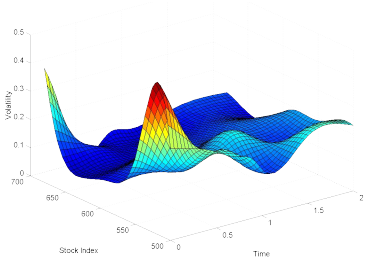


Online Scheduling of Parallel Programs on Hybrid Architectures

Nicolas Maillard, João Lima, Júlio Toss,
Thierry Gautier, Vincent Danjean

nicolas@inf.ufrgs.br

7th ***Scheduling for Large Scale Systems Workshop***

GPPD

INSTITUTO DE INFORMÁTICA UFRGS

Joint works INF/UFRGS + LIG/ INRIA/CNRS MOAIS
Jean-Louis Roch, Thierry Gautier, Vincent Danjean, Nicolas Maillard

João Lima
Júlio Toss
Vinícius Garcia

nicolas@inf.ufrgs.br - GPPD

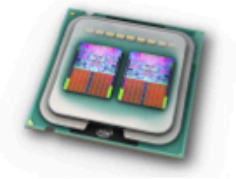
INSTITUTO DE INFORMÁTICA UFRGS

CHALLENGE:
SCHEDULING
FINE GRAINED AND IRREGULAR PARALLELISM ON HYBRID ARCHITECTURES

- Would you tell me, please, which way I ought to go from here?'
- 'That depends a good deal on where you want to get to,'
said the Cat.

2 PARTS:
1. ONLINE SCHEDULING OF LINEAR ALGEBRA PROGRAMS WITH KAAPI [JOÃO LIMA]
2. GPU WORK STEALING FOR IRREGULAR PROBLEMS [JÚLIO TOSS]

Hardware X Programming API

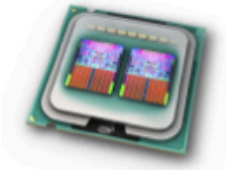


multicore(s)

OpenMP, Cilk, TBB (Intel), Kaapi,…
Atlas, PLASMA, …

GPU(s)

CUDA, Thrust, …
CUBLAS, MAGMA, …



multicore(s)

+

GPU(s)

StarPU, XKaapi

Cholesky Factorization

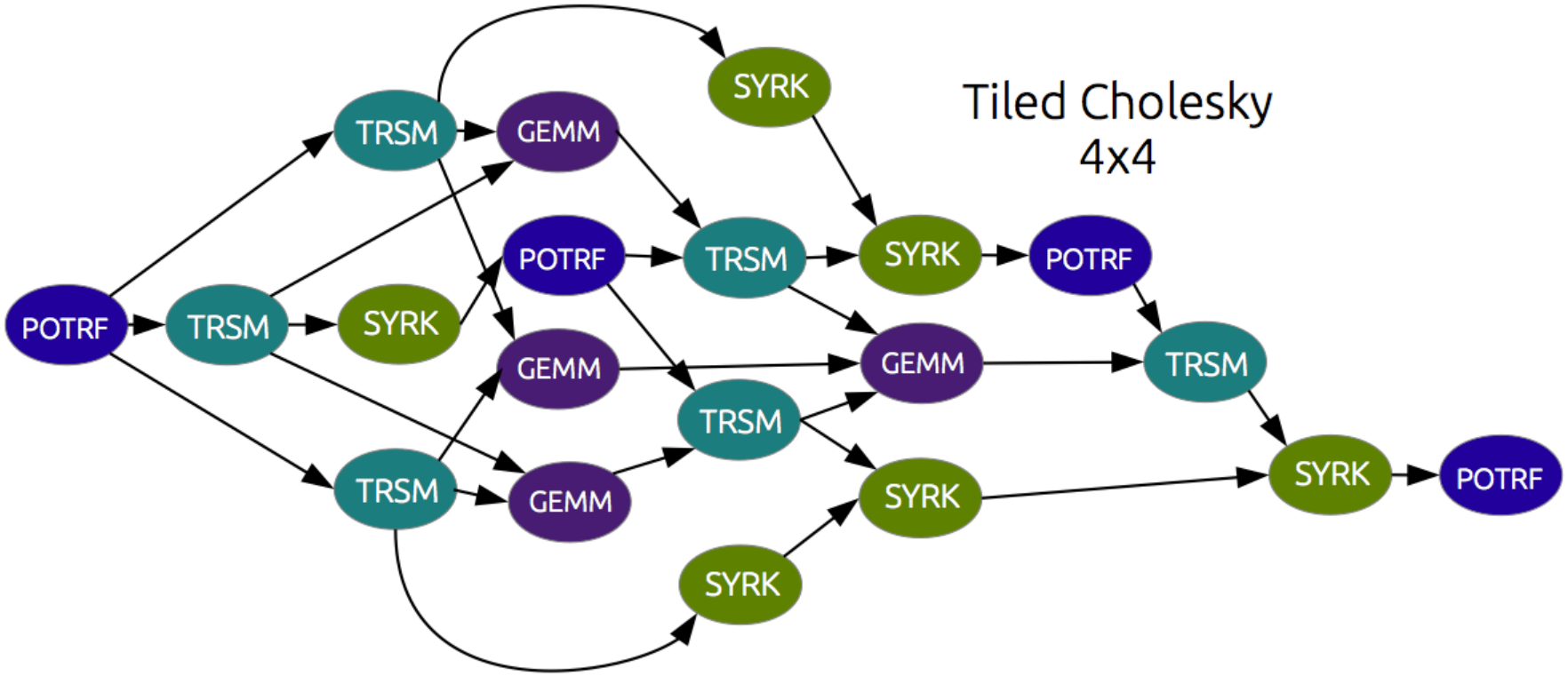
Tiled version, PLASMA

```
 1 for k = 1,...,p do
 2   Akk= POTRF(Akk)
 3   for i = k+1,...,p do
 4     Aik= TRSM(Akk,Aik)
 5   endfor
 6   for i= k+1,...,p do
 7     for j= k+1,...,i do
 8       Aij= GEMM(Aik, Ajk, Aij)
 9     endfor
10     Aii= SYRK(Aik, Aii)
11   endfor
12 endfor
```


