StarPU: task-based scalable Runtime system for heterogeneous multicore architectures

Olivier Aumage, Nathalie Furmento, Samuel Thibault
INRIA Storm Team-Project

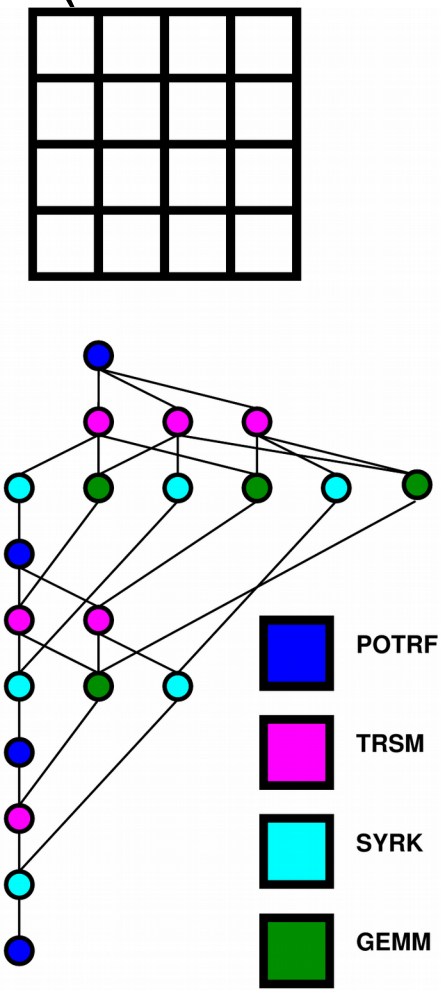
INRIA Bordeaux, LaBRI, University of Bordeaux

Task management

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Write your application as a task graph

Even if using a sequential-looking source code

➔ Portable performance

Sequential Task Flow (STF)

- Algorithm remains the same on the long term
- Can debug the sequential version.
- Only kernels need to be rewritten
  - BLAS libraries, multi-target compilers
- Runtime will handle parallel execution

