www.intertwine-project.eu
@intertwine_eu

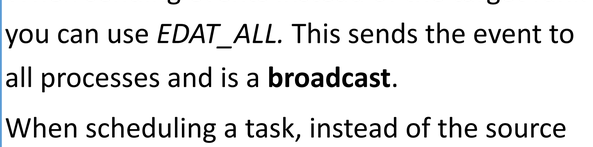
Event Driven Asynchronous Tasks (EDAT)

Nick Brown,
Oliver Brown,
Dan Holmes
epcc

Events

Fired by the programmer and sent to a process. An identifier (EID) is associated with events which is used to correlate tasks dependencies. Optionally, events can contain data. Events can be issued locally, to the same process, too.

```
void edatFireEvent(void * data, int number_elements, int data_type, int target, char * event_id)
```



Tasks

A function scheduled by the programmer which is executed when the specified events (dependencies) have been received. Tasks can depend on any number of events, identified by the event id and the source process. All events must have arrived before the task will run. The scheduling of tasks is non-blocking.

```
void edatScheduleTask(task_function, int number_dependencies, ...)
```

A non-blocking call, the schedule function accepts a variable number of event dependencies, each follows the *int source, char * event_id* format.

Collective events

When sending events instead of the target rank you can use *EDAT_ALL*. This sends the event to all processes and is a **broadcast**.

When scheduling a task, instead of the source rank you can use EDAT_ALL. The task depends on events from all ranks with the specific id and will only execute once they have been received.

What's the idea?

We present a task-based model where tasks are scheduled and depend upon a number of events arriving before they can execute. The programmer explicitly understands they are working in a distributed memory environment, interactions being driven via events which can be sent remotely or locally. Events may or may not have some data associated with them and tasks execute independently from others.

① *Task is scheduled on process 0 and has no dependencies, so it will run on a worker thread as soon as one is available*

```
int main() {
  if (edatGetRank() == 0) {
    edatScheduleTask(task1, 0);
  } else if (edatGetRank() == 1) {
    edatScheduleTask(task2, 1, 0, "evt1");
    edatScheduleTask(task3, 2, 0, "evt2", 1, "evt3");
  }
  return 0;
}

void task1(EDAT_Event * events, int num_events) {
  edatFireEvent(NULL, 0, EDAT_NONE, 1, "evt1");
  int data=33;
  edatFireEvent(&data, 1, EDAT_INT, 1, "evt2");
}

void task2(EDAT_Event * events, int num_events) {
  int data=100;
  edatFireEvent(&data, 1, EDAT_INT, EDAT_SELF, "evt3");
}

void task3(EDAT_Event * events, int num_events) {
  printf("%d\n", *((int*) events[0].data) + *((int*)
                 events[1].data));
}
```

② *This task fires an event, with no payload data to process 1 with id "evt1".*

③ *This task fires an event, with a single integer as data to process 1 with id "evt2".*

④ *This task fires an event, with a single integer as data to itself with id "evt3".*

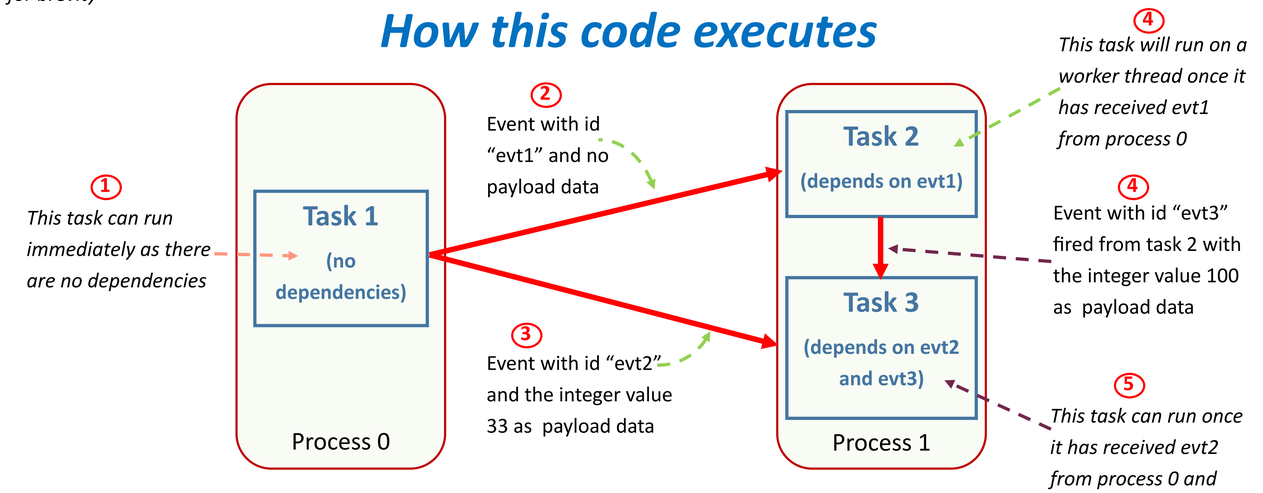
⑤ *This task extracts out both events and adds the associated integer data together.*

Note: Error checking on the number of events and data types omitted for brevity

This task is scheduled on process 1 and has one dependency which is an event with id "evt1" from process 0

This task is scheduled on process 1 and has two dependencies, an event with id "evt2" from process 0 and one from process 1 with id "evt3"

How this code executes



Resilience (ACID compliance): A parallel code's state can be described by outstanding events and scheduled tasks. These can be stored in a ledger. For atomicity we can delay a task's firing of events until that task has completed which avoids partially completed tasks in the event of hardware failure.

How's this different from Message Passing?

- As well as containing data, events explicitly activate tasks that depend upon them.
- Tasks are independent and will not interact directly with other tasks (only via events).
- With MPI collective communication the issue order on each process matters. EDAT supports collective events, but these are matched on event id instead.

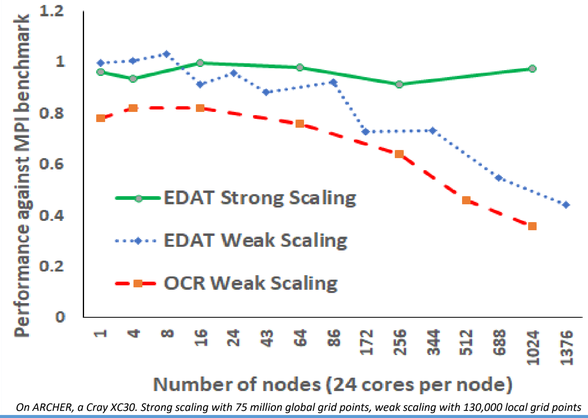
How's this different from other task based models?

- Other task based models are often oriented around shared memory. Supporting distributed memory architectures requires the underlying runtime to make important decisions around communications that is hidden from the programmer's view.
- Many task models, (e.g. OpenMP tasks & OmpSs) require compiler support, EDAT is all library based.

Use cases

- Codes with irregular communications
- Incrementally apply to existing HPC codes & support loosely coupled parallelism via asynchronous tasks.

Intel 2d stencil benchmark



On ARCHER, a Cray XC30. Strong scaling with 75 million global grid points, weak scaling with 130,000 local grid points

EDAT is open source, github.com/EPCCed/edat

For more information email n.brown@epcc.ed.ac.uk