nicolas@inf.ufrgs.br - GPPD

Graph Dependencies

[Buttari et al. 2009]



Magma pre-computes the graph and uses it
to (statically) schedule the tasks on a CPU+GPU environment.

Scheduling the Tasks on-line with Kaapi

Kaapi

- A French project - http://kaapi.gforge.inria.fr/
- C++ library that provides an API for parallel programming based on tasks.
- A shared global address space
  - You create objects inside it with the keyword “shared”
- A task is a call to a function, prefixed by “fork”
  - Just like Unix / Cilk
  - Tasks communicate through objects *shared*.
  - Tasks specify the access mode to shared objects (read/write)
- The Kaapi runtime builds the data-flow graph and uses it to schedule the tasks.

.inf INSTITUTO DE INFORMÁTICA UFRGS

A simple example

Fibonacci

```
struct Fibonacci {
  void operator()( int n, a1::Shared_w<int> result ) {
    if (n < 2) result.write( n );
    else {
      a1::Shared<int> subresult1;
      a1::Shared<int> subresult2;
      a1::Fork<Fibonacci>()(n-1, subresult1);
      a1::Fork<Fibonacci>()(n-2, subresult2);
      a1::Fork<Sum>()(result, subresult1, subresult2);
    }
  }
};
struct Sum { void operator()(a1::Shared_w<int> result,
                             a1::Shared_r<int> sr1,
                             a1::Shared_r<int> sr2 )
{ result.write( sr1.read() + sr2.read() ); }
}
```


nicolas@inf.ufrgs.br - GPPD

A simple example

Fibonacci

```
struct Fibonacci {
  void operator()( int n, a1::Shared_w<int> result ) {
    if (n < 2) result.write( n );
    else {
      a1::Shared<int> subresult1;
      a1::Shared<int> subresult2;
      a1::Fork<Fibonacci>()(n-1, subresult1);
      a1::Fork<Fibonacci>()(n-2, subresult2);
      a1::Fork<Sum>()(result, subresult1, subresult2);
    }
  }
};
struct Sum { void operator()(a1::Shared_w<int> result,
                             a1::Shared_r<int> sr1,
                             a1::Shared_r<int> sr2 )
{ result.write( sr1.read() + sr2.read() ); }
}
```


nicolas@inf.ufrgs.br - GPPD

Cholesky Factorization

Xkaapi [J. Lima, 2012]

```
for(k=0;k < N;k += blocsize){
  ka::Spawn<TaskDPOTRF>()( A(rk,rk) );
  for(m=k+blocsize;m < N;m += blocsize
    ka::Spawn<TaskDTRSM>()(A(rk,rk),
      A(rm,rk) /* B */ );

  for(m=k+blocsize;m < N;m += blocsize){
    ka::Spawn<TaskDSYRK>()( A(rm,rk),
       A(rm,rm) /* C */);

    for(n=k+blocsize;n < m;n += blocsize)
      ka::Spawn<TaskDGEMM>()( A(rm,rk), /* A */
        A(rn,rk), /* B */
        A(rm,rn)  /* C */ );
  }
}
```


nicolas@inf.ufrgs.br - GPPD

Task Body (CPU)

XKaapi [J. Lima, 2012]

```
template<>
struct TaskBodyCPU<TaskDPOTRF> {
  void operator()(CBLAS_ORDER order,
                  CBLAS_UPLO uplo,
                  ka::range2d_rw<double_type> A
  )
  {
    const int n      = A.dim(0);
    const int lda    = A.lda();
    double_type* const a = A.ptr();
    clapack_potrf( order, uplo, n, a, lda );
  }
};
```


nicolas@inf.ufrgs.br - GPPD

Task Body (GPU)

