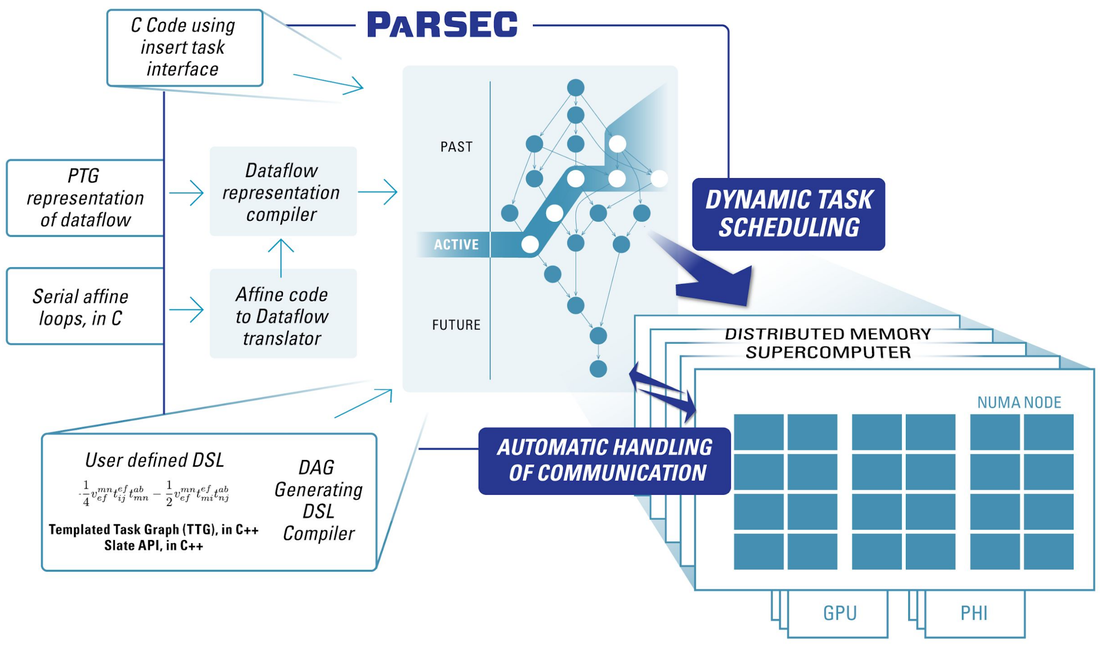
DTE

PaRSEC SYSTEM AND INTERFACES

The Distributed Tasking for Exascale (DTE) project extends the capabilities of ICL's Parallel Runtime and Execution Controller (PaRSEC) project—a generic framework for architecture-aware scheduling and management of microtasks on distributed, many-core, heterogeneous architectures. The PaRSEC environment also provides a runtime component for dynamically executing tasks on heterogeneous distributed systems along with a productivity toolbox and development framework that supports multiple domain-specific languages and extensions and tools for debugging, trace collection, and analysis.

With PaRSEC, applications are expressed as a **direct acyclic graph (DAG)** of tasks with edges designating data dependencies. This DAG dataflow paradigm attacks both sides of the exascale challenge: managing extreme-scale parallelism and maintaining the performance portability of the code.

PaRSEC **A generic runtime for Dynamic Task Scheduling on accelerated, distributed Exascale systems**



DISTRIBUTED, ACCELERATED

- ❑ Describe your distributed data collection, and let the dataflow harness the parallelism
- ❑ Asynchronous Scheduling with automatic load Balancing between many-cores and accelerators, delegation of Scheduling on Hyper-Threads
- ❑ Modular Communication System (MPI/UCX) providing Asynchronous Communication Progress and Collective Patterns
- ❑ Reliable Computation: DAG checkpointing and backward DAG recomputation available in the toolbox

INTEROPERABLE

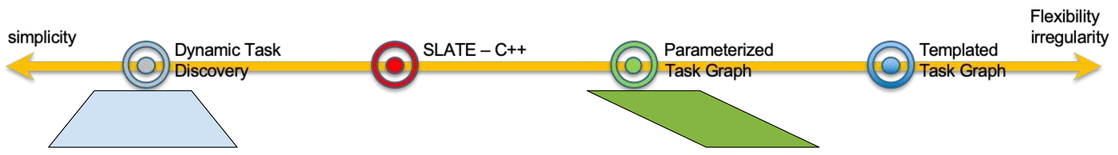
- ❑ Hardware Accelerators Support: execute CUDA, OpenMP target tasks
- ❑ PaRSEC routines can operate with MPI, OpenMP, Kokkos, etc., codes: upgrade the critical sections of a legacy application to use PaRSEC to enjoy dataflow acceleration, leave the rest unchanged
- ❑ Using ECP standard tools (SPack, CMake, MPI)
- ❑ Tested with Cray PrgEnv and IBM toolchains, simple to deploy on ECP Cori, Theta, Summit, etc.

