Exploring Asynchronous Many-Task Runtime Systems toward Extreme Scales

Samuel Knight (Florida Institute of Technology), Marc Gamell (Rutgers University), Gavin Baker (California Polytechnic State University), David Hollman (Sandia National Labs), Gregory Sjaardema (Sandia National Labs), Hemanth Kolla (Sandia National Labs), Keita Teranishi (Sandia National Labs), Jeremiah Wilke (Sandia National Labs), Nicole Slattengren (Sandia National Labs), Janine C. Bennett (Sandia National Labs)

Introduction

Asynchronous many-task (AMT) runtime solutions
- Show promise at sustaining performance in spite of system performance heterogeneity
- Task-graph
- Nodes are work (tasks) are data dependencies
- Active area of research
  - Charm++, PaRSEC, HPX, Legion, OCR, STAPL, Uintah, StarPU, Swift/T

Comparative Analysis of AMT Runtime

Asynchronous many-task programming models are a leading new paradigm with many variants:
- Goal: Address knowledge gaps
- Comparative analysis of leading candidate solutions
- Qualitative & quantitative tests using AMI relevant workloads

Outcome: Guidance to code development roadmap for next-generation platforms for AISC/Inigrid/Codes

We selected Charm++, Legion and Uintah
- Demonstrated science applications at scale
- Maturity of runtime
- Three very different implementations, APIs, sets of abstractions
- Accessibility of drivers

Target Application: MiniAero
- 3-dimensional, finite volume, CFD
- Runge-Kutta fourth order time marching
- Options for 1st or 2nd order spatial discretization
- Inviscid Roe and Newtonian fluxes
- Baseline 3800 lines C++ (MPI + Kokkos)

Qualitative Evaluation

Programmability: Does this solution enable expression of our workloads?
Maturity: Ease of adopting this solution, modifying it to suit our needs?
Qualitative measures can help assess programmability and maturity

<table>
<thead>
<tr>
<th>Measure</th>
<th>Charm++</th>
<th>Legion</th>
<th>Uintah</th>
<th>MiniAero</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maturity</td>
<td>How stable is the API?</td>
<td>How easy is the API?</td>
<td>How easy is the API?</td>
<td>How stable is the API?</td>
</tr>
<tr>
<td>Fault tolerance support</td>
<td>What fault models/recovery mechanisms are supported?</td>
<td>What fault models/recovery mechanisms are supported?</td>
<td>What fault models/recovery mechanisms are supported?</td>
<td>What fault models/recovery mechanisms are supported?</td>
</tr>
<tr>
<td>Dynamic workload support</td>
<td>What load-balancing/work stealing mechanisms are supported?</td>
<td>What load-balancing/work stealing mechanisms are supported?</td>
<td>What load-balancing/work stealing mechanisms are supported?</td>
<td>What load-balancing/work stealing mechanisms are supported?</td>
</tr>
<tr>
<td>Modularity (Runtime and Application)</td>
<td>How reusable are components?</td>
<td>How extensible is the framework?</td>
<td>How reusable are components?</td>
<td>How extensible is the framework?</td>
</tr>
<tr>
<td>API</td>
<td>What do developers like/dislike regarding the interface?</td>
<td>What do developers like/dislike regarding the interface?</td>
<td>What do developers like/dislike regarding the interface?</td>
<td>What do developers like/dislike regarding the interface?</td>
</tr>
</tbody>
</table>

Our internal survey indicates that there is no runtime solution that addresses all of our needs from qualitative (and subjective) perspectives

Summary of Qualitative Evaluation

<table>
<thead>
<tr>
<th>Runtime</th>
<th>Key Strengths</th>
<th>Key Limitations</th>
<th>Application perspectives</th>
<th>Computing time-to-solution leverage points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legion</td>
<td>Dynamic load-balancing, cooperative task scheduling</td>
<td>Rigidity of data model, direct support for non-blocking/data fetch model</td>
<td>Numerical and algorithmic approach</td>
<td>Computing time-to-solution leverage points</td>
</tr>
<tr>
<td>Charm++</td>
<td>Flexible data movement patterns, supports push model</td>
<td>Lack of data model, template support, range of application and performance</td>
<td>Two parallel codes for different runtime</td>
<td>Computing time-to-solution leverage points</td>
</tr>
<tr>
<td>Uintah</td>
<td>Application-driven approach very performant for specific use cases</td>
<td>Lack of support for unstructured meshes</td>
<td>Application-driven approach very performant for specific use cases</td>
<td>Computing time-to-solution leverage points</td>
</tr>
</tbody>
</table>

Conclusions

Pros:
- AMT runtimes show tremendous potential for addressing extreme-scale challenges
- The collective research being performed by AMT RTS developers is critically important precursor to establishing community standards

Cons:
- All leading AMT programming models and RTS have been designed or demonstrated on a limited set of applications
- None of the runtimes appear to satisfy all requirements of our application workloads (definition of requirements is still in progress)
- None of the runtimes are production ready for a broad class of applications

Acknowledgements

- National Energy Research Scientific Computing Center
- Alex Aiken, Woonchan Lee and Elliot Slaughter (Stanford University)
- Michael Bauer and Sean Treseler (NVIDIA)
- Rajini Jain, Laium Jaffe and Eric Millic (University of Illinois, Urbana-Champaign)
- Martin Berens, Todd Harman, and Alan Humphrey (University of Utah)
- Robert Clay, David Dedonis, Ryan Grant, Simon Hammond, Victor Kohn and Stephen Oliver (SLN)

References