XKaapi [J. Lima, 2012]

```
template<>
struct TaskBodyGPU<TaskDPOTRF>
{
  void operator()(
      ka::gpuStream stream,
      CBLAS_ORDER order, CBLAS_UPLO uplo,
      ka::range2d_rw<double_type> A )
  {
    const int n      = A.dim(0);
    const int lda    = A.lda();
    double_type* const a = A.ptr();
    magma_int_t info = 0;
    char uplo_=(uplo == CblasUpper) ? 'U' : 'L';

    magma_potrf_gpu( uplo_, n, a, lda, &info );
    if (info){ /* ERROR */ }
  }
};
```


nicolas@inf.ufrgs.br - GPPD
.inf INSTITUTO DE INFORMÁTICA UFRGS

Task Bodies (Others)

Xkaapi [J. Lima, 2012]

TaskDPOTRF

- GPU (*magma_dpotrf_gpu*) / CPU (*clapack_dpotrf*

TaskDTRSM

- GPU (*cublasDtrsm*) / CPU (*cblas_dtrsm*)

TaskDSYRK

- GPU (*cublasDsyrk*) / CPU (*cblas_dsyrk*)

TaskDGEMM

- GPU (*cublasDgemm*) / CPU (*cblas_dgemm*)

.inf INSTITUTO DE INFORMÁTICA UFRGS

Managing the tasks

Internals [J. Lima, 2012]

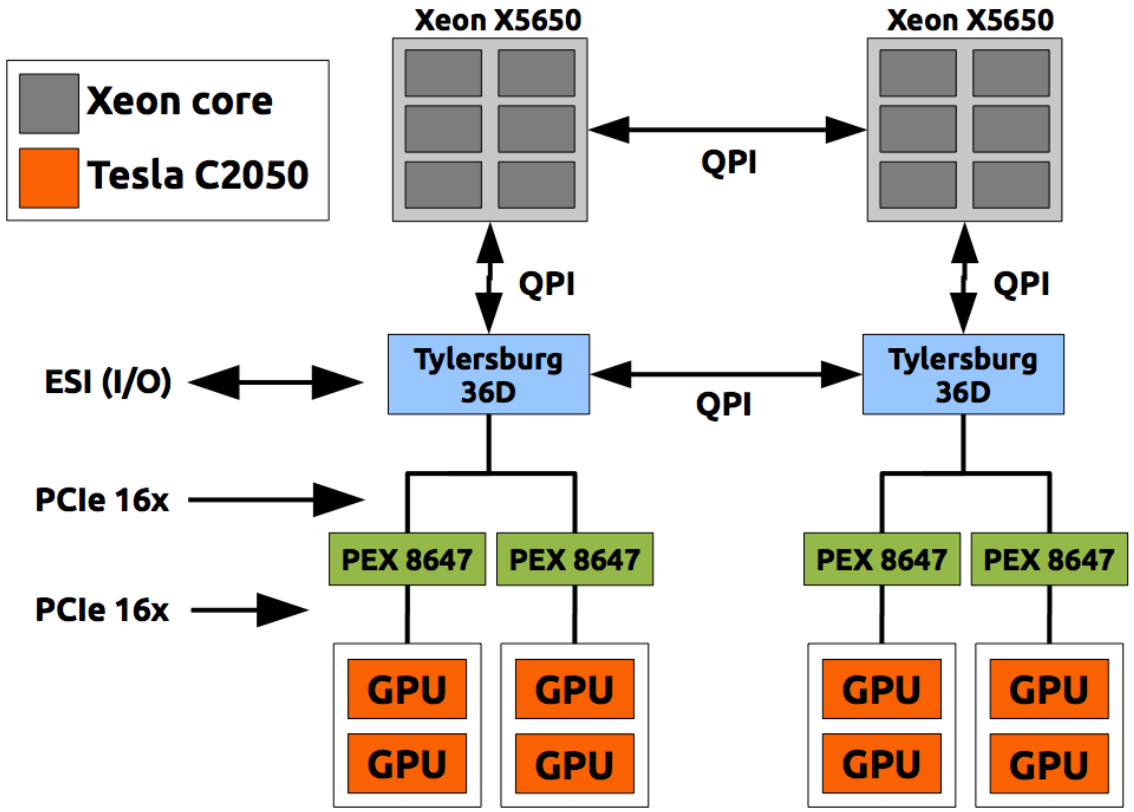
- GPU tasks receive GPU memory pointers
  - Alloc/transfer transparent to the programmer
  - Allocator – LRU blocks by lists
    Transfers to GPU by demand
    Transfers to CPU are queued in advance
- One CPU thread is responsible for one GPU
  - Work Stealing among CPU threads only.
- GPUs: Work Stealing unused (yet)
    They only execute asynchronously all ready tasks
- Overlap execution and transfers (full-duplex)
  - Three independent CUDA streams (HostToDevice H2D, DeviceToHost D2H, Kernel)

.inf INSTITUTO DE INFORMÁTICA UFRGS

Experimental Platform

IdGraf [J. Lima, 2012]