Overview of StarPU

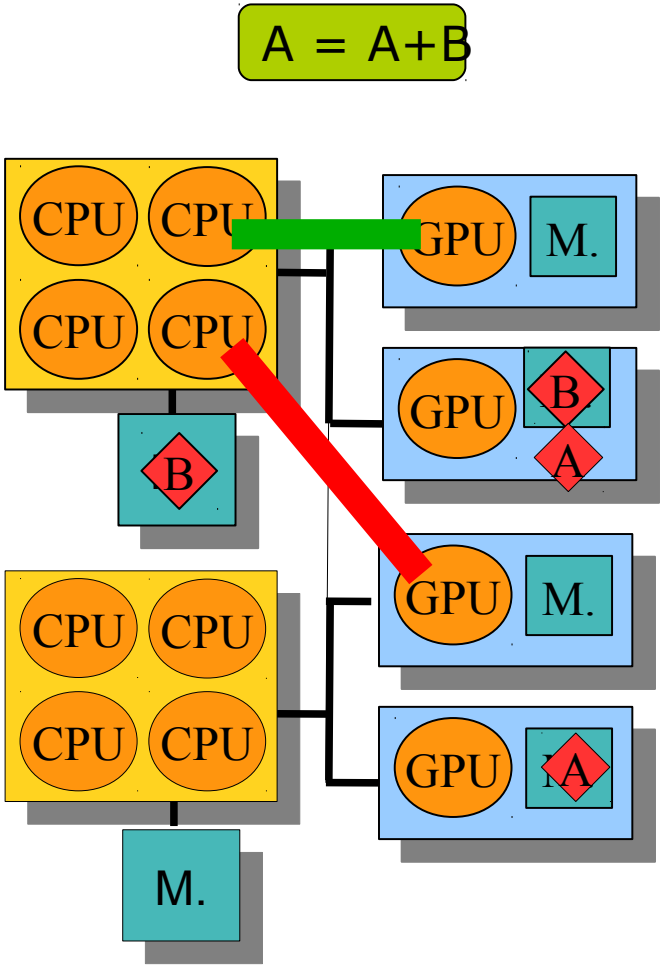
Rationale

Task scheduling

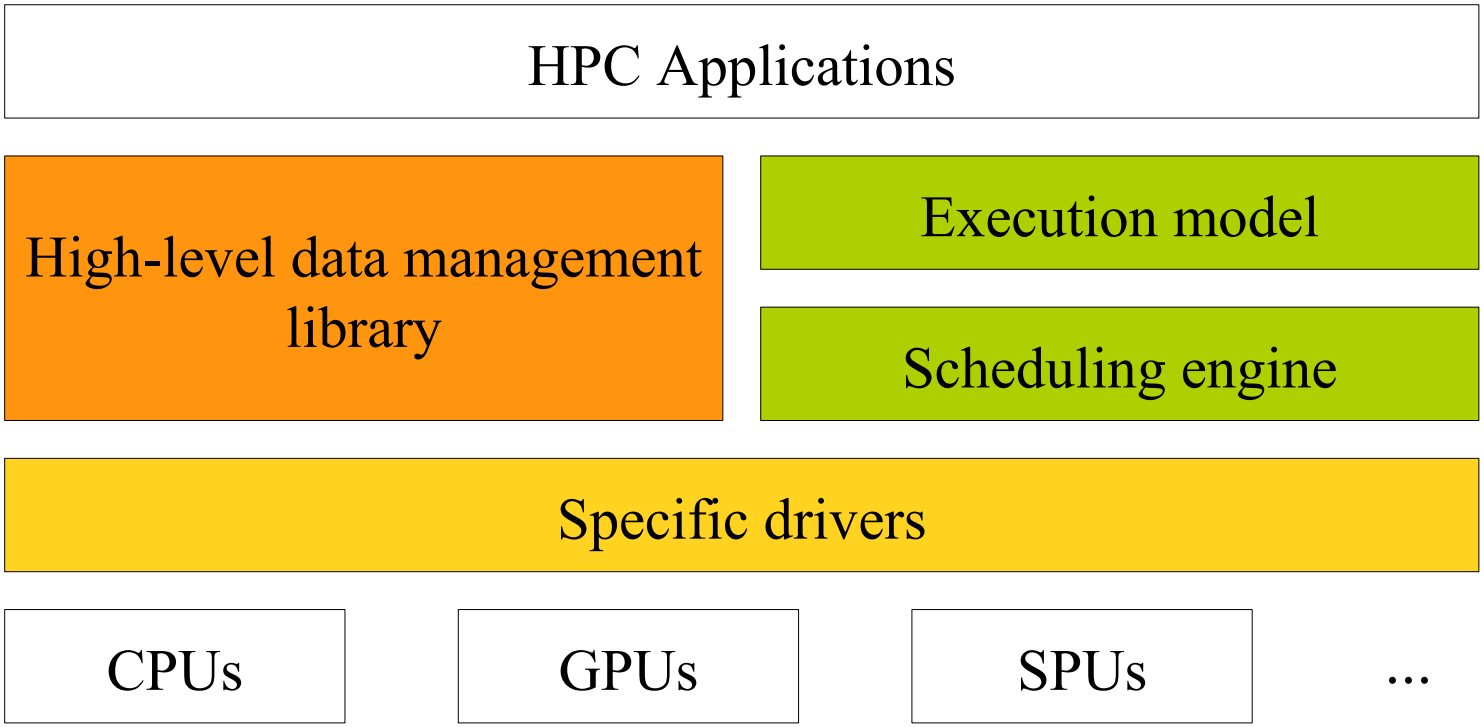
- Dynamic
- On all kinds of PU
  - General purpose
  - Accelerators/specialized

Memory transfer

- Eliminate redundant transfers
- Software VSM (Virtual Shared Memory)



The StarPU runtime system



Mastering CPUs, GPUs, SPUs … ***PUs → StarPU**

The StarPU runtime system

Development context

- History
  - Started about 9 years ago
    - PhD Thesis of Cédric Augonnet
  - StarPU main core ~ 70k lines of code
  - Written in C
- Open Source
  - Released under LGPL
  - Sources freely available
    - svn repository and nightly tarballs
    - See https://starpu.gforge.inria.fr/
  - Open to external contributors
- [HPPC'08]
- [Europar'09] – [CCPE'11],... >1000 citations

The StarPU runtime system

Supported platforms

- Supported architectures
  - Multicore CPUs (x86, PPC, ...)
  - NVIDIA GPUs
  - OpenCL devices (eg. AMD cards)
  - Intel Xeon Phi (MIC), Intel SCC
  - Kalray MPPA (experimental)
  - Cell processors (experimental) [SAMOS'09]
- Supported Operating Systems
  - Linux
  - Mac OS
  - Windows

Summary

```
starpu_codelet_t cl = { .cpu_func = my_f, ... };
float array[NX];
        ...
starpu_data_handle vector_handle;
starpu_vector_data_register(&vector_handle, 0,
            array, NX, sizeof(vector[0]));
...
starpu_task_insert(&cl, vector_handle, 0);
...
starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);
```

