semantics
  - No parallel regions composition problem !

Cholesky using the C++ interface

```
struct TaskCholesky: public ka::Task<2>::Signature<
  int,                                          /* BS */
  ka::RPWP<ka::range2d<double,ColMajor> >       /* A */
>{};

template<>
struct TaskBodyCPU<TaskCholesky> {
  void operator()(
         int BS,
         ka::range2d_rpwp<double,ColMajor> A
  )
  {
    int N = A->dim(0);
    for (int k= 0; k < N; k+=BS)
    {
      ka::rangeindex rk( k, k+BS );
      ka::Spawn<TaskDPOTRFBlas>()( A(rk,rk) );
      for (int m=k+BS; m < N; m += BS)
      {
        ka::rangeindex rm(m, m+BS);
        ka::Spawn<TaskDTRSMBlas>() ( A(rk,rk), A(rm,rk) );
      }
      for (int m=k+BS; m < N; m += BS)
      {
        ka::rangeindex rm(m, m+BS);
        ka::Spawn<TaskDSYRKBlas>()( A(rm,rk), A(rm,rm) );

        for (int n=k+BS; n < m; n += BS) {
          ka::rangeindex rn(n, n+BS);
          ka::Spawn<TaskDGEMMBlas>()( A(rm,rk), A(rn,rk), A(rm,rn) );
        }
      }
    }
    ka::Sync();
  }
};
```

```
struct TaskDTRSMBlas : public ka::Task<2>::Signature<
 ka::R<ka::range2d<double,ColMajor> >   /* Akk */
 ka::RW<ka::range2d<double,ColMajor> >  /* Amk */
> {};

template<>
struct TaskBodyCPU< TaskDTRSMBlas > {
  void operator() ( ka::range2d_r<double,ColMajor> > Akk,
                    ka::range2d_rw<double,ColMajor> > Amk )
  {
    /* call BLAS */
  }
};

template<>
struct TaskBodyCPU< TaskDTRSMBlas > {
  void operator() ( ka::range2d_r<double,ColMajor> > Akk,
                    ka::range2d_rw<double,ColMajor> > Amk )
  {
    /* call CUBLAS */
  }
};
```

Cholesky with OpenMP dependent task

```
#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
}
```

Overview of speedup / OpenMP Cholesky

KASTORS benchmark

- http://kastors.gforge.inria.fr

OpenMP experimentations with

- GCC-5.2
- CLANG-3.8
- OMPSS-16.06.1 (nanox-0.10.1, mcxx-2.0.0) using the affinity scheduler

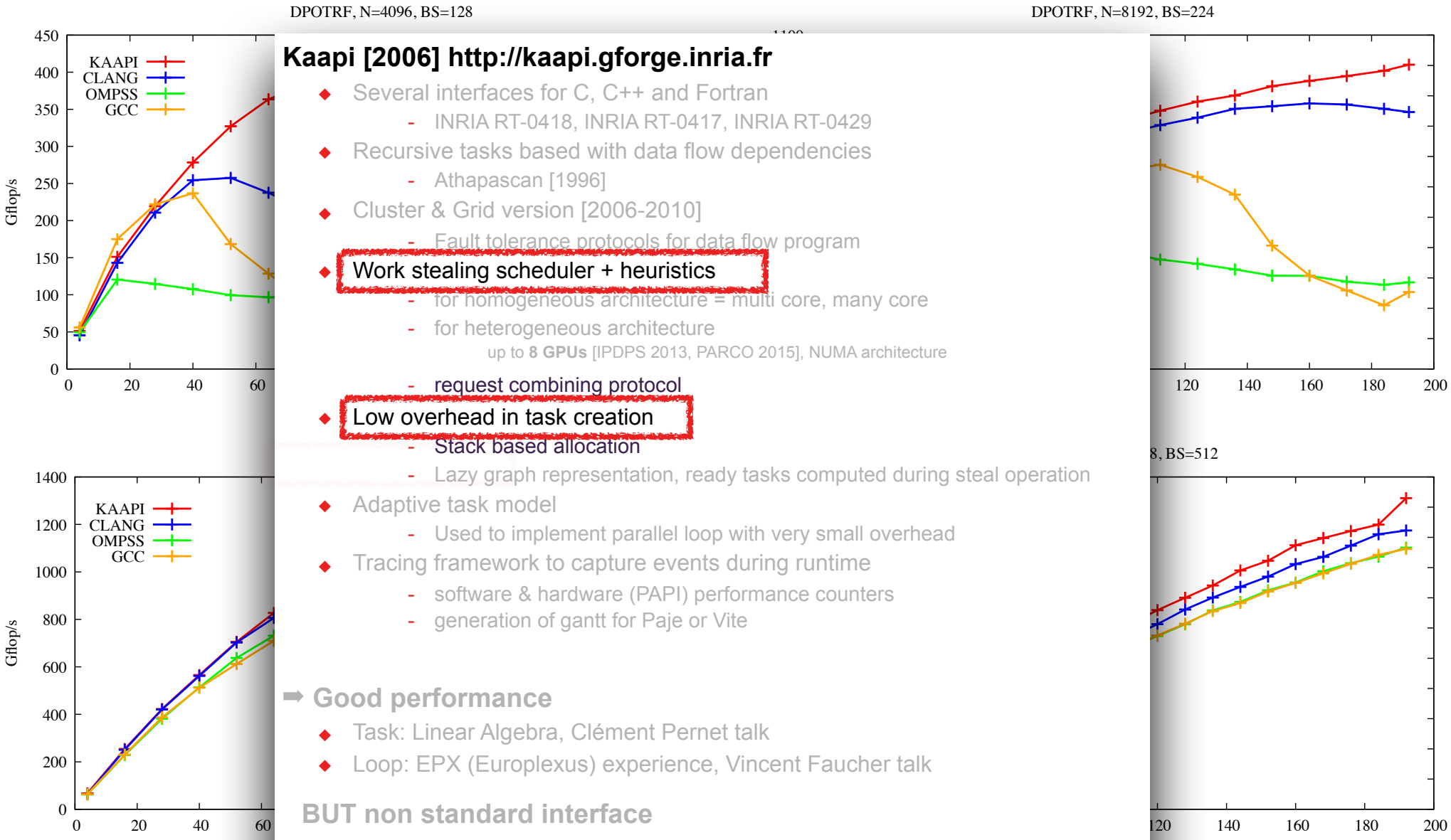
```
NX_ARGS="--verbose=true --schedule=affinity --affinity-use-immediate-successor"
```

- XKAAPI 3.1
  - using libGOMP compatibility (code compiled with GCC and executed with libKOMP)
- PLASMA 2.8
  - maximal performance = static scheduler ~ 2200Gflops
- OpenBLAS 0.2.19

idchire = UV 2000 SGI architecture

- Xeon SandyBridge 8 cores
- 192 cores
- 24 NUMA nodes

Weak scalability
(Dense Cholesky factorization)



Kaapi [2006] http://kaapi.gforge.inria.fr

- Several interfaces for C, C++ and Fortran
  - INRIA RT-0418, INRIA RT-0417, INRIA RT-0429
- Recursive tasks based with data flow dependencies
  - Athapascan [1996]
- Cluster & Grid version [2006-2010]
  - Fault tolerance protocols for data flow program
- Work stealing scheduler + heuristics
  - for homogeneous architecture = multi core, many core
  - for heterogeneous architecture
    - up to **8 GPUs** [IPDPS 2013, PARCO 2015], NUMA architecture
  - request combining protocol
- Low overhead in task creation
  - Stack based allocation
  - Lazy graph representation, ready tasks computed during steal operation
- Adaptive task model
  - Used to implement parallel loop with very small overhead
- Tracing framework to capture events during runtime
  - software & hardware (PAPI) performance counters
  - generation of gantt for Paje or Vite

➡ **Good performance**

- Task: Linear Algebra, Clément Pernet talk
- Loop: EPX (Europlexus) experience, Vincent Faucher talk

BUT non standard interface

- Specific Kaapi Fortran API required to be maintained

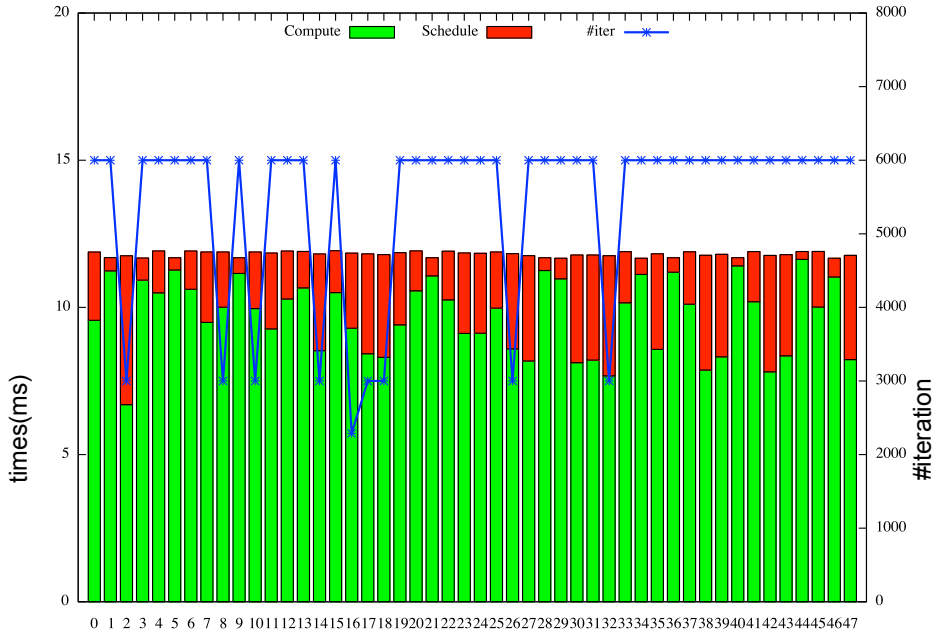
Loop scheduler for irregular workload

Problem

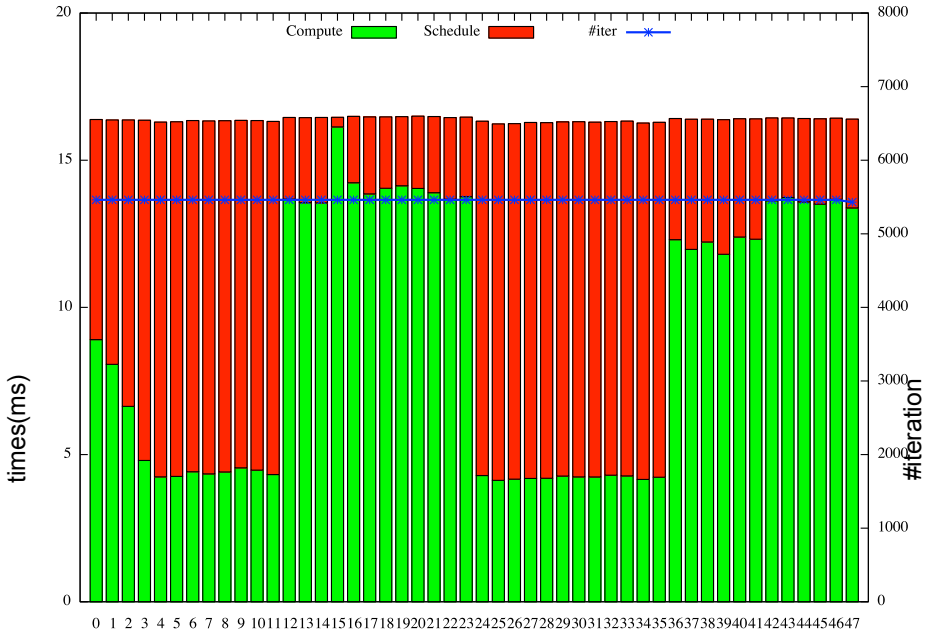
```
for (int timestep = 0; timestep < T; ++ timestep)
  #pragma omp parallel for [schedule(static|dynamic|guided)]
  for (int i = 0; i < N; ++i)
    compute(i)
```

- Irregular workload among the iterations !
- Irregular workload among the timestep iterations !