- 2 six-core Intel Xeon X5650
  - at 2.66 GHz,72 GB of main memory
- 8 NVIDIA Tesla C2050 GPUs
  - 448 cores, 3 GB GDD
- GCC 4.4 + CUDA 4.1
- MAGMA 1.1.0
- ATLAS 3.9.69 + LAPACK 3.4.0



nicolas@inf.ufr

Results

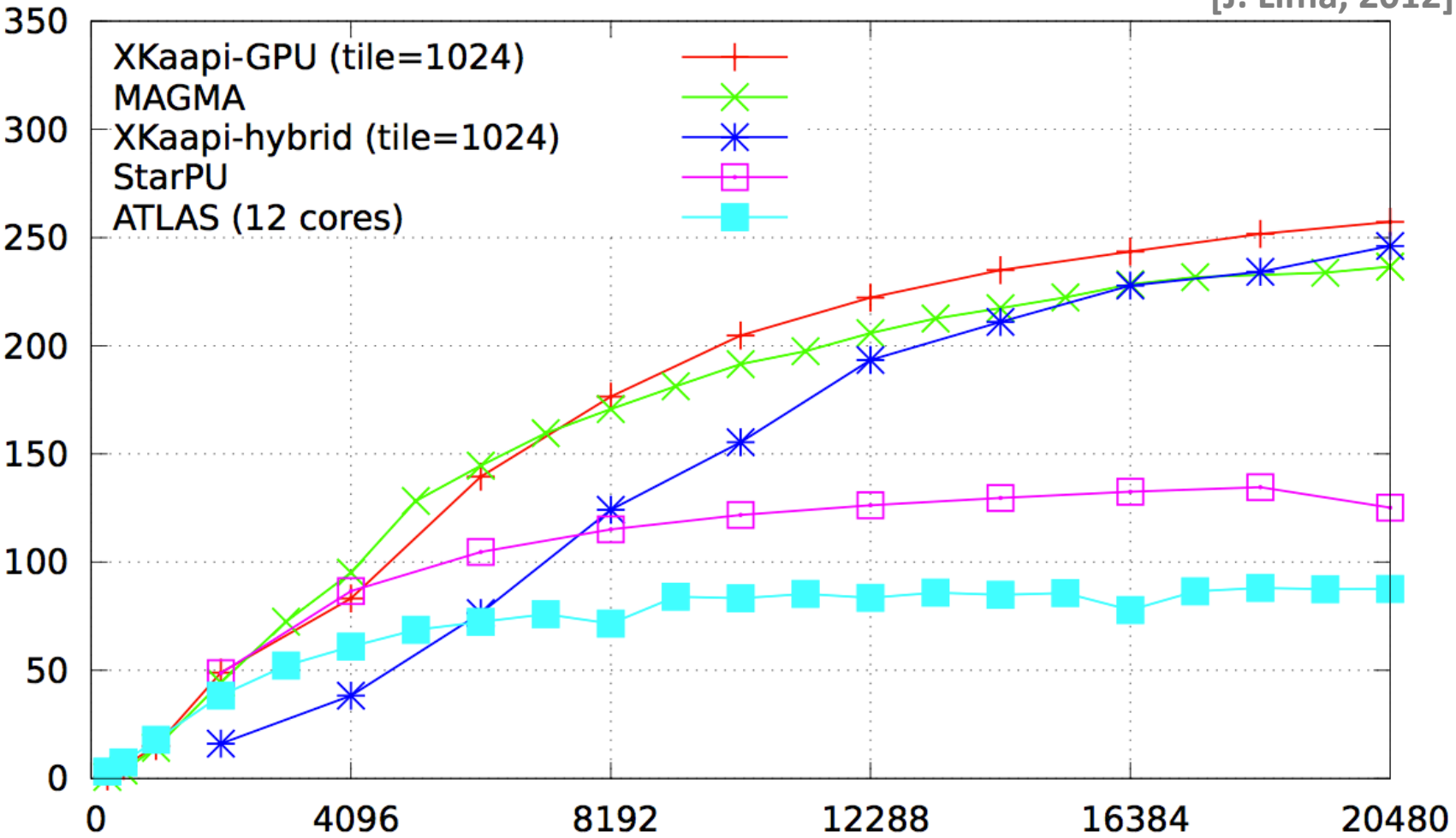
[J. Lima, 2012]

- Cholesky factorization and DGEMM (matrix product)
  - Cholesky uses MAGMA 1.1.0 (1 GPU)
  - DGEMM uses CUBLAS 4.1
- XKaapi (1 CPU + 1 GPU)
  - Multi-CPU is disabled
- Performance measured in Gflops/s
- Each results is the mean of 30 executions
- Execution time includes
  - XKaapi – allocation/transfer times (GPU)
  - CUBLAS – transfer times
  - MAGMA – allocation/transfer times (GPU)

