Enabling a Data-Centric Model on the Open Community Runtime

Sri Raj Paul (Student), Vivek Sarkar
Rice University, Houston, TX, USA
{sriraj, vsarkar}@rice.edu

Bala Seshasayee, Romain Cledat, Vincent Cave
Intel Corporation, Hillsboro, OR, USA
{bala.seshasayee, romain.e.cledat, vincent.cave}@intel.com

ABSTRACT

Achieving Exascale performance requires addressing challenges arising from the complexity of Exascale architecture, its power constraints, as well as application complexity. Exascale architectures, expected to consist of millions of heterogeneous cores and extremely non-uniform hierarchical memory, have to optimally distribute applications, data and parallelize computation. Dynamic task-based execution models hold promise in achieving this, as they help express fine-grained parallelism, along with decoupling computation and data from underlying resources. Open Community Runtime(OCR) is a community-led effort to explore various asynchronous task-parallel runtime principles that can support a broad range of higher-level programming constructs. Legion is a data-centric programming model in which the runtime extracts task-based parallelism from programs, freeing the developer from having to explicitly express them. In this poster we describe our efforts to run Legion programs on top of OCR as their underlying runtime, to combine the parallelism extracted by Legion with the flexibility provided by OCR.

1. INTRODUCTION

Future Exascale systems require several challenges to be tackled by the software, such as (i) addressing their hierarchical organization of processor and memory, (ii) heterogeneity in parallelism and (iii) short mean time between failures. At the same time, it is important to maintain the productivity and expressibility of today's high level application programming constructs. The Exascale runtime is expected to bridge this gap between high level programming models and the complex low level hardware, while at the same time decoupling the computation and data from the underlying system resource management. Task-based runtimes, with their fine-grained parallelism has recently been studied as a suitable solution in this scenario. Computations, decomposed into a large number of asynchronous tasks, can be distributed across multiple processing elements to achieve load balance. To support task migration, data used in a computation can be represented as relocatable data objects. Asynchronous Many Task models (AMT) includes the concepts of asynchronous tasks and relocatable data objects. The Open Community Runtime(OCR) [2] is a community led effort to address the challenges of Exascale systems using the AMT principles.

Legion [1] a high-level programming model and runtime system that targets distributed heterogeneous architectures. Data is represented in Legion using *logical regions* and computation using *tasks*. Based on the information provided using *logical regions*, Legion automatically extracts parallelism, data movement and required synchronization, thus removing those burdens from the application programmer.

This project aims to use OCR as the asynchronous task based runtime supporting the Legion framework to combine the benefits of OCR runtime such as dynamic adaptation and resiliency, with the ease of expressibility and productivity of programming in the Legion framework. We also hope to explore the challenges involved in expressing a high level runtime on a low-level one, where both the runtimes are task-based and independently designed for scalability.

2. LEGION SOFTWARE STACK

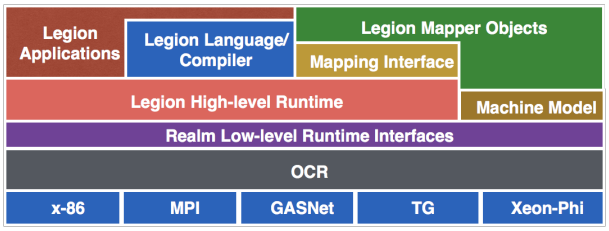

Figure 1: Legion software stack after the inclusion of OCR.

Figure 1 shows the Legion software stack [1] after the inclusion of OCR. It retains the machine independent Legion applications and the machine-dependent mappers that control the mapping of Legion applications to target architecture. High-level runtime converts the Legion application to operations of the low-level runtime (termed Realm [3]) with inputs from the mapper. The low-level operation primitives are mapped to OCR APIs. Ideally, the high-level runtime can directly use OCR without going through Realm interfaces, however this would involve extensive rewriting of the high-level runtime. As seen later, Realm has many common features with OCR that allows an easier translation at the low-level.

3. OCR OBJECTS

An OCR program can be represented using a DAG. Each node in the graph represent one of the three objects listed below and the edges represent either data or control dependence.

[1]Based on figure from http://legion.stanford.edu/pdfs/bauer_thesis.pdf

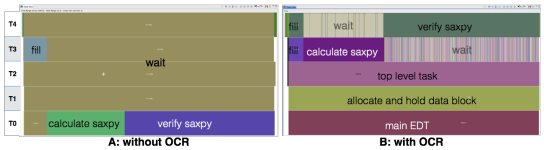

Figure 2: Comparison of execution traces of realm_saxpy programs. X-axis represents time and Y-axis represents different threads. Both the graphs have comparable execution times.