Classical OpenMP loop scheduler



dynamic scheduler
best chunk_size=3000



static scheduler

Time spent in computation

Time spent in scheduling (overhead)

Number of iterations executed by each thread

Static scheduler unable to balance the load

Better load balancing with dynamic scheduler, but poor affinity control

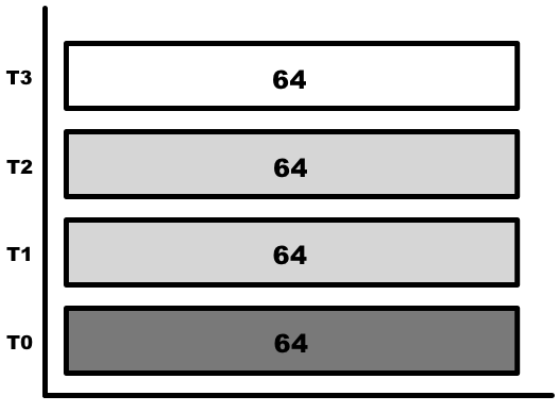
Adaptive Loop scheduler in libKOMP

Adapt to workload irregularity at runtime

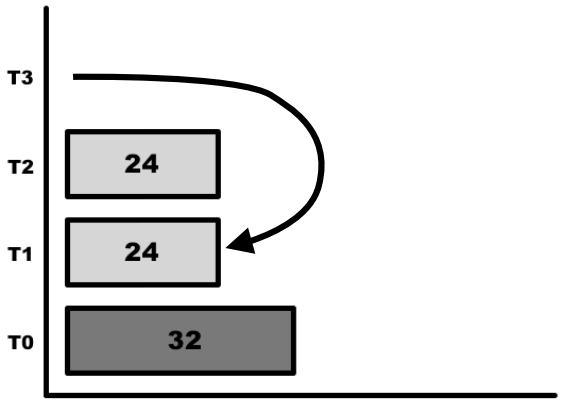
- make possible arbitrary splits of iterations assigned to a thread *at runtime*
- idle threads steal iterations from busy threads
- Try to keep spatial and temporal locality

Based on work stealing [Cilk]

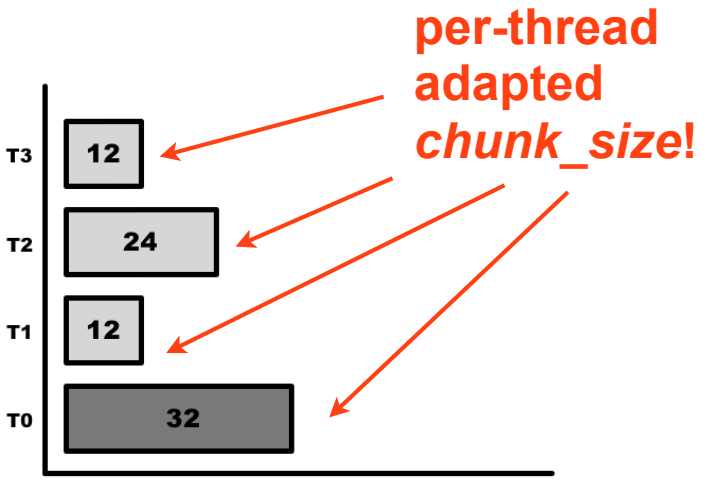
- Cilk work queue : range of iterations [a,b)
  - steal a task = steal a sub range [a',b')
- T.H.E like protocol to steal iterations
  - atomic read, atomic write and memory barrier (optional)



Initial workload distribution

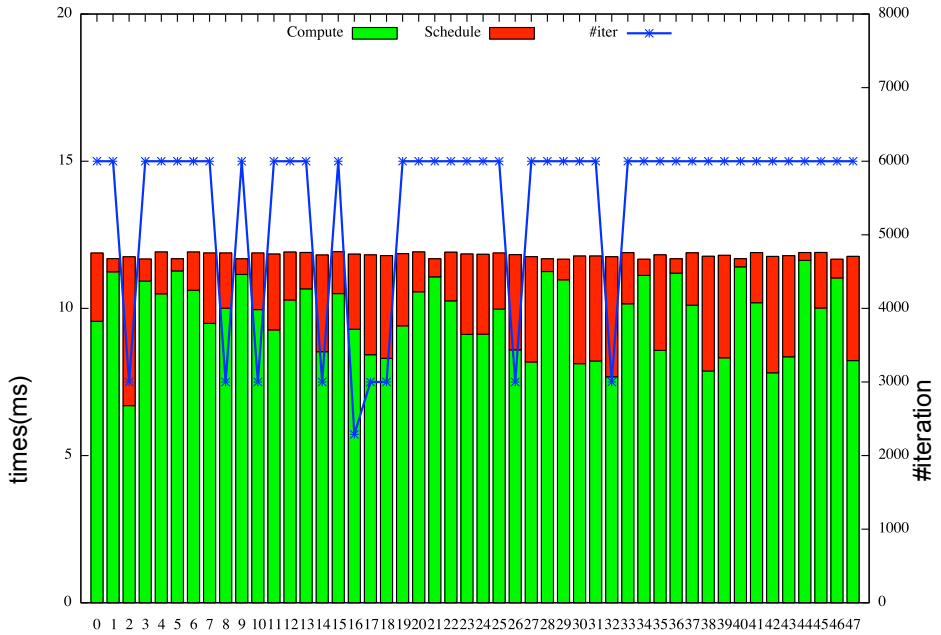


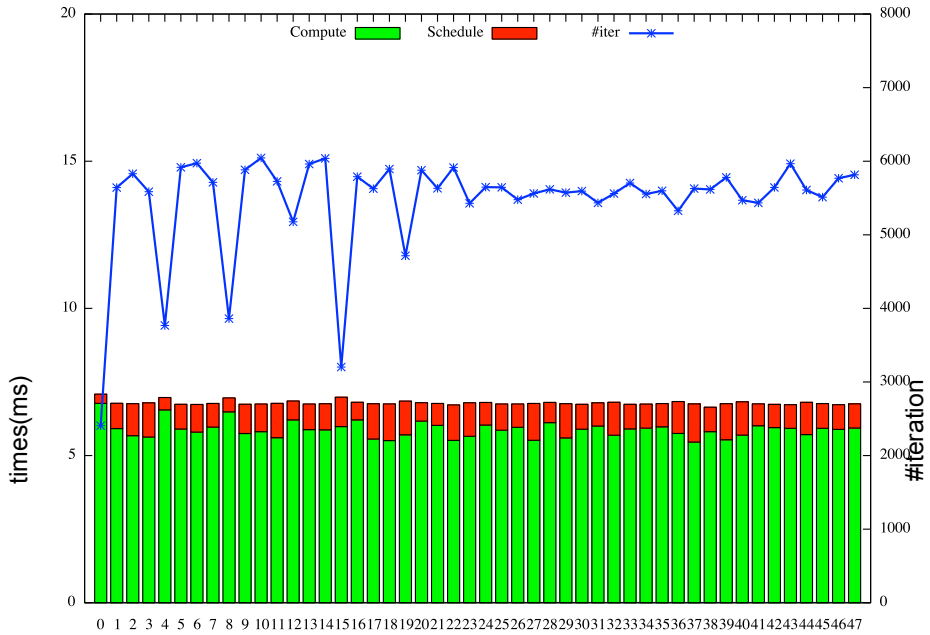
T3 has completed its range and starts stealing



T3 stole half of the remaining iteration of T1

Adaptive loop scheduler [IWOMP 2013]





dynamic scheduler
best chunk_size=3000

adaptive scheduler

Time spent in computation

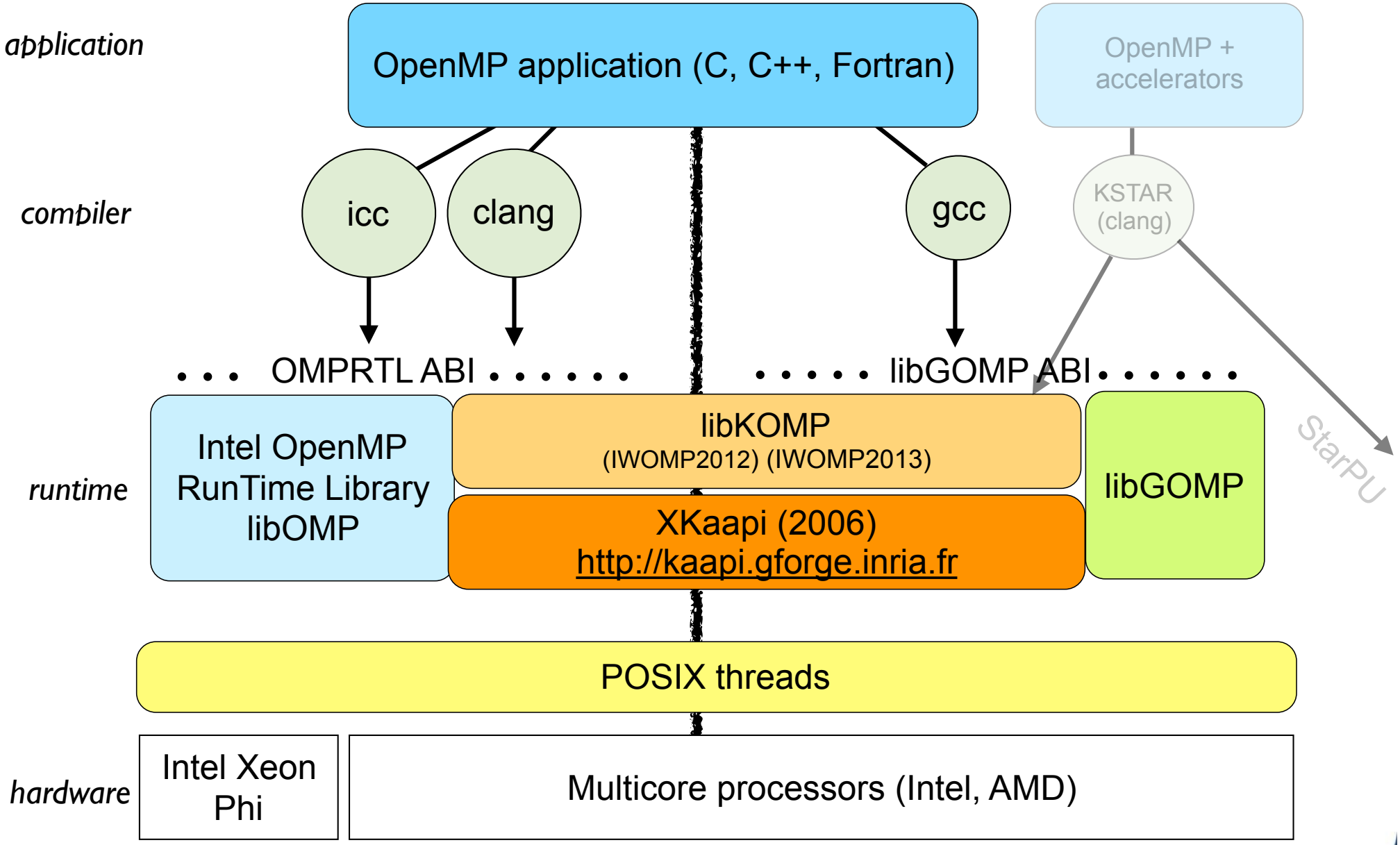
Time spent in scheduling (overhead)

Number of iterations executed by each thread

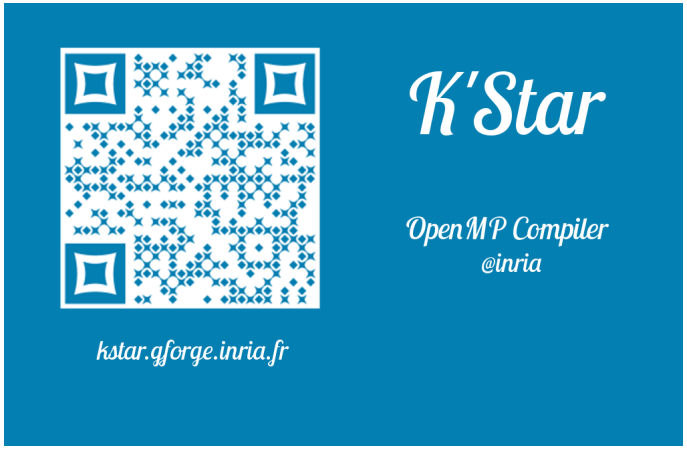
Adaptive scheduler: good load balancing + good locality => smaller overheads

M. Durand, F. Broquedis, T. Gautier, B. Raffin. An Efficient OpenMP Loop Scheduler for Irregular Applications on Large-Scale NUMA Machines. Proceedings of the 9th International Conference on OpenMP in a Heterogeneous World (IWOMP), 8122:141-155, Lecture Notes in Computer Science, Canberra, Australia, 2013.