.inf INSTITUTO DE INFORMÁTICA UFRGS

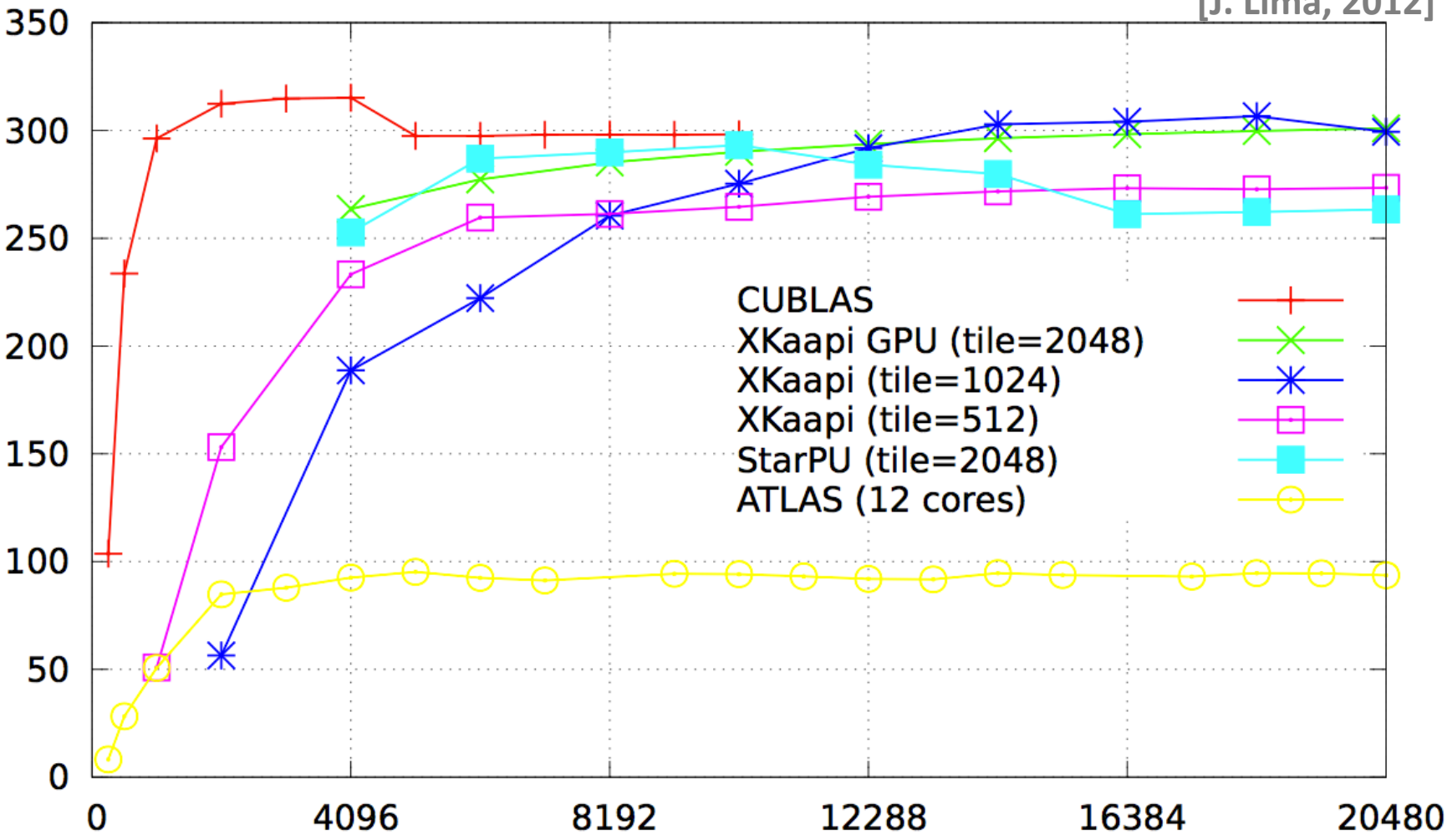
Gflops/s (Cholesky)

[J. Lima, 2012]



Gflops/s (dgemm)

[J. Lima, 2012]



Work Stealing on the GPU?

[J. Toss 2012]

- One GPUs == more than one multi-processor (MP) (e.g. 30).
  - You can use Work Stealing inside a GPU.
  - Each MP owns its own deque of tasks instead of:
    - Running a fixed amount of blocks of kernels (default);
    - or accessing a centralized list of tasks.

- This WS for GPU can be implemented with CUDA.

