Parallel Applications Workshop, Alternatives to MPI+X

Held in conjunction with SC22:
The International Conference for High Performance Computing, Networking, Storage, and Analysis

SC22 Dallas, TX | hpc accelerates.

Introduction to StarPU, Legion, PaRSEC

Technologies for Developers of Parallel Applications

Olivier AUMAGE
Team STORM, Inria & LaBRI lab. at the University of Bordeaux, France — olivier.aumage@inria.fr

High-Performance Computing

Supercomputers Hardware Evolution

- Fast paced
  - Short lifetime: 5 – 10 years
- Increasing complexity
  - ORNL Frontier: ~9M cores
- Increasing heterogeneity
  - Accelerators devices, FPGA, processing offload
- Increasingly diverse purposes and designs
  - Graph / Green / Top 500, ...

- **Porting applications is difficult**
  - Must be successful in a short time

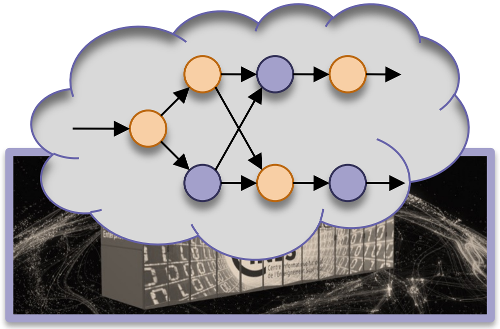
Top 10 positions of the 59th TOP500 in June 2022[30]

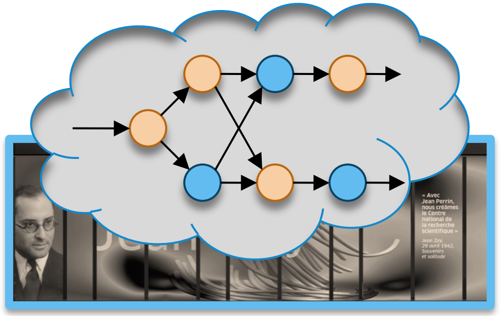
Rmax Rpeak (PetaFLOPS)	Name	Model	CPU cores	Accelerator (e.g. GPU) cores	Interconnect	Manufacturer
1,102.00 1,685.65	Frontier	HPE Cray EX235a	591,872 (9,248 × 64-core Optimized 3rd Generation EPYC 64C @2.0 GHz)	36,992 × 220 AMD Instinct MI250X	Slingshot-11	HPE
442.010 537.212	Fugaku	Supercomputer Fugaku	7,630,848 (158,976 × 48-core Fujitsu A64FX @2.2 GHz)	0	Tofu interconnect D	Fujitsu
151.90 214.35	LUMI	HPE Cray EX235a	75,264 (1,176 × 64-core Optimized 3rd Generation EPYC 64C @2.0 GHz)	4,704 × 220 AMD Instinct MI250X	Slingshot-11	HPE
148.600 200.795	Summit	IBM Power System AC922	202,752 (9,216 × 22-core IBM POWER9 @3.07 GHz)	27,648 × 80 Nvidia Tesla V100	InfiniBand EDR	IBM

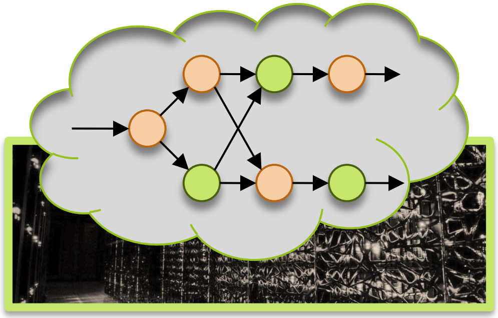
Wikipedia.org: Top500 highest ranking supercomputers (June 2022)

Applications should be expressed in a way that facilitates performance portability

Proposal







Programming for performance portability

- **Focus on expressing work instead of managing workers**
  - Rely on abstractions instead of hardware dependent work divisions
- **Confine adaptation effort to select kernel routines**
  - Tasks

- **Easier / safer / more efficient application development**
  - for human programmers
- **Easier / safer / more effective application analysis and optimization**
  - for tools

Task-based Parallel Programming

Principles

- **Separate multiple concerns**
  - General application algorithmics
  - Low-level task kernel optimization
  - Resource management and work assignment
- **Mostly fixed application structure**
  - Long term stability
- **Device-specific routines == Tasks**
  - Short term, localized optimization effort
- **Model maturity**
  - Cilk (Blumofe et al, 1995), OpenMP 3.0 standard (2008)
- **Active research ecosystem**

