Yearly Project Progress Report

DynAX: Innovations in Programming Models, Compilers and Runtime Systems for Dynamic Adaptive Event-Driven Execution Models

Award Number: DE-SC0008716
Dates of Performance: 9/1/2013-8/31/2014
Report Date: 5/30/14

Principal Investigator
Guang Gao
ET International Inc
100 White Clay Center Dr, Newark DE 19711

Co-PIs:
Benoit Meister, Reservoir Labs, Inc
David Padua, University of Illinois Urbana Champaign
John Feo, Pacific Northwest National Laboratories

Introduction

This report outlines the work that was done in the first three quarters (i.e. Q5, Q6 and Q7) of year 2 by the DynAX Team (ET International, Reservoir Labs, UIUC, and PNNL) as well as the projected work for the remainder of the contract.(Q8).

Accomplishments

The accomplishments for the year are broken down into four sections corresponding to each of the four teams: ETI, Reservoir, UIUC, and PNNL. Note that these accomplishments have already been reported in more details in our Q5, Q6 and Q7 reports -- the Year 2 report is more an extended summary and highlights.

ETI Accomplishments

In Year 2, ETI studied two applications: LULESH and TCE. ETI also extended SCALE.

LULESH

LULESH 2.0 demonstrates a workload imbalance that is consistent across iterations. In LULESH, this workload imbalance is represented as a differing loop count for certain regions. Regions with high loop factors will always have more work to do than other regions. In some cases, the loop count increases the workload by more than a factor of 10.

We spent a few weeks studying LULESH version 2.0. We understood the basic physical properties of the application, and proposed a few serial performance improvements. We then studied the parallel performance characteristics, and came to the conclusion that a direct SWARM adaptation of this version would have roughly equivalent performance to the MPI+OpenMP version provided by LLNL, and would be difficult to improve. We believe that a more fruitful approach may be to re-balance the static workload, by splitting up the volume of (simulated) space differently. The goal would be to repartition the space such that the computational load is equal, while minimizing the amount of data transmitted between compute nodes (i.e. minimizing the surface area between the partitions).

We have been told that a future version of LULESH will incorporate a viscoplasticity model, and that this will make the workload much more variable from one round to the next. We may resume our study of LULESH when this version becomes available.

TCE

ETI has adapted the Tensor Contraction Engine (TCE) provided by PNNL in order to make

it more suitable for exploring our research questions related to data placement, data movement, memory access models and scheduling.

In Q5, we produced a version of TCE which operates as a standalone package, complete with input and reference output data. This standalone package encompasses the code generation, compilation and execution steps of TCE, and is suitable for testing the accuracy of task-based adaptations of the code, and measuring the resulting performance. In Q6, we did additional work to make this standalone program compatible with larger data sets and data files created by NWChem, and we have made this code (together with several reference data sets) available to the other team members.

In Q7, we added the ability to generate code for the OCR and SWARM task-based runtimes, first on a function-by-function basis, and later block by block. The tensor data is in block-sparse format, which provides natural boundaries between blocks of data and will allow us to break parallel block operations into separate tasks.

In Q8, we will study the performance characteristics of this application, and find effective ways to balance and manage the workload on large distributed systems.

SCALE (Intermediate Representation)

A critical component of SWARM's programming model is object-oriented programming. For example, SWARM can then use run-time reflection, polymorphism, and type checking to enable an abstract programming layer for automatic data movement across compute nodes in a cluster. However, when written directly in C, object-oriented programming for SWARM is verbose and complex to comprehend, even when generating code automatically as an intermediate representation. For this reason, in this year we are carefully extending SCALE syntax to more completely support basic object-oriented programming as a well integrated extension of C. We are avoiding complexities of similar constructs from C++ that would impede SCALE language comprehension, language development, and development of the compiler and other language-based tools.

For the Task 7.1 deliverable at the end of Q8, we will produce a report that defines the full set of SCALE language constructs that we have developed over years 1 and 2 as an intermediate representation for expressing codelets, scheduling, data movement, synchronization, and type checking.

Reservoir Accomplishments

Reservoir worked on several aspects of the DynAX project during the first three quarters of the Y2 period. The efficiency of Exascale systems being strongly conditioned by the efficiency of their communications, we have focused our optimization efforts on the communication layer. In our effort to optimize unstructured computations, we worked with PNNL, ETI and the University of Delaware on adaptive sparse data structure transfer optimization techniques.

Tuned SWARM

Highly scalable synchronization mechanism

The Reservoir team identified scalability issues that appear in the synchronization mechanisms currently available when programming to the SWARM, CnC and OCR runtimes. We proposed a modification to counted dependences, available in SWARM and implemented in OCR since version 0.8, which does not only simplify programming to the Exascale runtimes, but has also optimal scalability in terms of the sequential overhead of task creation, the amount of tasks scheduled before they have a chance of being ready ("in-flight tasks"), and the space used by the synchronization objects and their garbage collection. A paper describing the importance of this improved synchronization mechanism, called "autodec", has been submitted for publication[1]. In this work, we greatly benefited from being able to discuss with the ETI team and their experience with large-scale, task-based synchronization mechanisms.