.inf INSTITUTO DE INFORMÁTICA UFRGS

Cuda Work Stealing

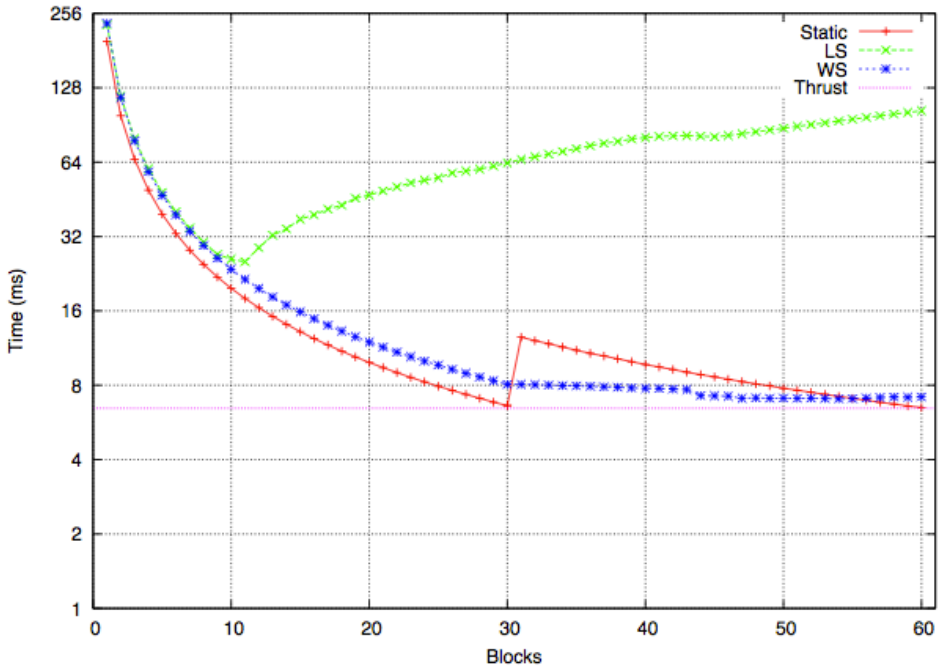
[J. Toss, Europar 2012]

```
1  __global__ void
2  arrayinc(float* g_idata, float* g_odata, int popSize) {
3   int *my_wq = kwq[blockIdx.x];
4   while (1) {
5     if (threadIdx.x == 0) {
6       if( ! pop(my_wq, locbeg, locend, popSize) ) {
7         int *victim_wq = kwq[rand()];
8         int stealSize = lenght(victim_wq)/2;
9         if ( stealSize >= popSize ) {
10          mutex_lock(victime_wq.mutex);
11          steal(victim_wq, my_wq.beg, my_wq.end, stealSize )
12          mutex_unlock(victim_wq.mutex);
13        }
14      }
15    }
16    __syncthreads();
17    transform(g_idata,g_odata, locbeg, locend);
18    __syncthreads();
19    if ( terminate ) break;
20    }
21 }
```

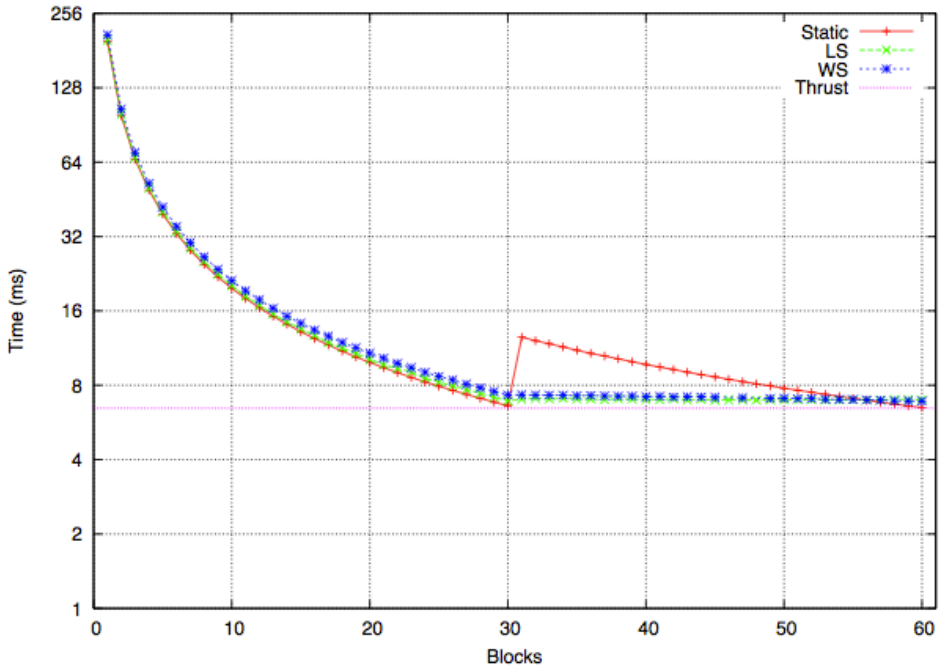

nicolas@inf.ufrgs.br - GPPD

Scheduling impact on the runtime

[J. Toss, Europar 2012]