LibKOMP overview based on Kaapi



KStar' compiler



ADT INRIA 2013-2015

- Coordinator T. Gautier [Grenoble/MOAIS -> Lyon/AVALON]
- O. Aumage co-coordinator [Bordeaux/RUNTIME -> STORM]

Goal

- C and C++ compiler OpenMP -> Kaapi or StarPU runtimes
- multi-GPUs

KStar' Prototype compiler where necessary before compiler for 4.0 features

- dependent task
- base clang + rewriter, parsing integrated by Intel Group
- extensions for concurrent write (Kaapi), commute (StarPU), affinity (Kaapi)

But

- robustness ?
- human resource for supporting compiler development
- no fortran support

Development stopped for KAAPI

libKOMP experiences

Good performance in managing fine grain OpenMP Task

- Almost fully compatible with non offloading part of libGOMP (OpenMP-4.1)
  - missing support for taskloop, cancellation of taskgroup, non standard mix of pthread + libGOMP
  - ~ 30 failures / 2144 gcc tests

OpenMP extensions is possible at runtime library only

- affinity, taskname
- new ICV

Code summary

	Kaapi 3.1	libGOMP gcc 5.2	libOMP Intel	StarPU 1.3 (K'Star
Source size - tarball (tgz)	1.5MBytes	720KBytes	4.9MBytes	8MBytes
library size	246KB (kaapi) 75KB (libKOMP) 83KB (libGOMP)	1MB (libGOMP)	964KB (libOMP & libGOMP)	6.75MB (libstarpu-1.3)
archicture	multi/many core	OpenMP + offloading	multi/many core	multi/many core heterogeneous

Bad news

KAAPI development is stopped

- [except support for application & libKOMP]

~~Bad~~ news

KAAPI development is stopped

- [except support for application & libKOMP]

but libKOMP continues

A new version of libKOMP

- enhanced features

Dilemma

How to have…

- Robust compiler ?
- Support for C, C++, Fortran ?
- Capacity to extend runtime / language construction ?

At low human cost ?

Solution: runtime library approach = libKOMP !

Basic Idea: reuse existing code of a runtime

- libOMP was choose !
- Focus on specific point inside a runtime
  - e.g. task scheduling = 3 main functions in Intel Runtime
- extensions to OpenMP standard are integrated as
  - an *OpenMP Runtime library functions*, e.g. omp.h / omp_get_num_threads()
  - compiler extension: take Clang and generate runtime call to libKOMP

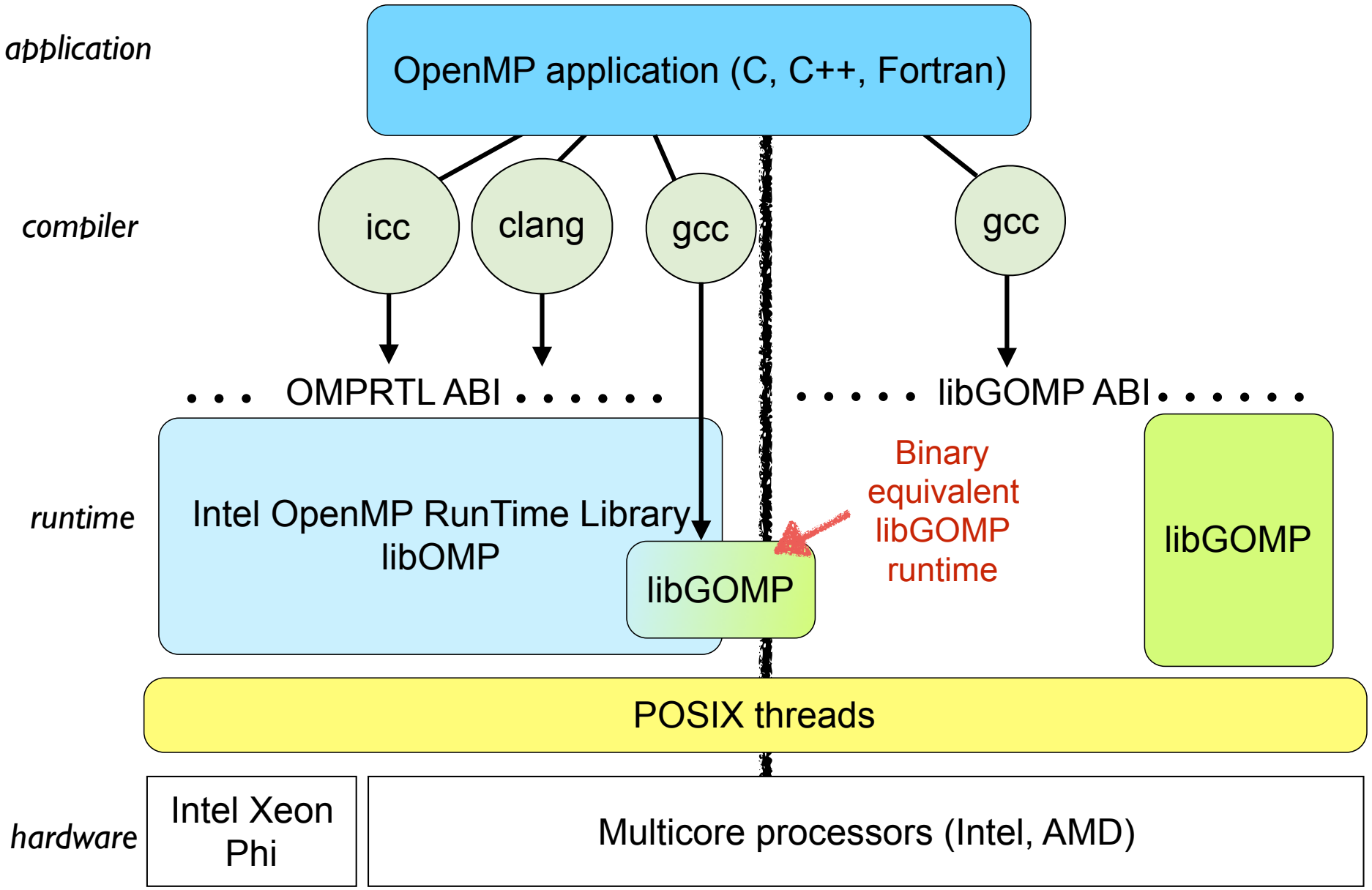
Having a robust & high quality OpenMP runtime

- For application developers
- For demonstrating our proposition

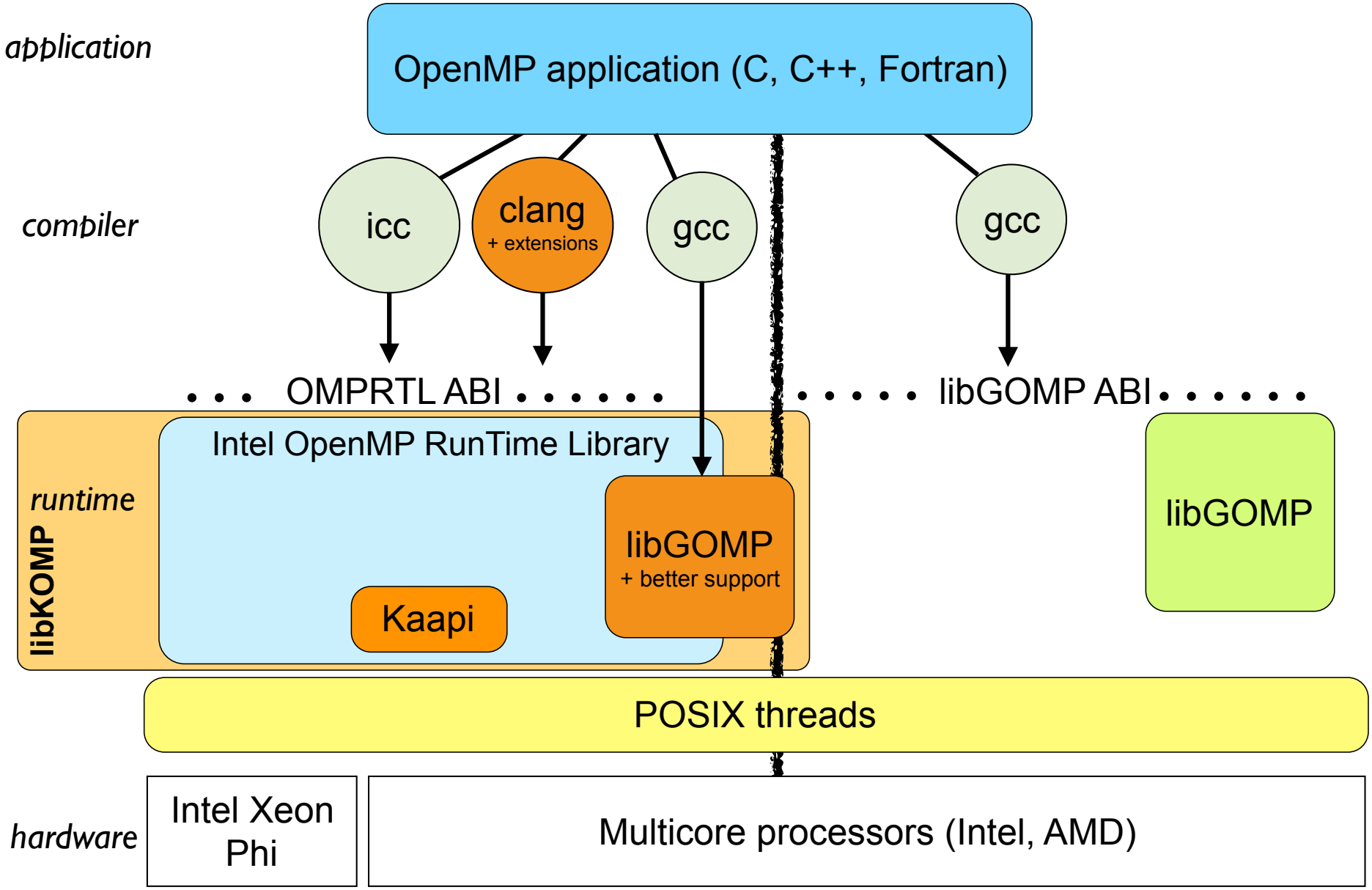
Making possible research

- affinity & locality aware scheduling
- resource agnostic scheduling
- energy efficiency
- scheduler loop extension

New libKOMP


application
OpenMP application (C, C++, Fortran)
compiler
icc
clang
gcc
gcc
OMPRTL ABI
libGOMP ABI
runtime
Intel OpenMP RunTime Library
libOMP
libGOMP
Binary
equivalent
libGOMP
runtime
libGOMP
POSIX threads
hardware
Intel Xeon
Phi
Multicore processors (Intel, AMD)

New libKOMP


application
OpenMP application (C, C++, Fortran)
compiler
icc
clang
+ extensions
gcc
gcc
OMPRTL ABI
libGOMP ABI
runtime
libKOMP
Intel OpenMP RunTime Library
libGOMP
+ better support
Kaapi
libGOMP
POSIX threads
hardware
Intel Xeon Phi
Multicore processors (Intel, AMD)

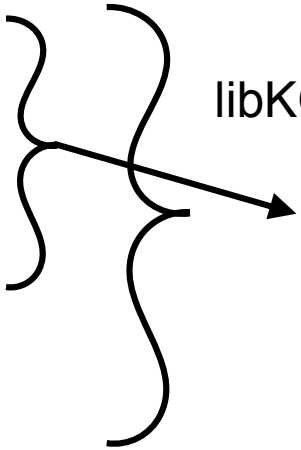
New libKOMP = parts of KAAPI into Intel libOMP

Why Intel Runtime ?

- Intel Runtime selected for several reasons
  - target of icc and ifort compiler
  - good algorithms and data structures basis
    - distributed work queues + (naive) work stealing algorithm
    - scalable barrier algorithm
    - small overhead in task creation
    - integration of a lot of loop schedulers (~ 15)
  - initial good support for libGOMP / GCC application
  - initial support for OMPT (interface for integrating tracing tools with OpenMP runtime)

Integration of KAAPI solutions

- task scheduling algorithm
  - affinity scheduler
- task concurrent write & commute
- tracing tools
- support for GCC dependent task
- lightweight task representation
- adaptive task model
  - specific loop scheduler



libKOMP@ http://gitlab.inria.fr/openmp/libkomp

What you will download today
- ongoing work!