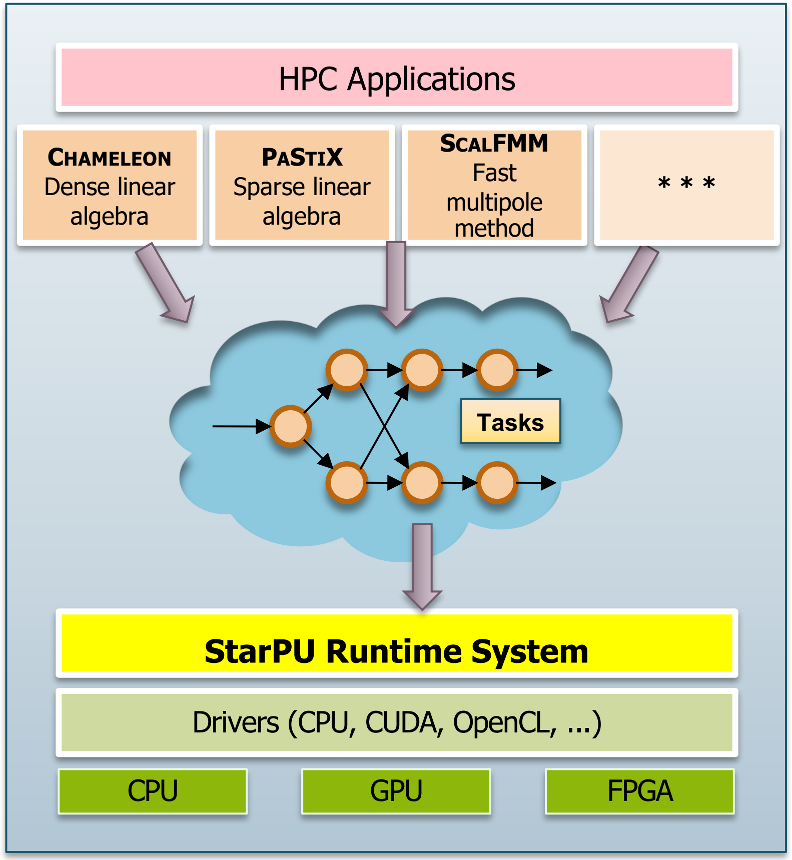
- **StarPU**
  - > Inria / LaBRI, Bordeaux
- **PaRSEC**
  - > ICL / UTK
- **Regent / Legion**
  - > Stanford
- **DuctTeip / SuperGlue**
  - > University of Uppsala
- **HPX**
  - > Louisiana State University
- **IRIS**
  - > ORNL
- **OCR**
  - > Intel+Rice University
  - > University of Vienna
- **OmpSs**
  - > BSC
- *. . . and many others . . .*

The StarPU runtime system

Inria & LaBRI lab. at the University of Bordeaux

- **Task scheduling on heterogeneous nodes**
  - General purpose cores: **CPU**
  - Specialized accelerators: **GPU**
  - Reconfigurable devices: **FPGA**

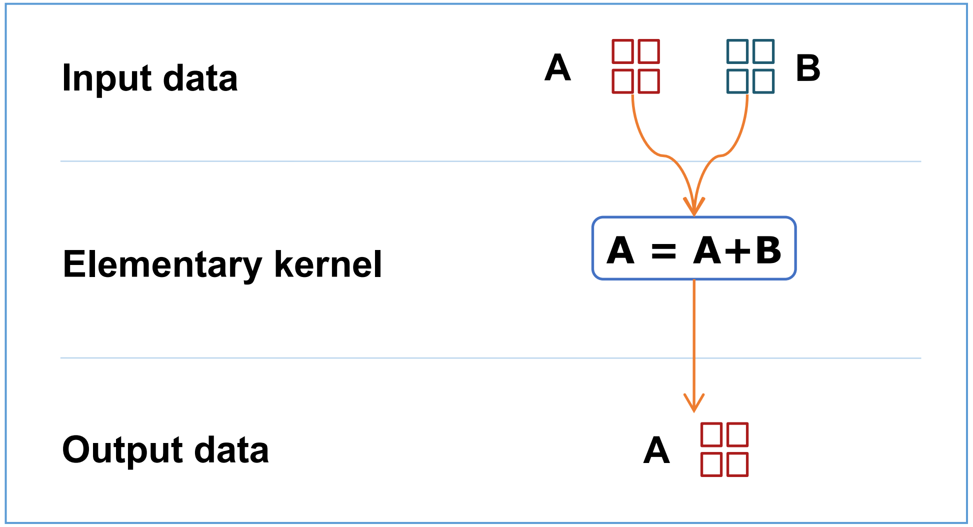
- **Usage**
  - Direct programming from application
    - C, C++, Fortran
  - Compiler / Language
    - OpenMP, Julia, Python
  - Parallel numerical libraries



StarPU — Fundamentals

Tasks + data dependencies

- **Tasks**
  - Annotated kernels
  - → **Potential parallelism**
- **Data dependencies**
  - Set of constraints
    - Input needed
    - Output produced
  - → **Degrees of freedom**