(a) Pop size of a single task



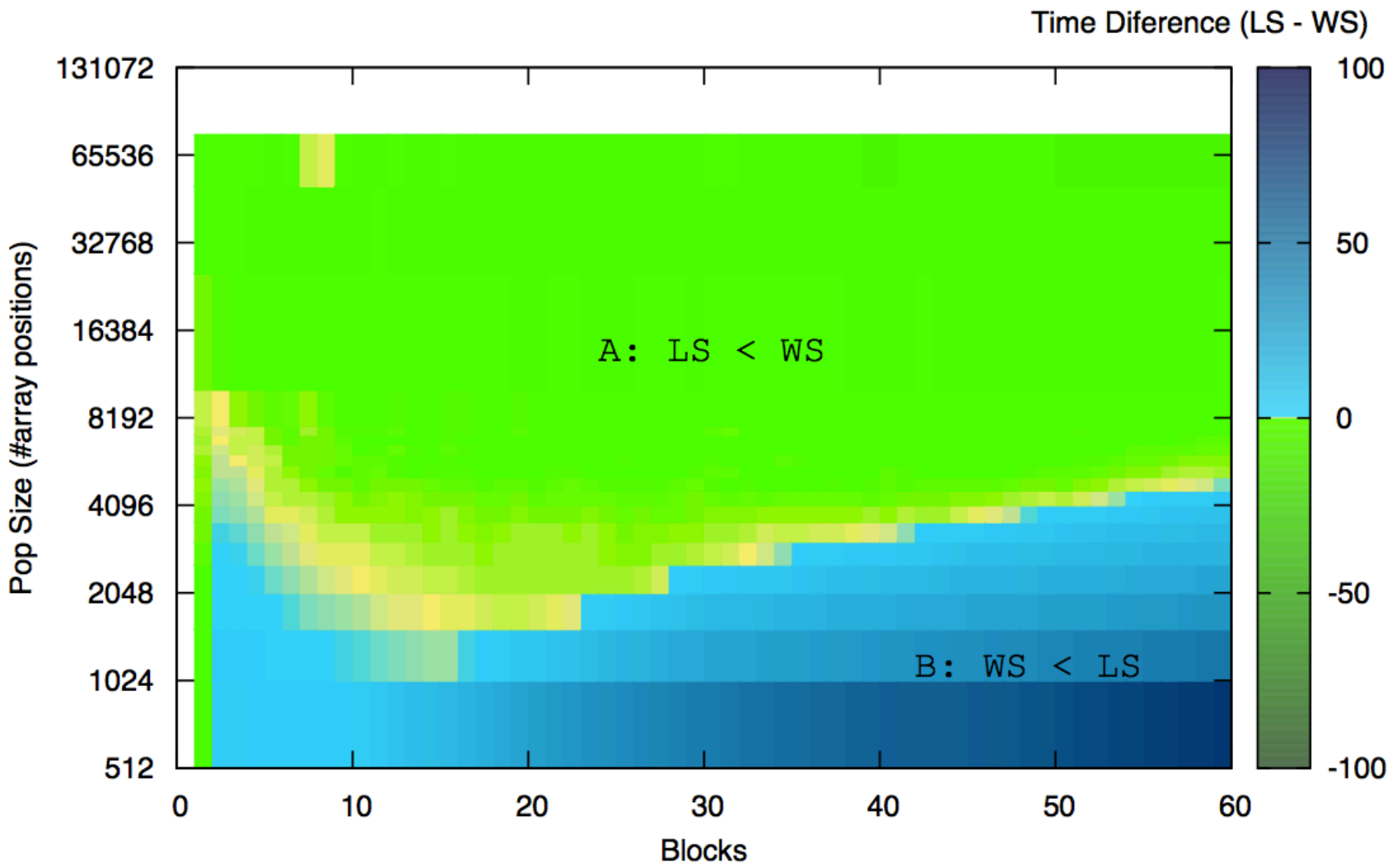
(b) Best pop size for each method

Comparing Static (block) scheduling, List scheduling, Work Stealing (e thrust).
The program is Transform()

.inf INSTITUTO DE INFORMÁTICA UFRGS

Tuning the parameters…

[J. Toss, Europar 2012]



For irregular workloads…

[J. Toss, Europar 2012]

- Still using Transform(), but on two different artificial patterns:
  - Workload 1: one task out of two is zeroed;
  - Workload 2: one task out of four is zeroed.