OpenMP extensions

Runtime extensions

- *better algorithms inside the runtime*
  - *Kaapi task representation and scheduling*
- with runtime application interface through OMP runtime function calls
  - e.g. omp_get_thread_num
  - **omp_set_task_affinity**
  - **omp_set_task_name**
  - new loop scheduler through **omp_set_schedule**
- do not require compiler support
- bindings in C, C++ and Fortran
- **libKOMP =** https://gitlab.inria.fr/openmp/libkomp

Compiler extensions

- taskname
- concurrent write access
- commute access [from StarPU]
- **affinity**
- Enhanced **CLANG compiler =** https://gitlab.inria.fr/openmp/clang

libKOMP configuration & installation

Source access

- Git access: https://gitlab.inria.fr/openmp/libkomp
  - branch « libkomp »
- download tarball

```
> thierry$ git clone https://gitlab.inria.fr/openmp/libkomp.git
Clonage dans 'libkomp'...
…
> thierry$ ll libkomp/
total 48
-rw-r--r--   1 thierry  wheel   150 18 jan 14:52 CMakeLists.txt
-rw-r--r--   1 thierry  wheel  1568 18 jan 14:52 CREDITS.txt
-rw-r--r--   1 thierry  wheel  9282 18 jan 14:52 LICENSE.txt
-rw-r--r--   1 thierry  wheel  1613 18 jan 14:52 README.md
drwxr-xr-x   6 thierry  wheel   204 18 jan 14:52 offload
drwxr-xr-x  10 thierry  wheel   340 18 jan 14:52 runtime
drwxr-xr-x  30 thierry  wheel  1020 18 jan 14:52 testsuite
drwxr-xr-x   7 thierry  wheel   238 18 jan 14:52 www
> thierry$ mkdir build; cd build
```

Configuration of libOMP

- cmake ../openmp -D<SPECIFIC OPTION>
  - -DLIBOMP_USE_AFFINITY=true : use affinity scheduler
    Req. -DLIBOMP_USE_HWLOC=true
  - -DLIBOMP_USE_THEQUEUE=true : use T.H.E. original Cilk protocol and tasks' queues
  - -DLIBOMP_USE_THE_AGGREGATION=true : add request combining protocols
  - -DLIBOMP_KAAPI_TRACING=on : compile support for Kaapi tracing.
    Req. -DLIBOMP_HAVE_OMPT_SUPPORT=on
  - -DLIBOMP_USE_PAPI=true : compile Kaapi tracing with Papi support

```
> thierry$ cmake ../libkomp -DLIBOMP_USE_HWLOC=true -DLIBOMP_USE_AFFINITY=true -DLIBOMP_OMPT_SUPPORT=off -DLIBOMP_KAAPI_TRACING=off  \
    -DCMAKE_INSTALL_PREFIX=$HOME/install-release-affinity -DCMAKE_BUILD_TYPE=release
> thierry$ make install -j 8
```

libKOMP configuration & installation

Installation

- prefix specified by « -DCMAKE_INSTALL_PREFIX »

```
> thierry$ make install
make install
[ 10%] Built target libomp-needed-headers
[100%] Built target omp
Install the project...
-- Install configuration: "release"
-- Installing: /home/gautier/install-release-affinity/lib/libomp.so
-- Up-to-date: /home/gautier/install-release-affinity/include/omp.h
```

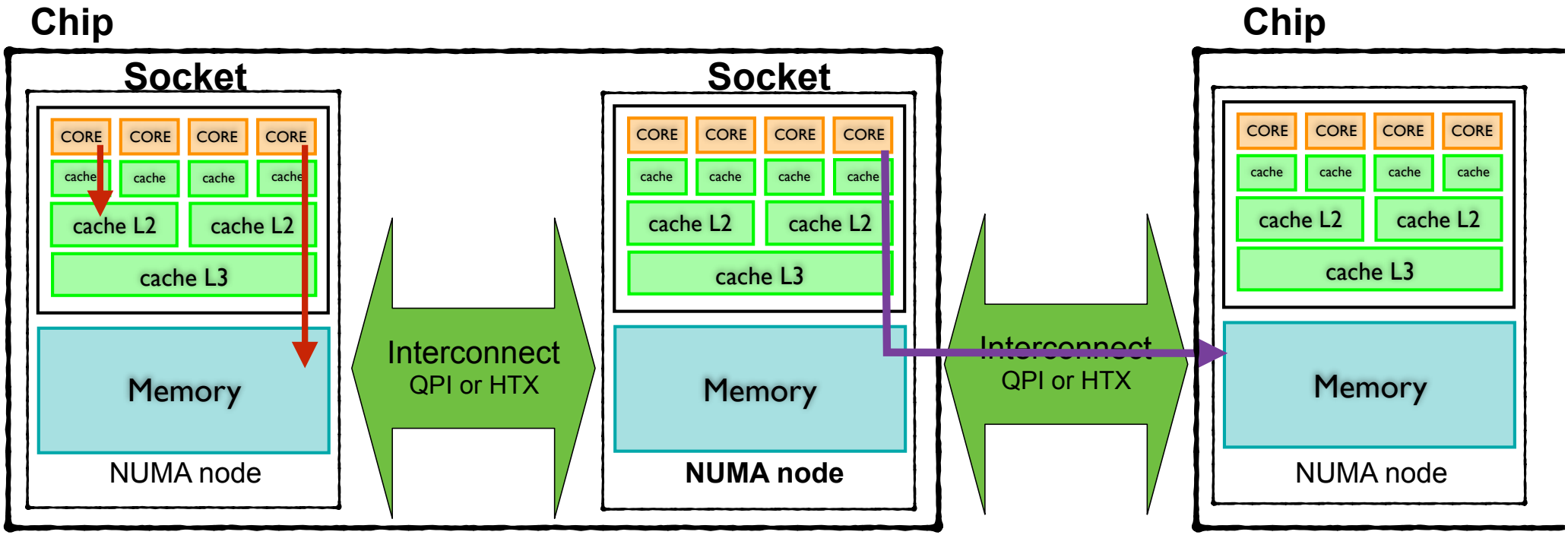
Running program

- Here Kastors Cholesky factorization

```
> thierry$ N=192; LD_LIBRARY_PATH=/home/gautier/LIBOMP_KAAPI/install-release-affinity/lib \
                OMP_DISPLAY_ENV=true OMP_NUM_THREADS=$N OMP_PLACES="cores($N)"  \
                ./dpotrf_taskdep -n 8192 -b 224 -i 10
```

one extension: NUMA affinity

NUMA architecture



Main characteristics

- NUMA (Non Uniform Memory Access)
- big memory banks are slow
- big L3 cache shared by multiple cores

Impacts

- algorithmic
- os/runtime

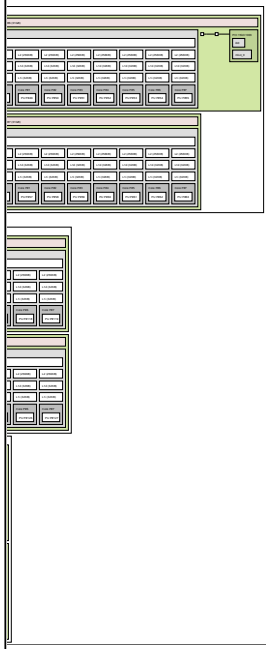
Classical NUMA multicore

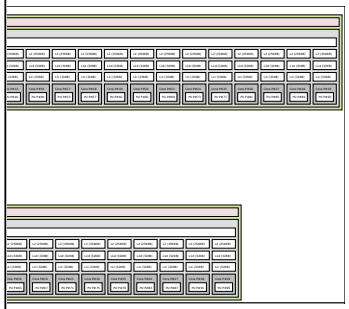
Idchire

- UV 2000 S
- 192 cores
  - sandy
  - 732G
- **24 NUMA**

Brunch

- Dell Power
- 96 cores
  - broad
  - 1.5TB
- 4 NUMA n





libKOMP: controlling the affinity task/data

NUMA architecture

- data are mapped onto NUMA node
- tasks are performed by core

Proposed OpenMP compiler and runtime extension

- Let the user to indicate that task *has better performance if* executed near a data

```
#pragma omp task depend(inout:dA[0:A.mb*A.mb]) affinity( data: dA )
LAPACKE_dpotrf_work(LAPACK_COL_MAJOR, lapack_const(PlasmaUpper), tempkm, dA, ldak);
```

- May be strict or not
- ARB discussion
  - affinity with respect to node (NUMA node) or core
  - supported by our implementation

Data initialization thank to first touch OS policy

- + existing OpenMP guarantee to schedule threads with spatial locality !

Affinity kind

data: <ref>

- ref is memory location, try to schedule the task on a core close to the memory location

core: i

- try to schedule the task on the i-th core of the team

node: i

- try to schedule the task on the i-th node of the team

Implementation

Interface is implemented:

- As a compiler clause: into CLANG
  - download: https://gitlab.inria.fr/openmp/clang
  - use classical llvm + clang installation procedure

```
#pragma omp task depend(inout:dA[0:tempnn*tempmm]) affinity( node: ++cnt, strict )
CORE_dplgsy( bump, tempmm, tempnn, dA, ldam, A.m, m*A.mb, n*A.nb, seed );
```

- As a runtime OpenMP function (accessible through C, C++ with icc or gcc, or Fortran)
  - omp_task_set_affinity(int kind, uintptr_t tag, int strict)

```
omp_set_task_affinity( omp_affinity_node, ++cnt, 1 );
#pragma omp task depend(inout:dA[0:tempnn*tempmm])
CORE_dplgsy( bump, tempmm, tempnn, dA, ldam, A.m, m*A.mb, n*A.nb, seed );
```

Algorithm

- work stealing + heuristic
- described in [Europar2016]
  - 2 queues per resource: one private for the local core; one public
  - push task to the resource define by the clause
  - steal task 1/ on local queues 2/ on remote public queues
  - strictness push task to private queue

Example: yet the Kastors/Cholesky

Main:

```
…
#pragma omp parallel
#pragma omp master
    plasma_pdpotrf_quark(uplo, *descA);
…
```

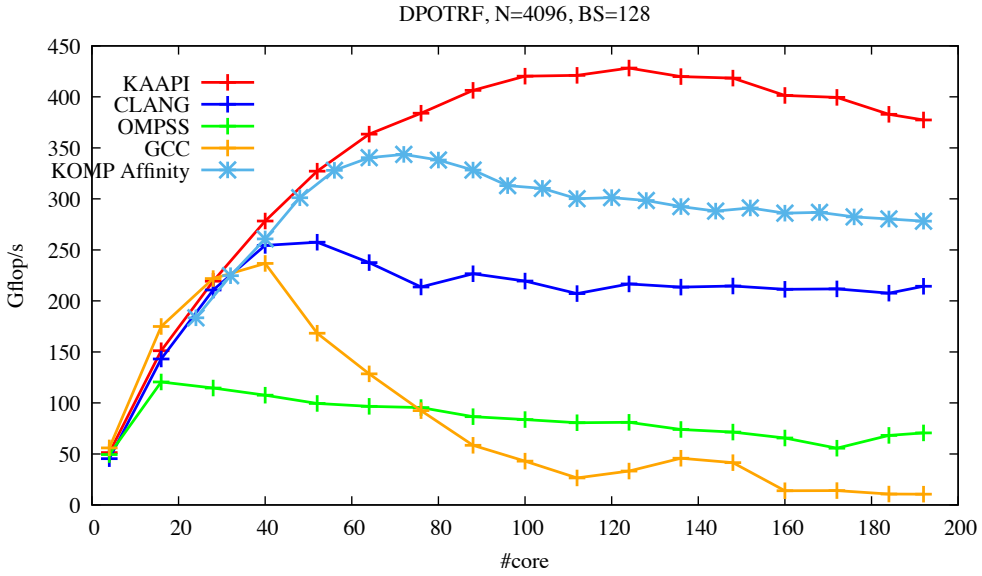
plasma_pdpotrf_quark:

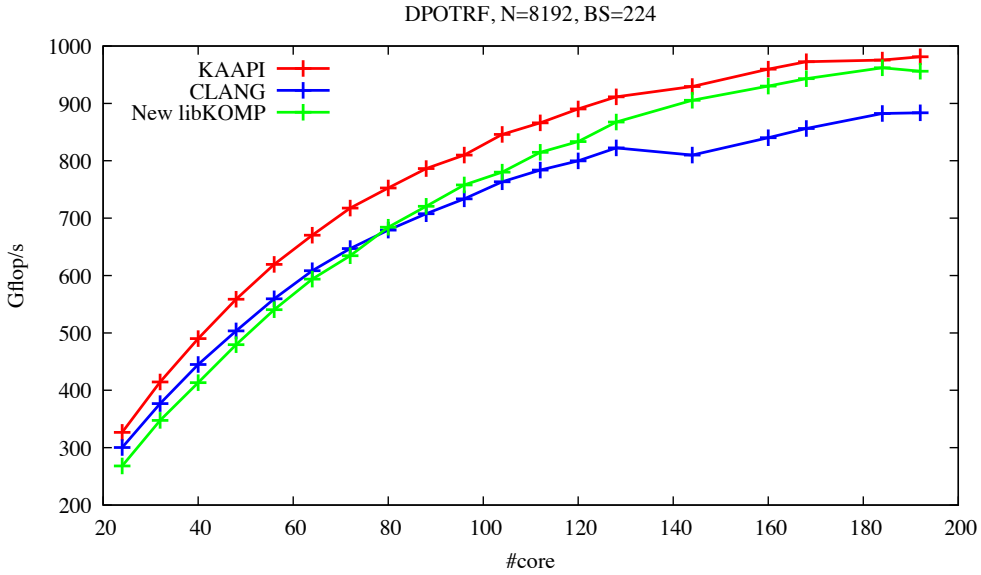
```
…
void plasma_pdplgsy_quark( double bump, PLASMA_desc A, unsigned long long int seed)
{
    int m, n;
    int ldam;
    int tempmm, tempnn;

    for (m = 0; m < A.mt; m++) {
        tempmm = m == A.mt-1 ? A.m-m*A.mb : A.mb;
        ldam = BLKLDD(A, m);

        for (n = 0; n < A.nt; n++) {
            tempnn = n == A.nt-1 ? A.n-n*A.nb : A.nb;
            double *dA = A(m, n);
#pragma omp task depend(out:dA[0:tempnn*tempmm]) affinity( node: m*A.nt+n, 1 )
            CORE_dplgsy( bump, tempmm, tempnn, dA, ldam, A.m, m*A.mb, n*A.nb, seed );
        }
    }
}
…
```