Task == kernel + data dependencies

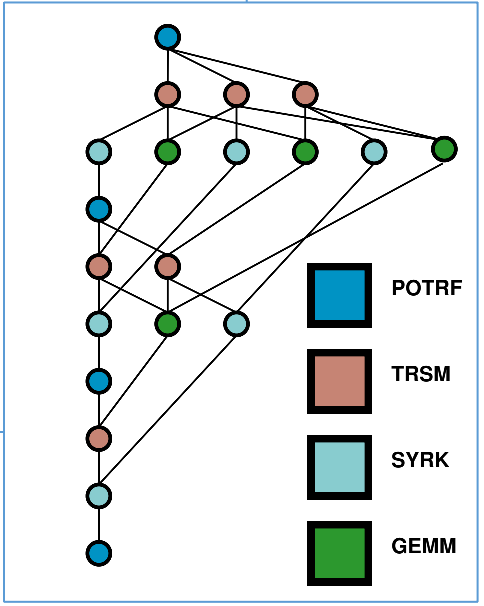
StarPU – Sequential Task Flow

StarPU programming model

- **Tasks submitted sequentially**
  - Deferred execution
- **Dependence graph built incrementally**
  - Vertex == task
  - Edge == data dependence

```
for (j = 0; j < N; j++)
{
    starpu_task_insert( POTRF (RW,A[j][j]) );

    for (i = j+1; i < N; i++)
        starpu_task_insert( TRSM (RW,A[i][j], R,A[j][j]) );

    for (i = j+1; i < N; i++)
    {
        starpu_task_insert( SYRK (RW,A[i][i], R,A[i][j]) );

        for (k = j+1; k < i; k++)
            starpu_task_insert( GEMM (RW,A[i][k], R,A[i][j],
    }
}

starpu_task_wait_for_all();
```

Flow of task submissions

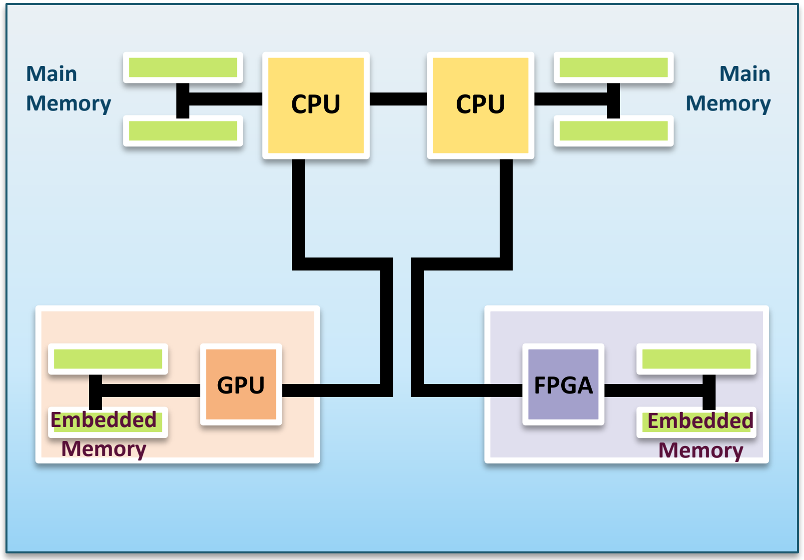


Dependence graph

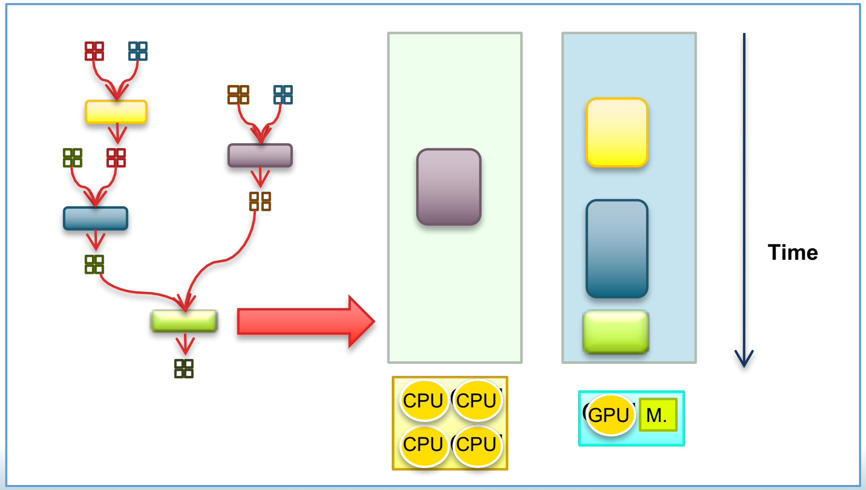
StarPU — Work Mapping

StarPU execution model

- **Programmable scheduling engine**
  - Anticipative (== planning)
  - Reactive (== work stealing)
- **Distributed Shared Memory (DSM) engine**
  - Data management
  - Data replication and consistency
- **Performance modeling engine**
  - Task execution time inference
  - Data transfer time inference