PRODUCTIVE

- ❑ Write once, execute on any: *adding distributed memory and GPU acceleration to a PaRSEC code is simple*, thanks to implicit data movement
- ❑ Multiple Domain Specific Languages* (DSL) Support for Linear Algebra, Chemistry, etc. Code in C, Templated C++, Fortran, Python, etc.
- ❑ *Write your application in a modular way*, compose the resulting Task Algorithms at runtime, even from different DSL.
- ❑ Tools for Trace and Performance Analysis, Debugging
- ❑ Quality and Documentation (user and developer documentation, tutorials, CI testing)

ADDING PARSEC TO YOUR APPLICATION

- ❑ **Link time substitution of routines in legacy codes with a PaRSEC accelerated, GPU capable, automatically load balancing distributed routine**
  - ❑ The PaRSEC runtime is a standalone library, just link-it in, run as a normal MPI program
  - ❑ Small list of dependencies (MPI, optional CUDA, HWLOC)
  - ❑ Integrates with CMake, pkg-config to ease redistribution and integration
  - ❑ Simply link with a PaRSEC accelerated routine (e.g., DGEMM), no changes to the application code
- ❑ **Using the PaRSEC API to schedule PaRSEC DAGs**
  - ❑ Operate on PaRSEC distributed data collections for better memory access patterns
  - ❑ Schedule multiple operations at once, wait for their result later
  - ❑ Simple API (C, Fortran) manages distributed data collections, create and schedules Taskpools DAG, and wait on Taskpools completion, PaRSEC can be called from C++ and Python as well.
  - ❑ Import PaRSEC as an embedded CMake subproject, or link with a system installed PaRSEC
  - ❑ Use of-the-shelf computational and data redistribution routines, or create your own
- ❑ **Writing your own PaRSEC Dataflow routines**
  - ❑ Use one of the PaRSEC Domain Specific Languages developed in collaboration with domain scientists to express your algorithm
  - ❑ Use a generic and simple interface (task insertion from serial code) with the Dynamic Task Discovery (DTD) DSL
  - ❑ Express the maximal parallelism with the Parameterized Task Graph (PTG) DSL
  - ❑ Use C++ to mask the complexity (e.g., SLATE API) or provide flexible, customized interfaces (e.g., TTG the Templated Task Graph)




```
int task_hello_with_arg( parsec_execution_stream_t  *es,
                         parsec_task_t  *this_task )

{
  int *i;
  parsec_dtd_unpack_args( this_task , UNPACK_VALUE , &i);
  printf("Hello World my index is: %d\n" , *i);
  return PARSEC_HOOK_RETURN_DONE ;
}

int generate_tasks(void)
{
  for(int i = 0; i < 10; i++) {
    parsec_dtd_taskpool_insert_task( dtd_tp,
task_hello_with_arg,
                                     0,  "hello_world_task",
                                     sizeof( int), &i, VALUE,
                                     0);  /* No more arguments */
  }
}
```

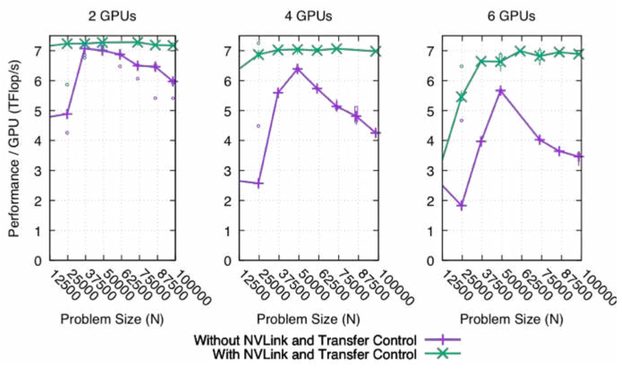
Example of DTD task insertion: 10 'hello_world' tasks are inserted in the discover_task() user function; each task, when scheduled, prints its index, found in the packed task arguments.

```
TRSM(k, m)
// Execution space
k = 0 .. NT
M = k+1 .. NT

: A(m, k)  // Partitioning

// Flows & their dependencies
READ A <- A POTRF(k)
RW   C <- (k == 0) ? A(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, m)
       -> A(m, k)
BODY
   trsm(A, C)
END
```

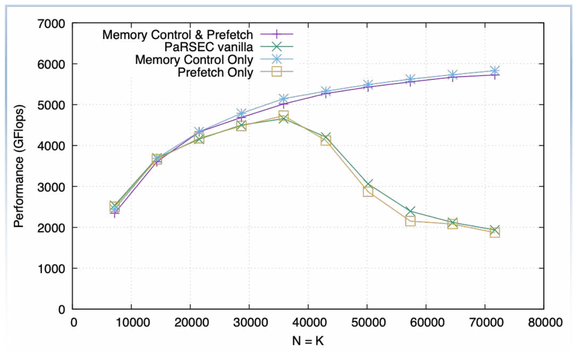
Example of PTG task description: the TRSM task is parameterized with two indices, k, and m; the task description specifies the task affinity with data, how data (A and C) flow to and from memory or from other tasks to and from the TRSM task, and what code executes the task

ADVANCEMENT IN RUNTIME CAPABILITIES

- ❑ New engine supporting multiple GPU dynamically fetches data to/from GPU memory, provides NVLink integration, and overlaps data transfers with kernel execution to operate close to peak performance even in an out-of-memory GPU memory context (problem sizes larger than the sum-memory available on the accelerators)
- ❑ Modular Communication Engine Optimized for many-core systems issues implicit communication orders and control, and overlap communication cost automatically
- ❑ Support for reliable computing (Silent Data Corruption, Checkpointing, ABFT) is available
- ❑ Extended support for irregular algorithms, both in tracking sparse and open naming of tasks and defining and providing coherence for irregular data collections
- ❑ Dynamic termination detection for irregular and data-dependent DAGs



Summit: PaRSEC Runtime uses NVLINK to increase throughput when using 6 GPUS on a Summit node and maintains close to peak performance thanks to runtime orchestrated data centric load-balancing



Tesla V100: PaRSEC Runtime prefetches relevant data blocks onto a V100 GPU and controls the block retaining policy on the GPU memory, henceforth decreasing host-device traffic for large problems that exceed the memory of a V100 accelerator.

ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY OF TENNESSEE KNOXVILLE
U.S. DEPARTMENT OF ENERGY | Office of Science
SPONSORED BY
ECP EXASCALE COMPUTING PROJECT
NNSA National Nuclear Security Administration

FIND OUT MORE AT
http://icl.utk.edu/dte/