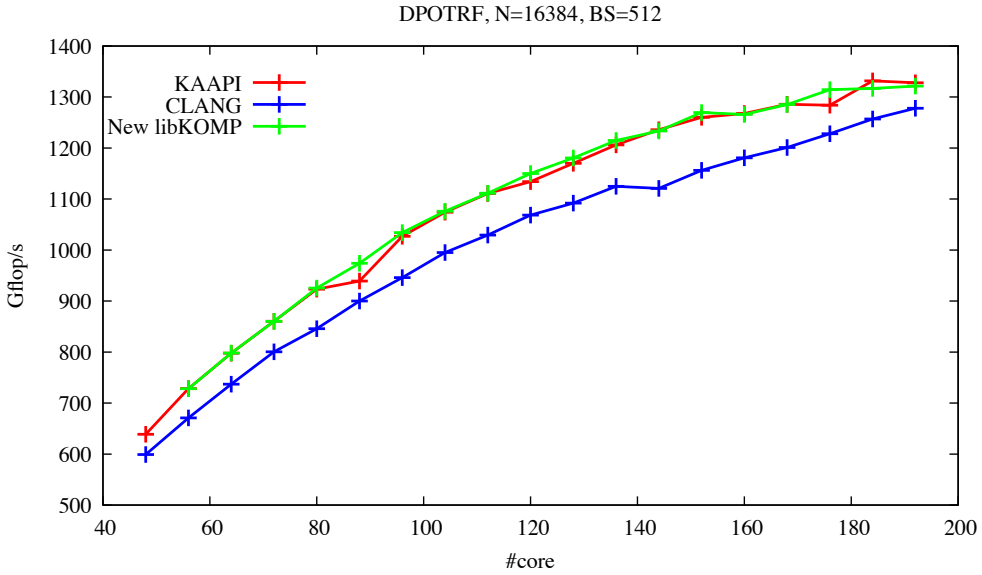
Preliminary results

Affinity heuristic in the Intel Runtime

- scheduling is effective
- Intel Runtime overhead in fine grain task creation yet present
- better if tasks are coarser



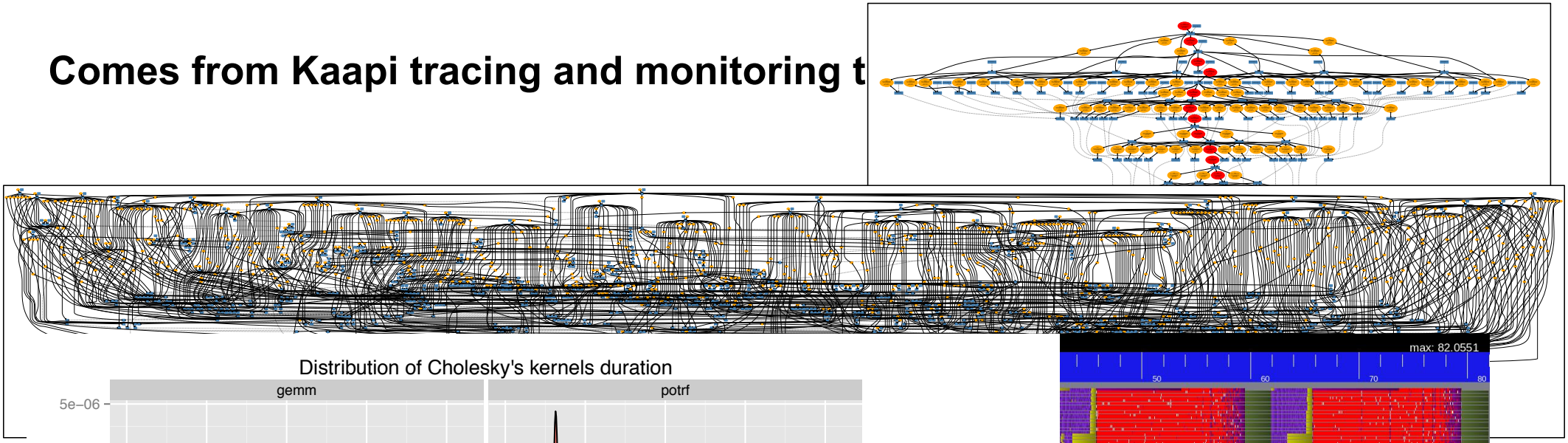


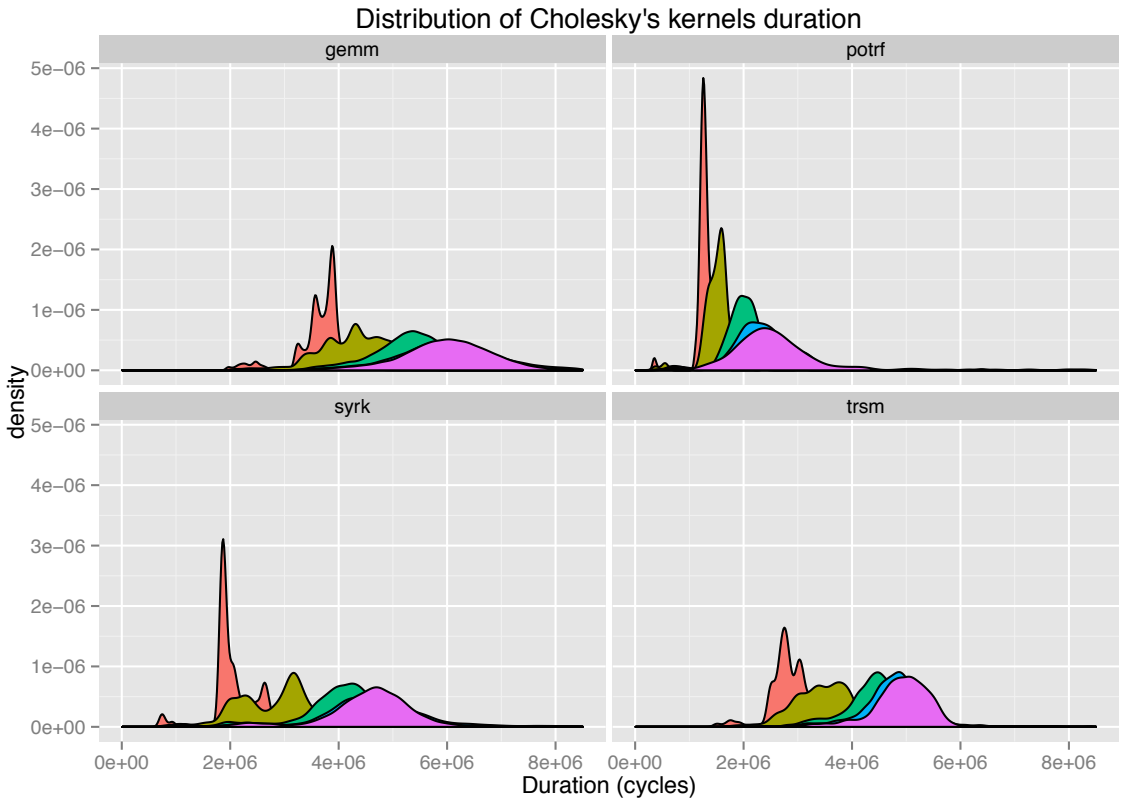


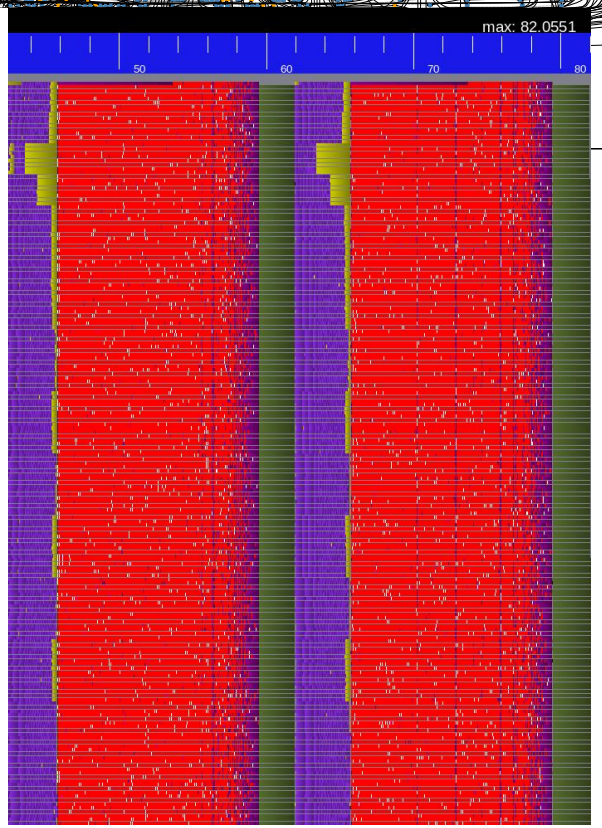
Tracing tool

libKOMP Tracing Tools

Comes from Kaapi tracing and monitoring t







Based on OMPT

Intel libOMP

- has already an implementation of OMPT for external tool
  - see www.openmp.org for specification
  - hpctoolkit has experimental support for OMPT
- we port Kaapi tracing and monitoring tools as specific OMPT tool
  - environment variable control
    - what events are recorded and what performance counters are captured
  - software and hardware performance counters [PAPI]
    - per task performance counters
  - proprietary binary event record files + tools to convert it
    - graph
    - gantt
    - extraction of task performance counters
- Getting started page on http://gitlab.inria.fr/openmp/libkomp

Conclusion

Conclusions

Performance portability among OpenMP solutions is not guaranteed !

- lot of mistake in the design of runtime
  - evident bottleneck when cores increases [GCC]
  - work-stealing in [ICC/CLANG] but with performances far away from the Cilk reference
- NUMA is not (yet) in the OpenMP standard
  - OMP_PLACES is a way to define affinity between threads and hardware cores
  - no « NUMA » compliant allocation => use first touch policy to allocate data on the right place

But solutions exist

- it is possible to have good performance by changing the runtime
  - KAAPI experiences since 2012 through our research OpenMP runtime libKOMP
  - under redesigned for better robustness and integration with compiler
    work with CORSE Inria Team
  - with a full and robust Fortran support !
- it is easy to add application extensions through direct call to the runtime
  - e.g. affinity, taskname

If you have any limitations with OpenMP, we are very interested in studying your problem in order to propose solution

Merci

Backup slides

Overhead of task management [AMD48]

Cilk+

```
long fib(long n)
{
  if (n < 2)
    return (n);
  else {
    long x, y;
    x = cilk_spawn
    y = fib(n - 2);
    cilk_sync;
    return (x + y);
  }
}
```

OpenMP

```
void fibonacci
{
  if (n<2)
    *result =
  else
  {
    long r1,r2
#pragma omp ta
```

TBB

```
struct FibContinuation: public tbb::task {
    long* const sum;
    long x, y;
    FibContinuation( long* sum_ ) : sum(sum_) {}
    tbb::task* execute() {
        *sum = x+y;
        return NULL;
    }
};

struct FibTask: public tbb::task {
    long n;
    long * sum;
    FibTask( const long n_, long * const sum_ ) :
        n(n_), sum(sum_)
    {}
```

```
ntinuation() ) FibContinuation(sum);
allocate_child() ) FibTask(n-1,&c.y);
);

two children".
```

Kaapi

```
void fibonacci(long* result, const long n)
{
  if (n<2)
    *result = n;
  else
  {
    long r1,r2;
#pragma kaapi task write(&r1)
    fibonacci( &r1, n-1 );
    fibonacci( &r2, n-2 );
#pragma kaapi sync
    *result = r1 + r2;
  }
}
```

Overhead of task management [AMD48]

Fibonacci (35) naive recursive computation

Serial	Cilk+	TBB (4.0)	OpenMP	XKaapi
0.0905s (x 1)	1.063 (x 11.7)	2.356s (x 26)	2.429s (x 27)	0.728s **(x 8)**

