The High Performance Open Community Runtime: Explorations on Asynchronous Many Task Runtime Systems

Joshua Landwehr†, Joshua Suetterlein∗, Andrés Márquez†, Joseph B. Manzano† Kevin J Barker† and Guang R. Gao∗
∗The Computer Architecture and Parallel Systems Laboratory
University of Delaware, Newark, Delaware 19716
Email: jodasue@udel.edu ggao.capsl@gmail.com
†Pacific Northwest National Laboratory
Richland, Washington 99354
Email: {joshua.landwehr,andres.marquez,joseph.manzano}@pnnl.gov

Abstract—The following presents the Performance Open Community Runtime, an asynchronous many task runtime which aims to provide a scalable, efficient, and productive platform to exercise novel runtime ideas that will exploit the massive parallel resources in today’s HPC systems. This mature platform goes beyond the proof-of-concept phase by demonstrating that it can scale up to thousands of cores. Moreover, it is highly configurable, aiding in the controlled exploration and validation of novel ideas and concepts. We have proven its scalability for both strong and weak scaling experimental setups with selected kernels running on leadership clusters. We have conducted preliminary studies with different memory models and used our own characterization / introspective framework for an in-depth attribution of performance.

I. INTRODUCTION

The trend for new hardware designs in HPC has been towards more parallel cores, esoteric networks on chips, and different types of memory organizations that aim to facilitate parallelism (more bandwidth, logic layers, deeper memory hierarchies, etc). However, current software stacks are still immature dealing with increasing amounts of heterogeneous resources that these designs present. Even when the resources themselves are fixed, power states, frequency domains, resiliency aspects and pathological behaviors on shared resources will introduce non-deterministic latencies that current software stack paradigms are ill prepared to exploit.

A new generation of software stack concepts are being explored and tested in several sandbox environments. Systems like CILK, Global Arrays, CHARM++ among others have proven that fine grain tasking and global addressing spaces can help to develop more agile and robust systems in the face of computational heterogeneity and non deterministic latencies. Although these efforts guided the initial steps, there are other concerns that are essential to explore in providing more adaptable, efficient, and productive software stacks. Questions like what is the trade off between programmability and the need for coherence in global address spaces, runtime overhead characterization and its interplay with scheduling, data layout and task granularity, the feasibility to work-steal across nodes, etc. All the while, there is a need in ensuring that these questions are answered with high performance in environments that are efficient and scalable. Thus, they can be disseminated and adopted rather efficiently by application developers and system designers.

This poster presents the Performance Open Community Runtime (P-OCR), an incarnation of the Open Community Runtime introduced in the Extreme Scale Software Stack (XSTACK) project from the Department of Energy. This mature platform goes beyond proof-of-concept to provide a scalable and configurable platform to test research ideas in the asynchronous many task runtime arena. We have proven its scalability for both strong and weak scaling experimental setups with selected kernels running on leadership clusters. We have conducted preliminary studies with different memory models and use our own characterization / introspective framework for an in-depth attribution of performance.

II. THE OPEN COMMUNITY RUNTIME

Under the OCR programming model, computation is represented as a data acyclic graph (DAG) and it is composed of three primitive objects: Event Driven Tasks (EDTs), Data Blocks (DBs), and events. OCR’s abstraction for computational work is the concurrent, asynchronous, dependency based EDT. The various data and control dependencies of an EDT are expressed with events forming a computational graph. A typical EDT will consume data and satisfy a dependency with either modified or new data processing an application. DBs are the global data abstraction for OCR and provide an encapsulation for higher level data structures. DBs are augmented with user annotated access properties, giving the runtime the opportunity to exploit various optimizations based on parallelism and locality. All OCR primitives can be globally addressed across the system by a unique identifier (GUID), and can be migrated anywhere in the system.

By abstracting several low level minutiae from the programmer, the runtime is enabled to efficiently schedule and layout its own computation and data. As long as the application programmer expresses the algorithm as a composition of OCR’s primitive objects, the system designer is free to explore different techniques to map to the underlying hardware. For more information about OCR fundamentals, please refer to [2].