Heterogeneous computing node

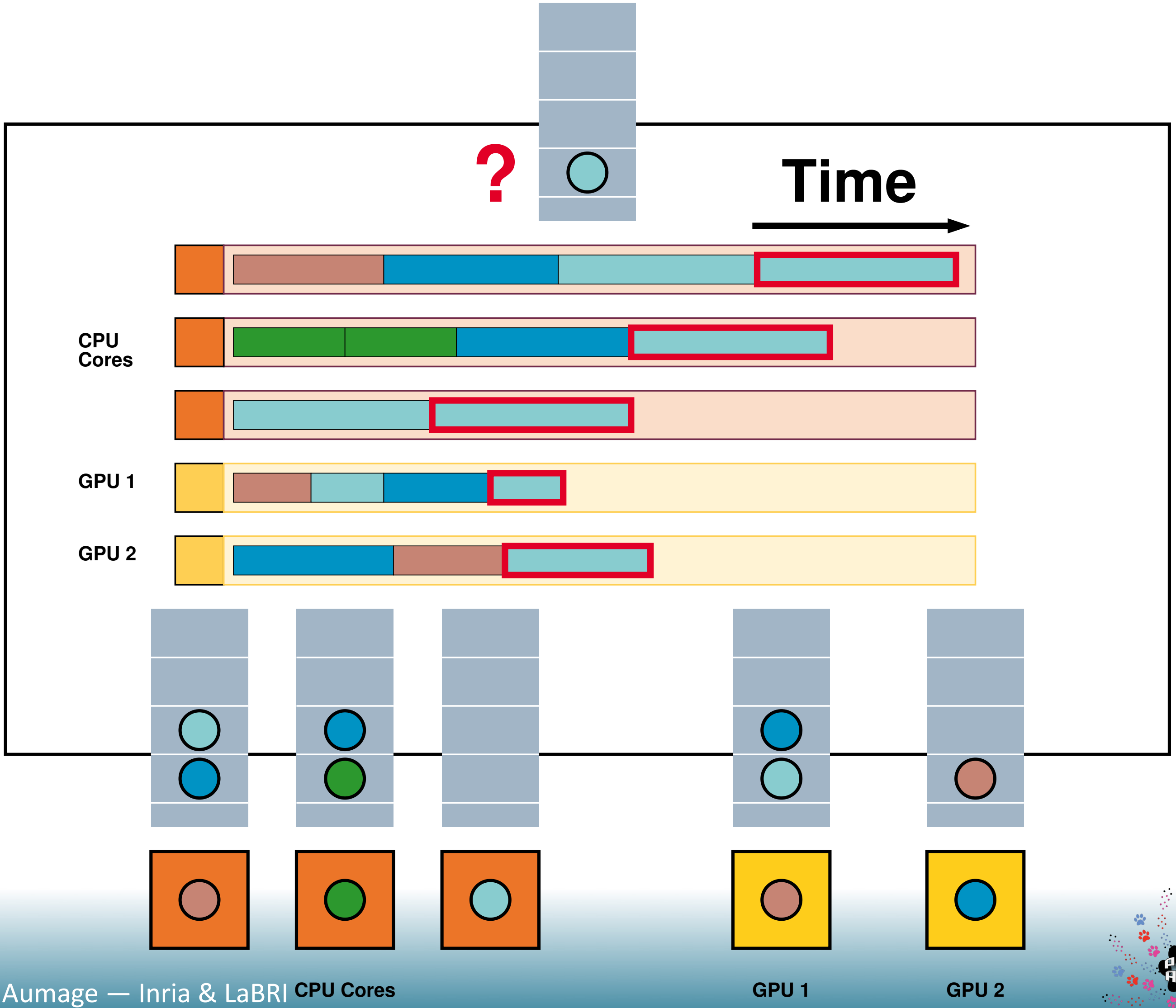


Mapping a task graph on hardware resources

StarPU — Heterogeneous Task Scheduling

Dynamically planned execution

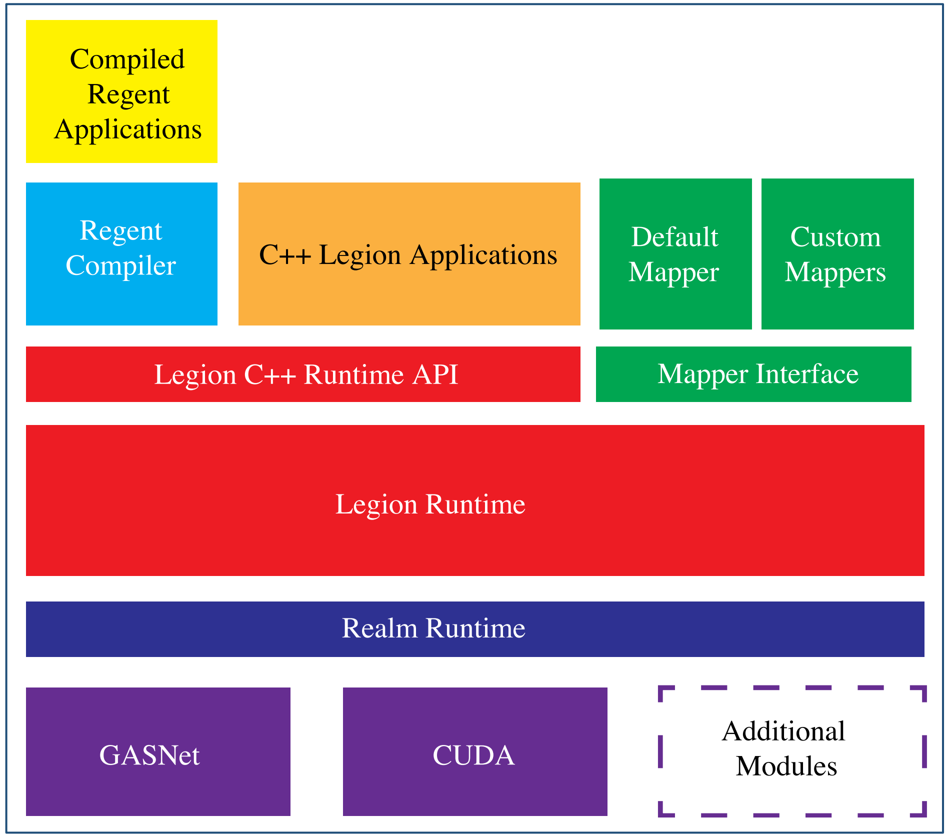
- **Kernel performance estimation**
  - Per input size
  - Per routine variant
  - Per device
- **Task execution time inference**
  - History-based
  - Custom cost function
- **Data transfer time inference**
  - Bus sampling