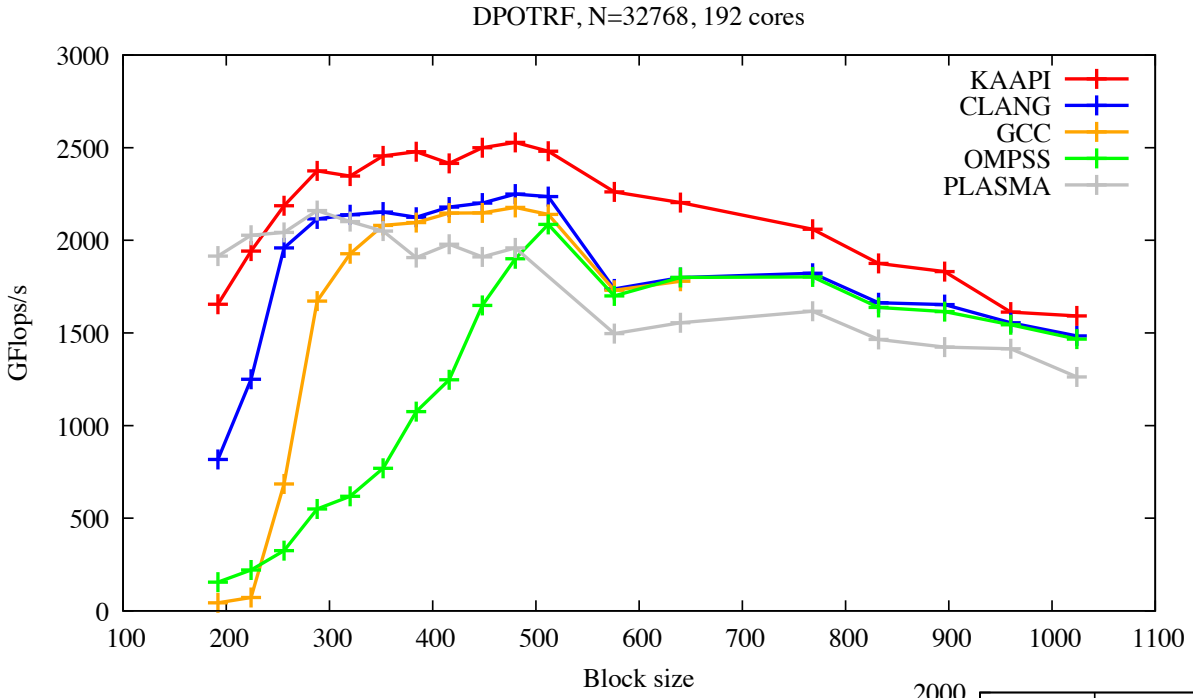
#Cores	Cilk+ (s)	TBB 4.0 (s)	XKaapi (s)	OpenMP (s)
1	1.063	2.356	0.728	*2.43*
8	0.127	0.293	0.094	*51.06*
16	0.065	0.146	0.047	*104.14*
32	0.035	0.072	0.024	**NO TIME**
48	0.028	0.049	0.017	**NO TIME**

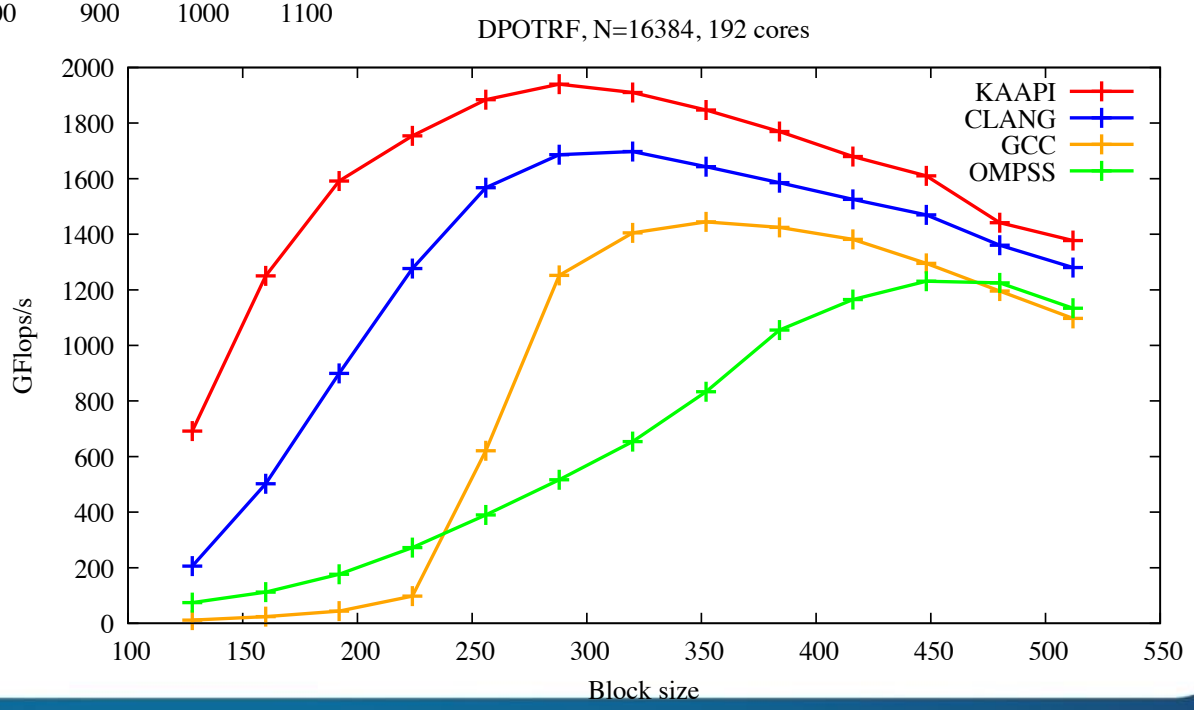
Tasks implementation: e.g. TRSM

Fr

```
template<typename T>
struct TaskBodyGPU<TaskTRSM<T> >
{
  void operator() (
   ka::gpuStream      stream,
   CBLAS_ORDER order, CBLAS_SIDE side, CBLAS_UPLO uplo, CBLAS_TRANSPOSE transA, CBLAS_DIAG diag,
   T alpha, ka::range2d_r <T> A, ka::range2d_rw<T> C
  ) {
    const T* const a = A->ptr();
    const int lda = A->lda();

    T* const c = C->ptr();
    const int ldc = C->lda();

    const int n = C->dim(1);
    const int k = (transA == CblasNoTrans ? A->dim(1) : A->dim(0) );

   const cublasStatus_t status = CUBLAS<T>::trsm(
     kaapi_cuda_cublas_handle(), convertToSideMode(side),
     convertToFillMode(uplo), convertToOp(transA), convertToDiagType(diag),
     n, k,
     &alpha, a, lda,
     c, ldc
   );

   if (status != CUBLAS_STATUS_SUCCESS)
      printf("%s::cublasTrsm() == %d\n", __FUNCTION__, status);
  }
};
```

Impact: Grain size





Running OpenMP BOTS with libKOMP

Barcelona OpenMP Task Suite

- A set of representative benchmarks to evalute OpenMP tasks implementations

kernel	***libGOMP***	***libKOMP***	***Intel***
Alignment	38.8	**40.0**	37.0
FFT	0.5	**12.2**	12.0
Floorplan	27.6	**32.7**	29.2
NQueens	43.7	**47.8**	39.0
MultiSort	0.6	**13.2**	11.3
SparseLU	44.1	**44.4**	35.0
Strassen	20.8	**22.4**	20.5
UTS	0.9	**25.3**	15.0

- **Evaluation platforms**
  - **AMD48: 4x12 AMD Opteron (6174) cores**

- **Softwares**
  - **gcc 4.6.2 + libGOMP**
  - **gcc 4.6.2 + libKOMP**
  - **icc 12.1.2 + Intel OpenMP runtime (KMP)**

Main publications

OpenMP & Kaapi

- Philippe Virouleau, François Broquedis, Thierry Gautier, Fabrice Rastello. Using Data Dependencies to Improve Task-Based Scheduling Strategies on NUMA Architectures. Euro-Par 2016: 531-544
- Philippe Virouleau, Adrien Roussel, François Broquedis, Thierry Gautier, Fabrice Rastello, Jean-Marc Gratien: Description, Implementation and Evaluation of an Affinity Clause for Task Directives. Proceedings of the 12th International Conference on OpenMP in a Heterogeneous World (IWOMP), Osaka, Japan, oct 2016
- Philippe Virouleau , Pierrick Brunet, Francois Broquedis, Nathalie Furmento, Samuel Thibault, Olivier Aumage, Thierry Gautier. Evaluation of the OpenMP Dependent Tasks with the KASTORS Benchmarks Suite. Proceedings of the 10th International Conference on OpenMP in a Heterogeneous World (IWOMP), Bahia, Brazil, sep 2014.
- Marie Durand, Francois Broquedis, Thierry Gautier, Bruno Raffin. An Efficient OpenMP Loop Scheduler for Irregular Applications on Large-Scale NUMA Machines. Proceedings of the 9th International Conference on OpenMP in a Heterogeneous World (IWOMP), 8122:141-155, Lecture Notes in Computer Science, Canberra, Australia, sep 2013.
- Francois Broquedis, Thierry Gautier, Vincent Danjean. libKOMP, an Efficient OpenMP Runtime System for Both Fork-Join and Data Flow Paradigms. IWOMP, :102-115, Rome, Italy, 2012.Applications

Applications

- Jean-Guillaume Dumas, Thierry Gautier, Clément Pernet, Jean-Louis Roch, Ziad Sultan. Recursion based parallelization of exact dense linear algebra routines for Gaussian elimination. Parallel Computing 57: 235-249 (2016)
- Thierry Gautier, Jean-Louis Roch, Ziad Sultan, Bastien Vialla. Par.llel algebraic linear algebra dedicated interface. PASCO 2015: 34-43
- Jean-Guillaume Dumas, Thierry Gautier, Clément Pernet, Ziad Sultan. Parallel Computation of Echelon Forms. Euro-Par 2014: 499-510
- Mathias Ettinger, François Broquedis, Thierry Gautier, Stéphane Ploix, Bruno Raffin. VtkSMP: Task-based Parallel Operators for VTK Filters. EGPGV 2013: 41-48
- Daouda Traore, Jean-Louis Roch, Nicolas Maillard, Thierry Gautier, Julien Bernard. Deque-free work-optimal parallel STL algorithms. EUROPAR 2008, Las Palmas, Spain, aug 2008.
- Jean-Louis Roch, Daouda Traore, Julien Bernard. On-line adaptive parallel prefix computation. EUROPAR 2006, :843–850, Dresden, Germany, aug 2006.
- Thierry Gautier, Fabien Le Mentec, Vincent Faucher, Bruno Raffin. X-kaapi: A Multi Paradigm Runtime for Multicore Architectures. ICPP 2013: 728-735
- Marc Tchiboukdjian, Nicolas Gast, Denis Trystram: Decentralized List Scheduling CoRR abs/1107.3734: (2011). To appear.
- Thierry Gautier, Xavier Besseron, Laurent Pigeon. KAAPI: A Thread Scheduling Runtime System for Data Flow Computations on Cluster of Multi-Processors. Parallel Symbolic Computation'07 (PASCO'07), (15–23), London, Ontario, Canada, 2007.

Multi-CPUs - multi-GPUs

- João V. F. Lima, Thierry Gautier, Vincent Danjean, Bruno Raffin, Nicolas Maillard. Design and analysis of scheduling strategies for multi-CPU and multi-GPU architectures. Parallel Computing 44: 37-52 (2015)
- Raphaël Bleuse, Thierry Gautier, João V. F. Lima, Grégory Mounié, Denis Trystram. Scheduling Data Flow Program in XKaapi: A New Affinity Based Algorithm for Heterogeneous Architectures. Euro-Par 2014: 560-571
- Thierry Gautier, Joao Vicente Ferreira Lima, Nicolas Maillard, Bruno Raffin. XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures. In Proc. of the 27-th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Boston, USA, jun 2013.
- Thierry Gautier, Joao Vicente Ferreira Lima, Nicolas Maillard, Bruno Raffin. Locality-Aware Work Stealing on Multi-CPU and Multi-GPU Architectures. 6th Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG), Berlin, Allemagne, jan 2013.
- Julio Toss, Thierry Gautier. A New Programming Paradigm for GPGPU. EUROPAR 2012, Rhodes Island, Greece, aug 2012.
- J.V.F. Lima, Thierry Gautier, Nicolas Maillard, Vincent Danjean. Exploiting Concurrent GPU Operations for Efficient Work Stealing on Multi-GPUs. 24rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Columbia University, New York, USA, oct 2012.
- Everton Hermann, Bruno Raffin, François Faure, Thierry Gautier, Jérémie Allard. Multi-GPU and Multi-CPU Parallelization for Interactive Physics Simulations. EUROPAR 2010, Ischia Naples, Italy, aug 2010.