1. `Event Driven Task(EDT)`: is used to perform a computation. As the name suggests, it refers to a task that is dependent on other objects for its execution. The dependences are expressed through explicit ordering, termed "pre-slots". An `EDT` is scheduled for execution when all its dependences have been met.
2. `Data Block(DB)`: is a relocatable, contiguous block of memory.
3. `Event`: is the synchronization mechanism in OCR and is used to express the dependences between `EDT`s and other `Event`s. There are a few types of `events` used according to various synchronization needs as described below.
   - `Once-event`: is the default event type which is "triggered" when any of its dependence is satisfied. It is destroyed upon satisfaction.
   - `Sticky-event`: is the same as `once-event` except that it is not destroyed automatically.
   - `Latch-event`: has an increment and a decrement pre-slot, and gets triggered when it receives an equal number of satisfactions in both pre-slots.
   - `Channel-event`: triggers when it has been satisfied a certain number of times(n), and the same number of dependences has been added to it. It gets reset after it is triggered and can be reused.

4. MAPPING REALM TO OCR

Realm interfaces mainly include *Tasks* for computation, *Events* for dependency, *Regions* for data and *Reservations* for synchronization. A *task* is mapped to an `EDT`. An *event* is a persistent entity and therefore mapped to a `sticky-event`. Realm allows merging of *events* to create a conjunction of a set of *events*. To merge n *events*, we create a `latch-event` whose increment pre-slot is satisfied n times and attach the n *events* to its decrement pre-slot. Realm abstracts memory using *memory* objects and *regions* are allocated in the *memory*. We mapped *memory* to a `data block`, and thus *region* interfaces worked without any rewriting.

Reservation is a synchronization mechanism to replace locks in a deferred execution environment. *Reservations* are requested using *acquire* API and instead of waiting it returns immediately with an *event* that gets triggered when the *reservation* is granted. *Reservations* are released using the *release* API. A *Reservation* is mapped to a `channel-event` of size one. During *acquire*, one dependence is added to the `channel-event` and during *release* the `channel-event` is satisfied once. To enable the first *acquire* to proceed without waiting, the `channel-event` is satisfied once during creation.

Listing 1: Original hello_world Legion program

```
1  int main(int argc, char **argv){
2  ..
3  HighLevelRuntime::register_legion_task<
       hello_world_task>(.., Processor::LOC_PROC
```

Listing 2: hello_world Legion program after integration of OCR

```
1  int legion_ocr_main(int argc, char **argv){
2  ..
3  HighLevelRuntime::register_legion_task<
       hello_world_task>(.., Processor::OCR_PROC
```

5. RESULTS AND OPEN ISSUES

As a first step, after mapping *tasks*, *events* and *regions* we are able to run the realm_saxpy Realm program provided in the Legion code base using OCR as the underlying runtime. A comparison of execution trace collected using Rice University's HPCToolkit without and with OCR is shown in Figure 2. OCR disallows synchronizations within a task, and consequently, any blocking operation uses spin-wait, unlike Realm which uses thread blocking. As a result, the top_level_task and mainEdt which calls `wait` get sampled only in the trace while using OCR. The current Legion-OCR implementation uses an additional thread to keep the pointer to data block alive and valid. Additionally after mapping *reservations*, we are able to run the hello_world Legion program from the code base with minimal changes (shown in red) as illustrated in Listing 1 and Listing 2.

Currently, we can run the examples mentioned above on a single node. Our current and future work involves extending these to a distributed environment that involves mapping of a *region* rather than *memory* directly to a `data block`. Legion is an implicit task dependence programming model where the high-level runtime extracts the dependences during a scheduling window and maps them to an explicit dependence programming model such as Realm or OCR. Another promising direction for improving performance is to use a compiler to extract the dependences from a Legion program and directly emit OCR code. We are also exploring the use of blocking tasks rather than busy-wait, to allow better utilization of CPU cores, and evaluate its improvement.

Acknowledgment

We would like to thank Sean Treichler for the insightful discussions regarding the Legion runtime.

6. REFERENCES

[1] M. Bauer et al. Legion: Expressing Locality and Independence with Logical Regions. SC'12. IEEE/ACM, 2012.

[2] T. Mattson et al. The Open Community Runtime: A Runtime System for Extreme Scale Computing. HPEC'16. IEEE, 2016.

[3] S. Treichler et al. Realm: An Event-Based Low-Level Runtime for Distributed Memory Architectures. PACT'14. ACM, 2014.