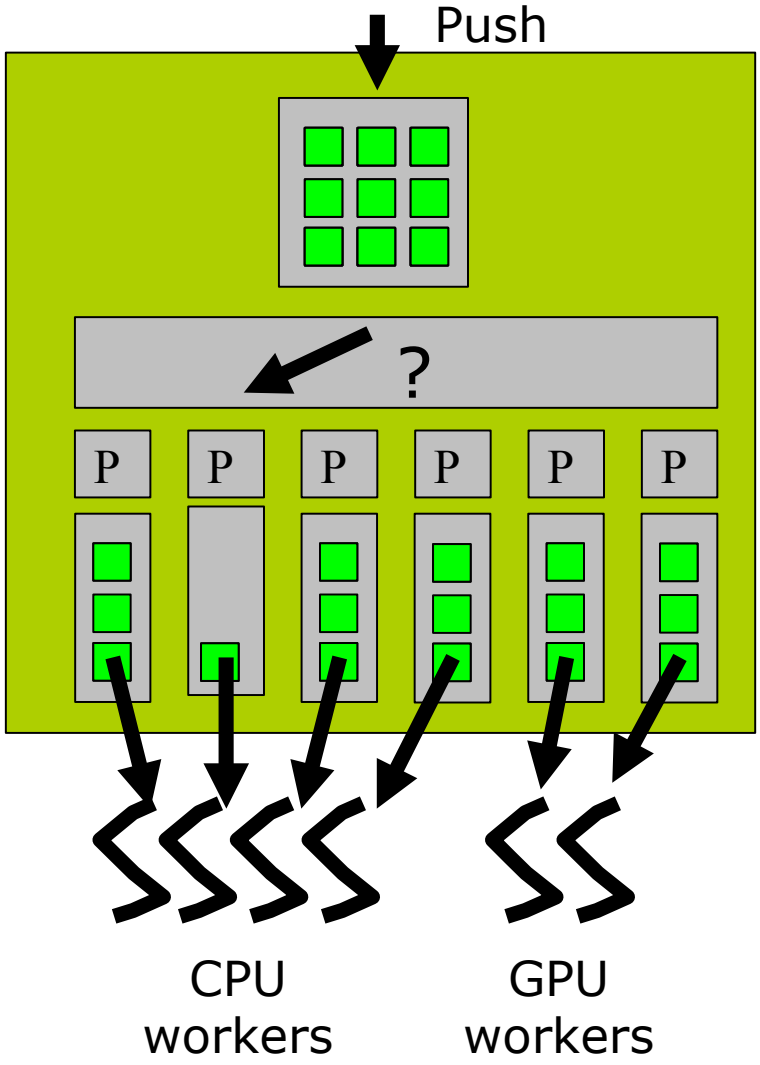
Task scheduling

Component-based schedulers

- Containers
  - Priorities
- Switches
- Side-effects (prefetch, …)

Push/Pull mechanism

S. Archipoff, M. Sergent



More features

- Cluster support
  - MPI communication
  - Decentralized model
    - Application-provided data mapping
    - Automatic optimized transfers
- Memory consumption control
- Out of core support
  - Disk as optimized « swap »
  - or as backstore for matrix tiles
- Execution simulation support
- OpenMP and OpenCL interfaces
- ...

Applications on top of StarPU

Using CPUs, GPUs, distributed, out of core, ...

- Dense linear algebra
  - Cholesky, QR, LU, ... : Chameleon (based on Plasma/Magma)
- Sparse linear algebra
  - QR_MUMPS
  - PaStiX
- Compressed linear algebra
  - BLR, h-matrices
- Fast Multipole Method
  - ScalFMM
- Conjugate Gradient
- Other programming models : Data flow, skeletons
  - SignalPU, SkePU
- ...

HiHAT wishes, in a few phrases

- Have better interface to hardware layers
  - Extremely low overhead
- Reusable components which we prefer not to maintain alone
  - Performance models, allocators, tracing, debugging, …
- (ideally) Standard flexible task-based interface
  - Plus OpenMP etc. interfaces
  - Or at least set of helpers for outlining, marshaling, etc.
- Everything usable independently and interoperable

HiHAT wishlist

- Portable and flexible async APIs for driving accelerators
- Shared event management
  - Used throughout HiHAT
    - Dependencies between all kinds of requests
  - User-definable events
    - Synchronization with non-HiHAT pieces
  - Flexible event waiting API : interruptible WaitAny
    - Register a set of event
    - WaitAny(set)
    - Efficient loop over completed events
    - Can add user-defined event to the set to interrupt WaitAny easily

HiHAT wishlist

- Memory allocation
  - Uniform low-level API over devices
  - Efficient sub-allocator (cudaMalloc is not efficient)
  - Same-size pools
    - Allocation reuses
    - In RAM case, hierarchical balancing between cores/caches/NUMA
- Disk support : store/key/value, basically
  - store = plug(path)
    - key = allocate(store, size)
      - write(store, key, buffer, offset, size)
      - read(store, key, buffer, offset, size)
    - free(store, key)
  - unplug(store)

HiHAT wishlist

- Data transfer priorities
  - Take precedence over already-queued transfers
- Inter-node communication layer instead of MPI
  - Transfer priorities, again
  - Completely asynchronous and flexible event waiting API
  - Could be a PGAS : no need for messages, just memory coherency

HiHAT wishlist

- Performance models
  - We currently have history-based, linear reg. and multi-linear reg.
- Standard trace formats and debugger
  - We currently use paje, vite, temanejo

Could be moved to shared components