Fault Tolerence

- Xavier Besseron, Thierry Gautier. Impact of over-decomposition on coordinated checkpoint/rollback protocol. Workshop on Resiliency in High-Performance Computing, 17-th International European Conference On Parallel and Distributed Computing, Bordeaux, France, aug 2011.
- Xavier Besseron, Thierry Gautier. Optimised recovery with a coordinated checkpoint/rollback protocol for domain decomposition applications. Modelling, Computation and Optimization in Information Systems and Management Sciences (MCO'08), :497–506, Metz, France, sep 2008.
- Samir Jafar, Axel W. Krings, Thierry Gautier. Flexible Rollback Recovery in Dynamic Heterogeneous Grid Computing. IEEE Transactions on Dependable and Secure Computing, 6(1), 2009.

Main applications using XKaapi

Dense linear algebra

- numerical -academic benchmarks-
- non-numerical (Finite Field)
  - J.-G. Dumas, C. Pernet, Z. Sultan

Parallel support VTK algorithms http://www.vtk.org

- Kaapi is one of the supported runtime in the VTK distribution

XKaapi is the runtime for multicore EPX http://www-epx.cea.fr

- V. Faucher [CEA]

All of them are using OpenMP-4.0 or are in the process of using it

Code size (http://kaapi.gforge.inria.fr), v3.1

- runtime 30k C lines, OpenMP support 8k C lines (wc -l)
- 180kB (libkaapi.so), 64kB (libkomp.so) = 244kB
- [libGOMP: 670kB; libOMPSS: 664kB; libStarPU -K'STAR dist- 5MB]

EPX

EPX (EUROPLEXUS) code [CEA - IRC - EDF - ONERA], V. Faucher

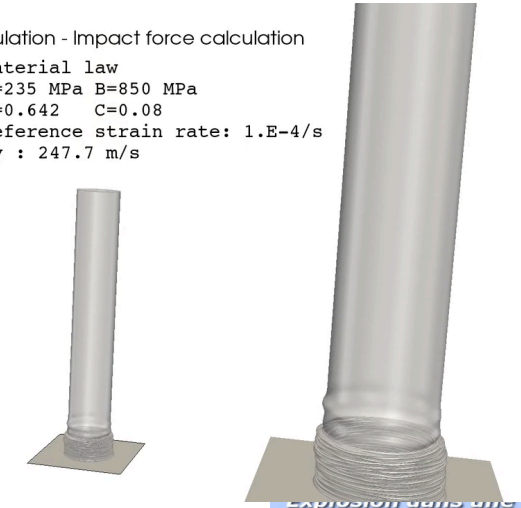
- Fluid-Structure systems subjected to fast transient dynamic loading

« Grand prix SFEN 2013 »

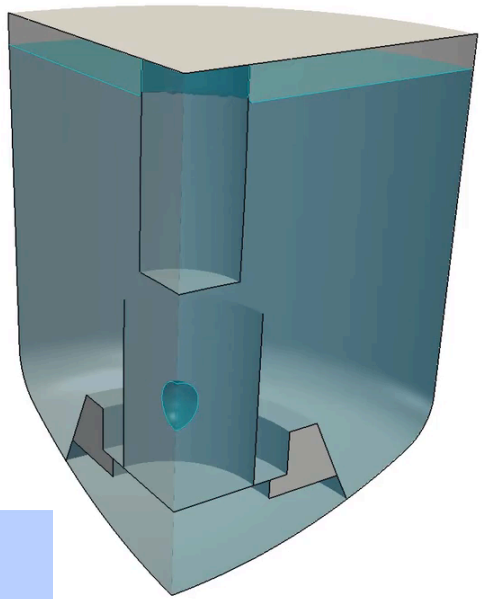
- http://energies.sfen.org/emploi/le-grand-prix-sfen

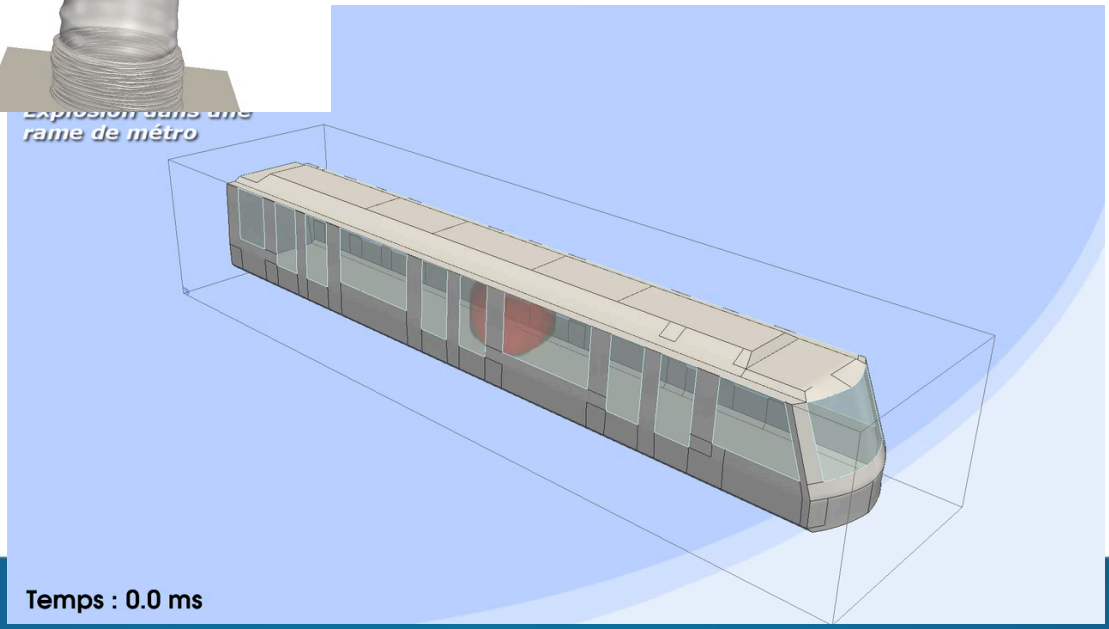
EUROPLEXUS
Meppen tests simulation - Impact force calculation

Johnson-Cook material law
Parameters : A=235 MPa B=850 MPa
n=0.642 C=0.08
Reference strain rate: 1.E-4/s
Impact velocity : 247.7 m/s

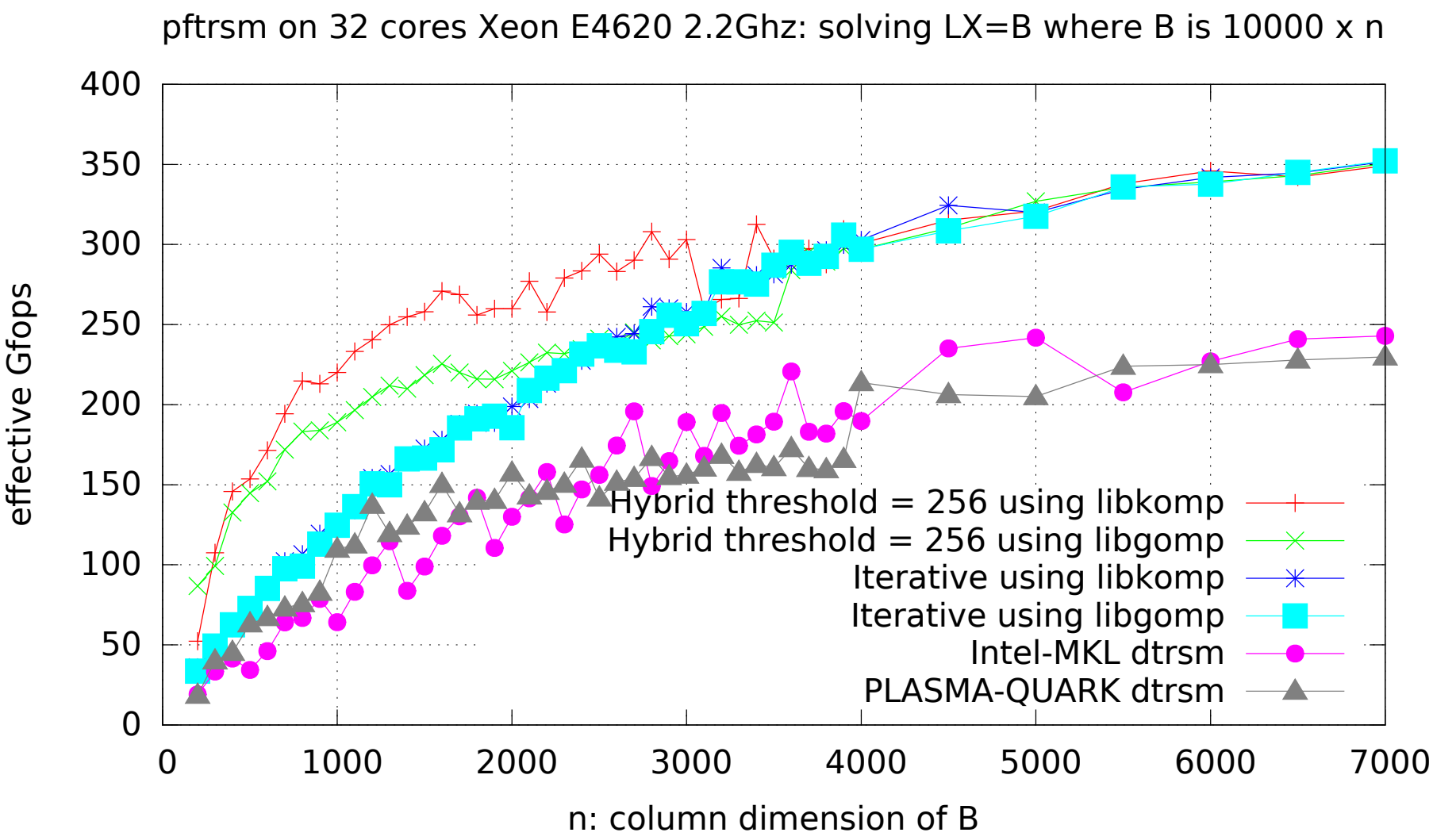


EUROPLEXUS
Simulation of MARA10 experiment
ADCR material - VOFIRE algorithm





Other comparizon libGOMP / Kaapi - libKOMP



Jean-Guillaume Dumas, Thierry Gautier, Clément Pernet, Jean-Louis Roch, and Ziad Sultan. Recursion based parallelizatior of exact dense linear algebra routines for Gaussian elimination. *Parallel Comput.* 57, C (September 2016), 235-249.