Legion

Stanford, SLAC, LANL, NVIDIA

- **Recursive task model for heterogeneous architectures**
  - **Data-centric task parallelism**
  - **Application-controlled mapping**
- **Usage**
  - Direct programming through C++ & Fortran API
  - Domain specific languages & libraries
  - Regent programming language and compiler



Legion architecture (credit: Legion)

Legion — Fundamentals

Notion of Logical Region

- **Abstraction for data description**
  - Relationships
  - Privileges, coherence
- **Two logical data spaces**
  - Rows == Index space
  - Columns == Field space
- **Operations**
  - Partitioning into sub-regions, on index space
  - Slicing on field space

```
const Domain domain(DomainPoint(0), DomainPoint(1023));
IndexSpace is = runtime->create_index_space(ctx, domain);
FieldSpace fs = runtime->create_field_space(ctx);

FieldAllocator allocator = runtime->create_field_allocator(ctx, fs);
FieldID fida = allocator.allocate_field(sizeof(double), FID_FIELD_A);
FieldID fidb = allocator.allocate_field(sizeof(int), FID_FIELD_B);

LogicalRegion lr = runtime->create_logical_region(ctx, is, fs);

```

Logical region with 2 fields

	Field A	Field B
0		
1		
. . .		
1023		

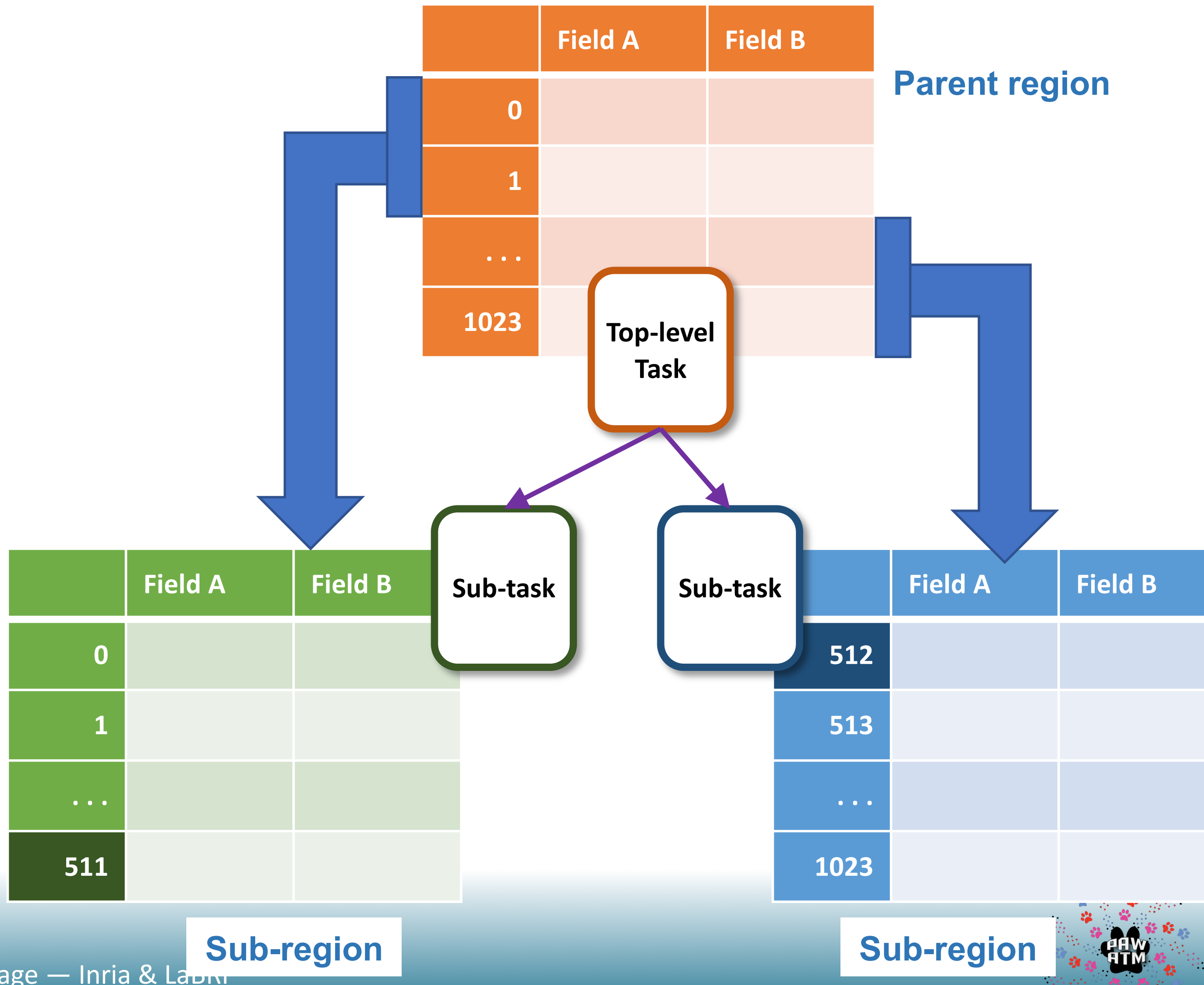
Index space

Field space

Legion — Recursive Task Parallelism

Tree of Tasks

- **Recursive spawn**
  - Top-level task
  - Sub-tasks
- **Task – Data relationship**
  - Partitioning & slicing
    == data independence
  - Privileges & coherence
    == data access mode
    - **`READ_ONLY`**, **`READ_WRITE`**, **`WRITE`**, …
    - **`EXCLUSIVE`**, **`SIMULTANEOUS`**, …



Legion — Tasks

Pure Functions