III. KEY P-OCR FEATURES

P-OCR is a high performance implementation of the OCR standard and serves as a platform to introduce novel runtime concepts. P-OCR is divided into two broad interconnected component classes: the intranode components which handle OCR requests and maintenance within a node, and the internode components which handle the communication across nodes between two instances of the intranode components. Key features of the runtime include a system wide out-of-order engine, which allow for a truly asynchronous creation and access of OCR primitives; a novel memory model that allow system wide coherence while preserving the scalability of P-OCR; and a fully instrumented runtime permitting P-OCR certain degree of introspection and characterization of its performance.

System Wide Out-of-Order Engine The operation of creating or requesting an OCR primitive is usually considered to be a blocking operation in which an ack is necessary to continue
computation. This might create performance degradation while waiting for the creation or request to come back, especially in larger systems. Under P-OCR, the creation and request operations are truly asynchronous since any requests for a yet-to-be-created object are handled by an out-of-order engine. When a creation or request for access arrives, the Global Unique Identifier Table is accessed. If there is no entry for such an object, a new entry is created and the request is queued. Any subsequent requests will be added to that queue and satisfied in order once the object is created. This allows for a non-strict behavior when creating and requesting OCR objects and permits a more efficient use of resources inside the runtime.

**Novel Memory Model for Consistency at Scale** OCR's default memory model uses the concept of acquire and release pairs to maintain coherency across the entire distributed system. This approach can create large amounts of network traffic becoming a performance bottleneck for more write intensive access patterns. By leveraging the properties of the OCR computational graph and restricting competing writes, P-OCR uses an invalidation-signal delay memory protocol to ensure coherency at scale while still preserving the application parallelism. Under this protocol, an invalidation is sent to the global object while the requesting EDT is working on the data block at the same time. To prevent any inconsistencies, all outgoing signals are delayed until an invalidation acknowledgment is received. This allows for a consistency view of the data blocks at scale while allowing to overlap the computation with the protocol operations.

**Introspective Framework** One of the most important questions for any runtime developer – and by extension, compiler and algorithm developer – is how much overhead does my API / runtime incur for a given application. To solve this question (and many others), P-OCR features a fully instrumented runtime. Each runtime event can be collected in both the frequency and time domains and visualized (with additional tools). This allows the attribution of perceived application behaviors to different runtime components or to the application itself. Furthermore, the visualizer allows a high level view of this behavior and ensures that coarse behaviors can be identified rather efficiently.

### IV. Testcases

To test P-OCR, we utilize four high performance clusters located in two National Laboratories: The Constance and PAL clusters [5] located in Pacific National Laboratories and the Cori and Edison supercomputers located in Berkeley National Laboratory [3], [4]. The Constance cluster is composed of 300 nodes each with 24 core Haswells connected through FDR Infiniband network. PAL is a cluster composed of 144 nodes, each with 32 AMD Barcelona cores connected through Infiniband. The Edison Supercomputer is composed of 5576 nodes, each with 24-cores Ivy bridge connected through a Cray Aries Interconnect. Finally, Cori is composed of 1630 nodes, each with 32 Haswell cores, also connected through a Cray Interconnect. The different configurations used can be seen in Table 3.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Nodes</th>
<th>Threads</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL</td>
<td>128</td>
<td>16</td>
<td>Strong</td>
</tr>
<tr>
<td>Constance</td>
<td>400</td>
<td>22</td>
<td>Strong</td>
</tr>
<tr>
<td>Cori</td>
<td>1024</td>
<td>30</td>
<td>Strong</td>
</tr>
<tr>
<td>Edison</td>
<td>1024</td>
<td>20</td>
<td>Weak</td>
</tr>
</tbody>
</table>

### V. Conclusions

This poster demonstrates the P-OCR framework and its different capabilities that provide an efficient testbed to test different runtime ideas and methodologies. Our framework is currently being used to explore the effects of the network backends, configurations, task scheduling, and data layouts for asynchronous, fine-grain execution models, and has proven to be an invaluable tool for understanding non-determinism.

### VI. Acknowledgments

This work was supported by the DOE Office of Science, Advanced Scientific Computing Research, under award number 66678 and performed at Pacific Northwest National Laboratory (PNNL). PNNL is operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830.

### References