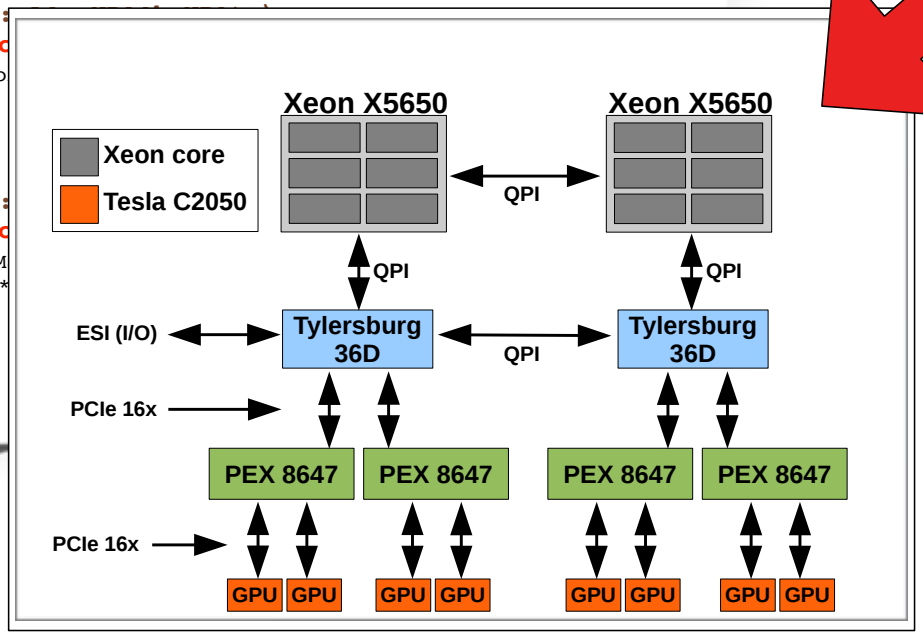
Multi-CPUs - Multi-GPUs

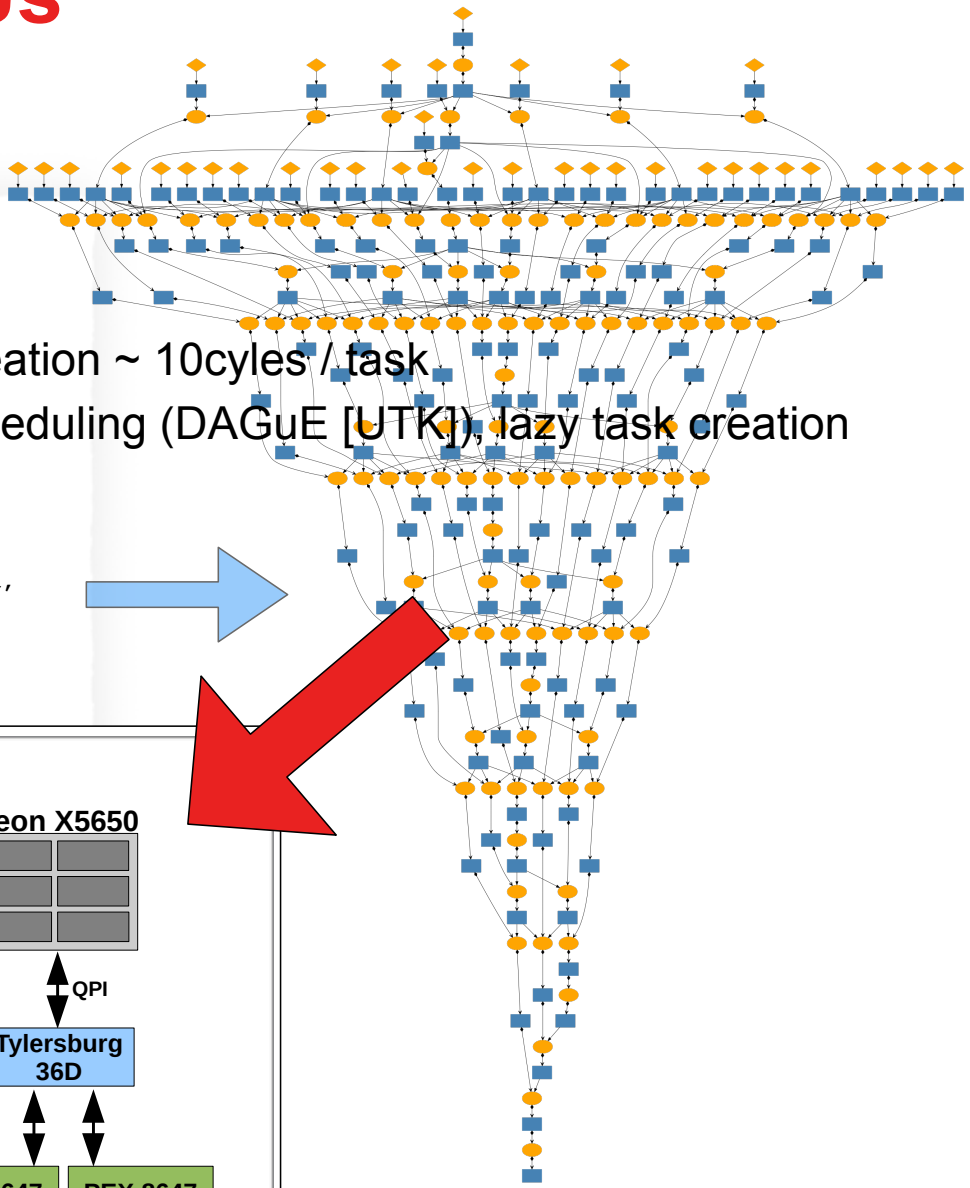
Size of the graph !

Two complementary approaches

- fine grain task management: XKaapi task creation ~ 10cyles / task
- compact dag representation for symbolic scheduling (DAGuE [UTK]), lazy task creation

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

void Cholesky( int N, double A[N][N], size_t BS )
{
  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:
                 depend(inc
      cblas_dsyrk ( CblasRowMajo
        NB, NB, -1.0, &A[m*N+k],

      for (size_t n=k+NB; n < m;
      {
#pragma omp task depend(in:
                 depend(inc
        cblas_dgemm ( CblasRowM
          NB, NB, NB, -1.0, &A[m*
      }
    }
  }
#pragma omp taskwait
}
```





Work stealing with heuristic

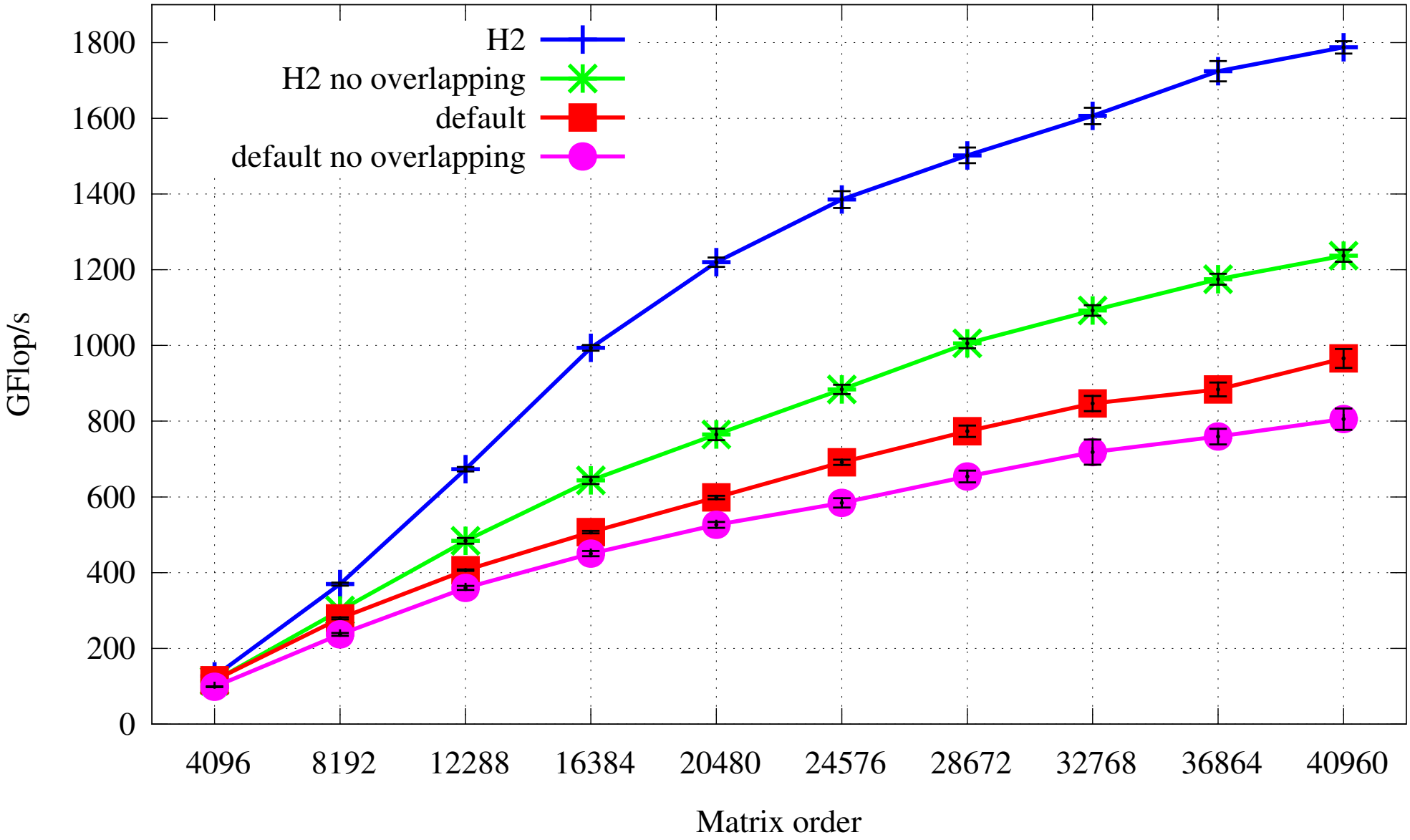
[Lima et al. IPDPS 13, PARCO 15]

- Cholesky factorization on hybrid architecture
- Up to 8 GPUs and 12 CPUs
  - XKaapi 2.2

Key points

- Reduction of the critical path
  - multi-CPUs parallelization of the panel factorization
- Overlapping communication by computation
  - task execution splits in three steps:
    data access from ressource, data copy to target, kernel launch
  - 3 CUDA streams (inputs/outputs/kernels)
- DSM management
  - cache strategy: 2 sets maintained (data Read; and data RW or Write)
  - evict first data in R set using LRU
- Work Stealing heuristic
  - GPU : local steal first in its queue, then remote
  - H1: task activation : enqueue task on the resource that holds most of its data
  - H2: task activation : enqueue task on the resource that holds its data write (OCR strategy)

Strategies / Overlapping

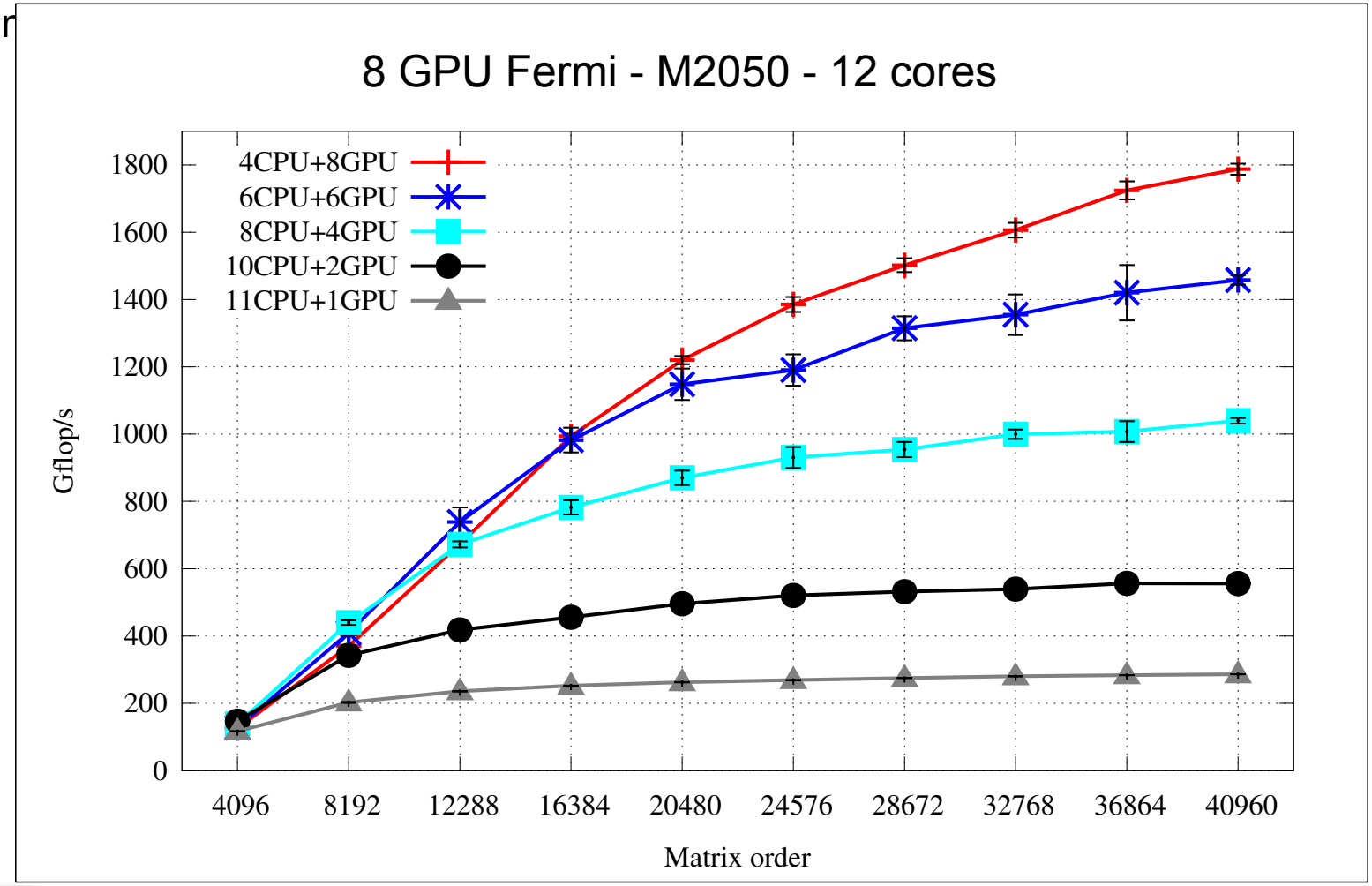


Multi-GPUs Support

[IPDPS 2013] work with J. Lima, N. Maillard, B. Raffin

DPOTRF

- usir



Cholesky factorization on Xeon Phi

[SBAC-PAD 2013]

N=8192

Ongoing work to reduce cache usage