- **Constrained side effects**
  - Data access
  - Field access
  - Index space domain
- **Orthogonality**
  - Work mapping
  - Correctness

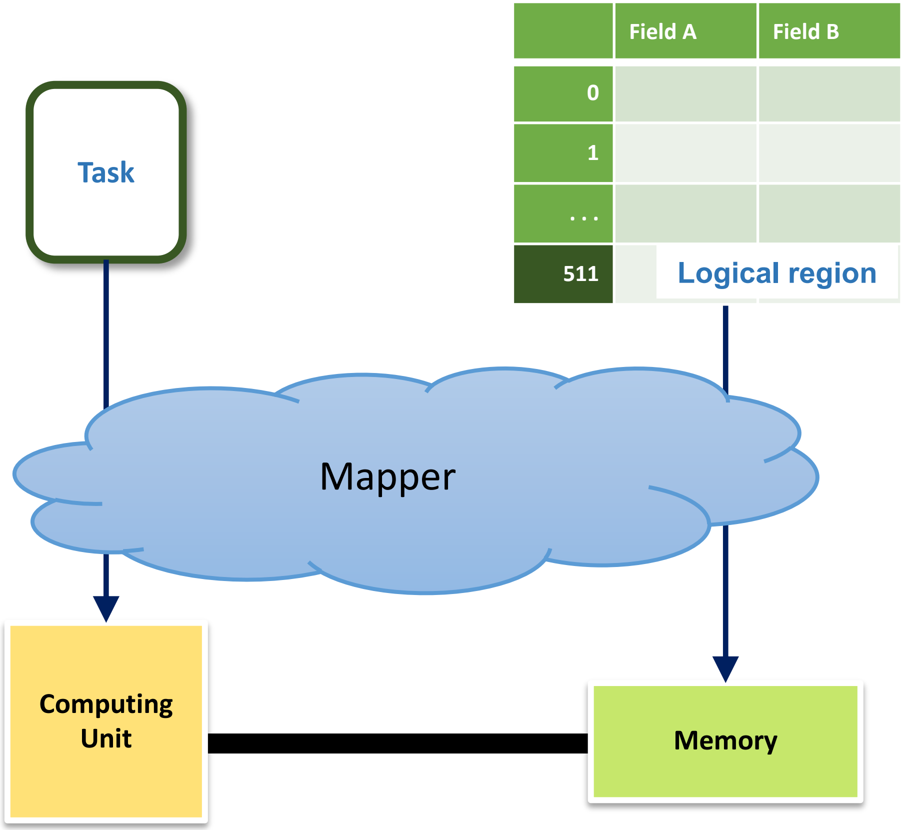
```
void daxpy_task(const Task *task,
                const std::vector<PhysicalRegion> &regions,
                Context ctx, Runtime *runtime)
{
  const double alpha = *((const double*)task->args);
  const int point = task->index_point.point_data[0];

  const FieldAccessor<READ_ONLY,double,1> acc_x(regions[0], FID_X);
  const FieldAccessor<READ_ONLY,double,1> acc_y(regions[0], FID_Y);
  const FieldAccessor<WRITE_DISCARD,double,1> acc_z(regions[1], FID_Z);

  Rect<1> rect = runtime->get_index_space_domain(ctx,
                  task->regions[0].region.get_index_space());
  for (PointInRectIterator<1> pir(rect); pir(); pir++)
    acc_z[*pir] = alpha * acc_x[*pir] + acc_y[*pir];
}
```

Example Legion Task (credit: Legion)

Legion — Execution model

Application Mapping onto Target Hardware

- **Mapper**
  - Default + custom
- **Tasks**
  - Instantiation properties
    - inlining, stealing, locality, ...
    - target kind, target unit, ...
  - Index space distribution
- **Logical regions**
  - Physical instances
  - Memory areas selection from machine topology



PaRSEC

ICL, UTK

- **Task scheduling for heterogeneous architectures**
  - **Dynamic Task Discovery (DTD) model**
    ≈≈ Sequential Task Flow (STF)
  - **Parameterized Task Graph (PTG) model**
    == compact, parametric graph representation
- **Usage**
  - Direct programming: C/C++ & Fortran
  - JDF language (PTG model) + compiler
  - DPLASMA linear algebra library