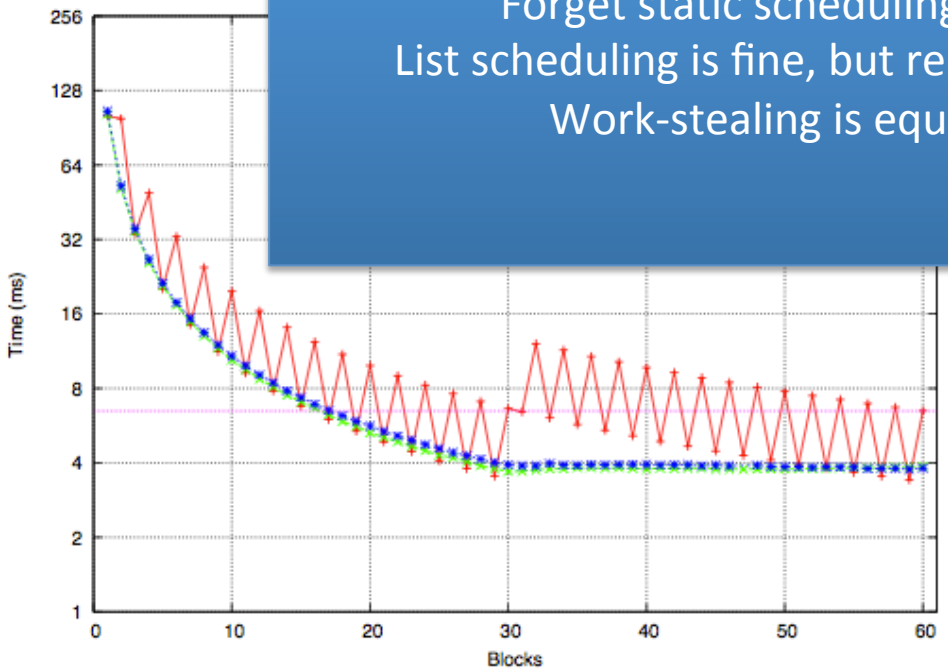
- best configuration of pop size found for each method:
  - LS uses a pop size of 10240 (20 tasks) for workload 1 and 25600 (50 tasks) for workload 2;
  - WS uses pop sizes of 4096 (8 tasks) for both workloads.

.inf INSTITUTO DE INFORMÁTICA UFRGS

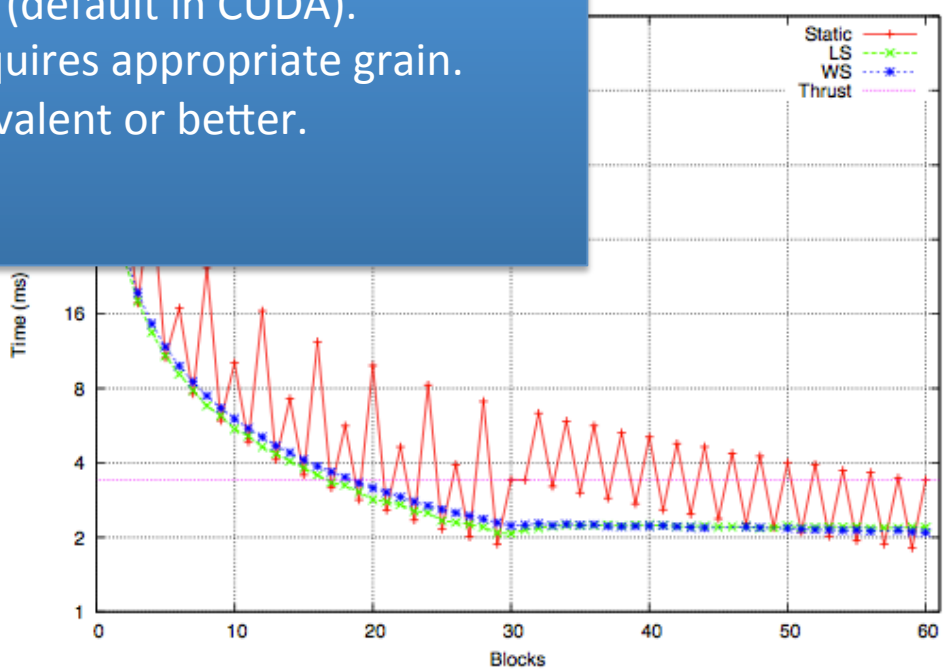
For irregular workloads...

...uropar 2012]

Forget static scheduling (default in CUDA).
List scheduling is fine, but requires appropriate grain.
Work-stealing is equivalent or better.



Workload 1

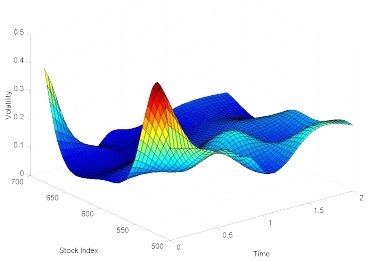


Workload 2

Conclusions

- Using asynchronous memory transfers, one can overlap communication and computation
  - enable efficient online scheduling of the tasks.
- We have prototypes with Kaapi:
  - For Multi-CPU (WS), Mono-GPU (work Pushing) systems and asynchronous transfers.
  - For Mono-CPU (WS), multi-GPUs (WS) systems.
- Next stage: how do you refine the basic Work Stealing in hybrid environments:
  - Use (static) priorities?
  - Data locality, to re-use data that has been pre-fetched into a GPU memory?
  - Steal more than one task?
  - Improve the GPU to GPU communication?

.inf INSTITUTO DE INFORMÁTICA UFRGS



Online Scheduling of Parallel Programs on Hybrid Architectures

Nicolas Maillard, João Lima, Júlio Toss,
Thierry Gautier, Vincent Danjean

nicolas@inf.ufrgs.br

7th ***Scheduling for Large Scale Systems Workshop***

GPPD

INSTITUTO DE INFORMÁTICA UFRGS