PaRSEC — Fundamentals

Dynamic Task Discovery vs Parameterized Task Graph

- **Dynamic Task Discovery (DTD)**
  - Instantiation time relation inference
  - Problem-shape independence
  - Native language API
- **Parameterized Task Graph (PTG)**
  - Compile time relation inference
  - Problem-size independence
  - Custom graph description language
    - JDF == Job Data Flow language

```
int task_hello_world( ... ) {
    /* ... */
    printf("Hello World my rank is: %d\n", this_task->taskpo
    /* ... */
}

int main(int argc, char ** argv) {
    /* ... */
    parsec_dtd_insert_task(dtd_tp, task_hello_world, ... );
    /* ... */
}
```

DTD Hello World (credit: PaRSEC)

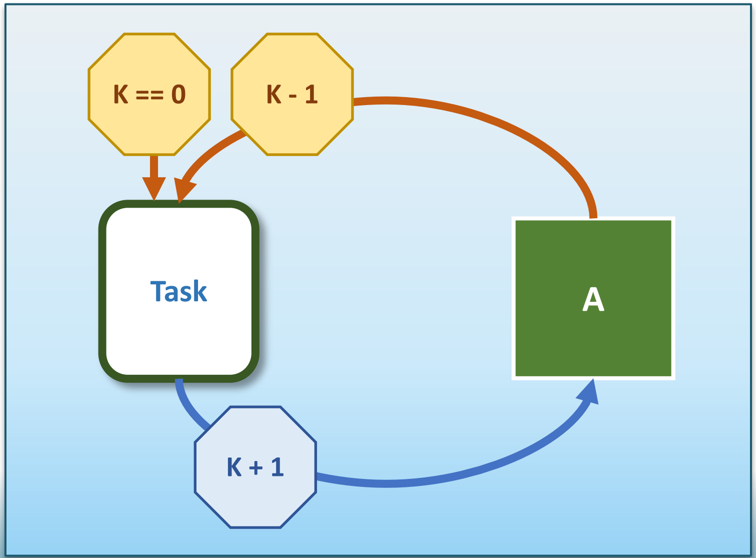
```
extern "C" %{
#include "parsec.h"
%}

HelloWorld(k)
k = 0 .. 0
: taskdist( k )
READ A <- NULL
BODY
{
    printf("HelloWorld %d\n", k);
}
END

extern "C" %{
/* ... */
%}
```

PTG Hello World (credit: PaRSEC)

PaRSEC — Job DataFlow (JDF) Language

Compact Task Graph Representation

- **Capture all task's relationships**
  - State machine
  - Notion of flows == edges




```
NB    [ type="int" ]

Task(k)

k = 0 .. NB

: taskdist( k )

RW  A  <- (k == 0) ? NEW : A Task( k-1 )
       -> (k < NB) ? A Task( k+1 )

BODY
{
    int *Aint = (int*)A;

    if ( k == 0 ) {
        *Aint = 0;
    } else {
        *Aint += 1;
    }

    printf("I am element %d in the chain\n", *Aint );
}
END
```

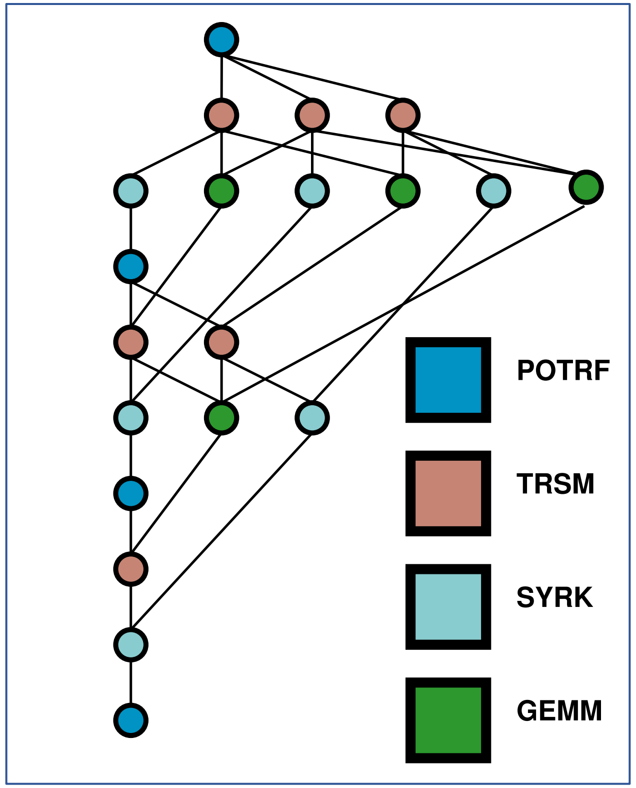
PTG Chain (credit: PaRSEC)

PaRSEC — Example: Cholesky Decomposition

Tiled DTD algorithm (Sequential Task Flow)

```
for (k=0; k<NT; k++) {
  POTRF( A[k][k] );
  for (m=k+1; m<NT; m++)
     TRSM( A[k][k], A[m][k] );
  for (n=k+1; n<NT; n++) {
    SYRK( A[n][k], A[n][n] );
    for (m=n+1; m<NT; m++)
        GEMM( A[m][k], A[n][k], A[m][n] );
  }
}
```

Sequential Task Flow Cholesky



Resulting Task Graph

PaRSEC — Example: Cholesky Decomposition

Tiled PTG algorithm: TRSM task

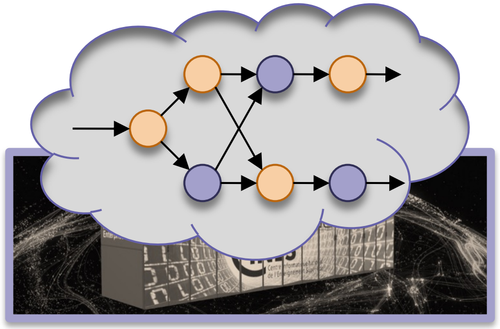
```
TRSM(k, m)

// Execution space
k = 0   .. NT-1
m = k+1 .. NT-1

// Partitioning
: dataA(m, k)

// Flows & their dependencies
READ  A <- A POTRF(k)
RW    C <- (k == 0) ? dataA(m, k)
        <- (k != 0) ? C GEMM(k-1, m, k)
        -> A SYRK(k, m)
        -> A GEMM(k, m, k+1..m-1)
        -> B GEMM(k, m+1..NT-1, m)
        -> dataA(m, k)

BODY
  trsm(A, C);
END
```

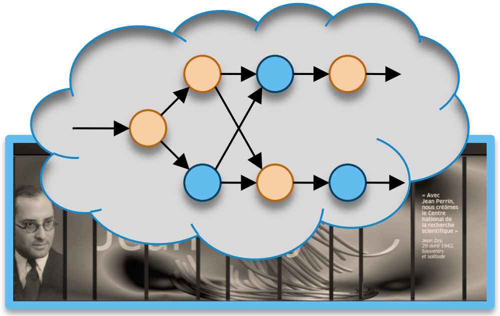
```
for (k=0; k<N; k++) {
  POTRF(RW, A[k][k]);
  for (m=k+1; m<N; m++)
    TRSM(R,  A[k][k],
         RW, A[m][k]);
  for (n=k+1; n<N; n++) {
    SYRK(R,  A[n][k],
         RW, A[n][n]);
    for (m=n+1; m<N; m++)
      GEMM(R,  A[m][k],
           R,  A[n][k],
           RW, A[m][n]);
  }
}
```

PTG Cholesky (credit: PaRSEC)

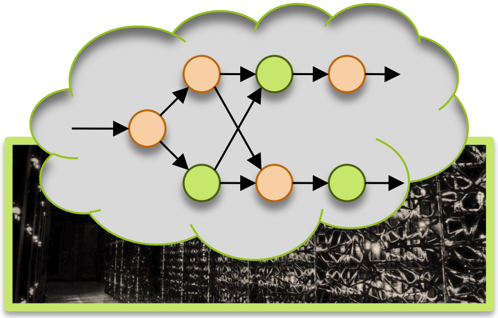
Conclusion

Task-based Parallel Programming for Performance Portability

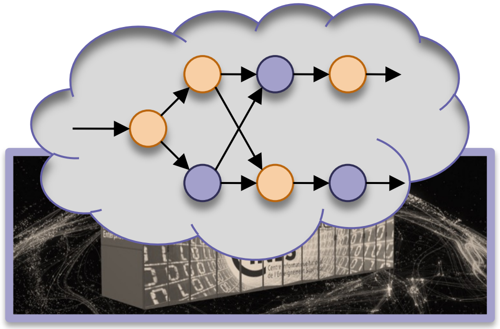




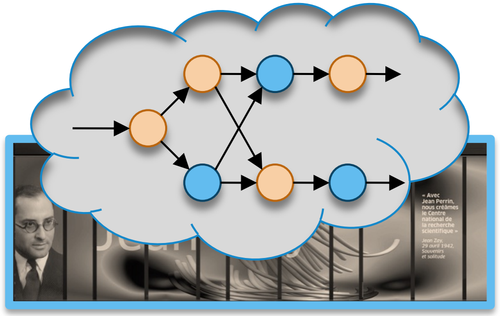
- **Easier / safer / more efficient application development**
  - for human programmers
- **Easier / safer / more effective application analysis and optimization**
  - for tools

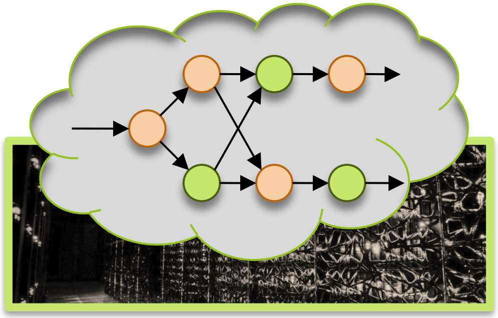


- **StarPU** — https: // starpu . gitlabpages . inria . fr
- **Legion** — https: // legion . stanford . edu
- **PaRSEC** — https: // icl . utk . edu / parsec



Thanks!





- **StarPU** — https: // starpu . gitlabpages . inria . fr
- **Legion** — https: // legion . stanford . edu
- **PaRSEC** — https: // icl . utk . edu / parsec