Reservoir developed a backend to R-Stream and runtime layer which support autodecs based on the SWARM counted dependences, as well as a compiler optimization that achieves the optimal overheads. In particular, a task is initialized only when one of its predecessors completes, and is scheduled only when all its predecessors have completed.

Virtual DMA

The Reservoir team developed a communication runtime library for x86 platforms that leverages x86 vector cache-bypassing load and store instructions to implement direct data transfers into and out of the cache. This is an extension of Reservoir's "virtual scratchpad" optimization technique, in which the working dataset of a codelet is first copied into a set of memory regions that fit in the cache, accessed with little or no cache misses by the codelet, and written back to its original location afterwards. The API of this runtime library is that of a typical two-dimensional Direct Memory Access (DMA) engine (strides being allowed on both source and destination ends), hence we called it a "virtual DMA" library.

We modified R-Stream to produce SWARM code using the virtual scratchpad optimization combined with the virtual DMA library (for short we call this technique "virtual scratchpad optimization.") A side effect of accessing data whose range fits in a cache, the number of memory pages accessed during a codelet execution is small, resulting in little or no TLB misses during the execution of a codelet.

Another extremely interesting result of the virtual DMA technique. If:

[1] B. Meister, M. Baskaran, T. Henretty, R. Lethin, "An Exascale-Ready Event-Driven Runtime Synchronization Construct." Submitted to the International Conference on Supercomputing, ICS'2014.

- reuses to a certain data are correctly translated into reuses of data in the virtual scratchpad,
- the set of live-in and live-out data are defined exactly, and
- up to intra-codelet reuse,

then the number of transfers between DRAM and the virtual scratchpad (i.e., the cache) during the execution of a codelet is minimal. As such, the virtual DMA technique applied to a cache-based system is communication-avoiding. However, other communication-related aspects, such as network contention and load-store queues are not taken into account by this technique as of now, except by the sheer reduction in the number of transfers.

Preliminary experiments using this technique are extremely encouraging. Virtual DMAs open up a lot of interesting research and experimentation capabilities, which we plan to investigate further in the next quarters.

Unstructured Mesh computations

We have started our effort to parallelize codes for unstructured meshes by studying a subset of them, called “Structured Adaptive Mesh Refinement” (SAMR). In particular, we are looking at the Chombo Framework from Lawrence Berkeley National Labs, and the underlying BoxLib infrastructure as the reference SAMR implementation. Our initial goal is to learn about the algorithms, parallelization strategies and data-structures involved in Chombo and BoxLib.

We also looked at the approach that uses Legion as a task graph programming API.
The main source of unstructuredness in Chombo is the data-dependent position and size of the hierarchy of boxes in which computations occur.

Forward plan

Our overall plan for the next quarters is to work on leveraging the combination of asynchronous task graph execution model and smart communication layers to support the parallelization of mesh applications, *and* the better mapping of dense codes. We are making good use of the collaborative environment provided by the DynAX project to achieve these goals.

- Q8-Q12
    - Unstructured mesh computations
        - Started, w/ Chombo study.
        - Considering data-centric approach

- Q8-Q10
    - In support of Unstructured mesh computations
        - Data/communication layer (with DynAX team)
        - Keep improving aspects of R-Stream as needed

UIUC Accomplishments

During the last year, the research team at UIUC focused on various enhancements to the Parallel Intermediate Language (PIL) which can translate parallel programs expressed in task graphs into ETI SCALE. The enhancements include tiled array data representation in Hierarchically Tiled Array (HTA), Structured PIL (SPIL) extension and the design of SPMD PIL. This year, we advanced our work by implementing benchmark programs in HTA notation for performance evaluation and overhead analysis, extending HTA library to support manipulation of irregular tiles, and implementing HTA for the distributed memory environment.

NAS Parallel Benchmark Implementation with PIL HTA

In Q5, the UIUC team implemented six of the NAS Parallel Benchmarks (NPB) using the Hierarchical Tiled Array (HTA) library which is in turn written in terms of Parallel Intermediate Language (PIL) primitives. The benchmarks can now be executed on OpenMP and ETI's Shared Memory SCALE. The performance results were retrieved from a workstation with many cores.

The following table shows the performance comparison between the fine-tuned handwritten OpenMP-C NPB implementation and our preliminary HTA implementation targeting SCALE (PIL2SCALE). We conducted the experiments on a multi-core shared memory machine with Intel Xeon E7-4860 CPU using up to 64 cores. The results show that the execution time of the SCALE version decreases with the number of cores used. However, the yet-to-be-identified overhead caused the slowdown in execution time compared with the OpenMP implementation, varying from 68% -- 14%. We are working on further identifying the sources of the overhead. Possible sources of overhead includes: (1) Algorithm difference, (2) HTA library overhead, (3) PIL compiler generated code and (4) SWARM runtime inefficiency for certain usage patterns.

Benchmark	OpenMP Exec. time (s)	PIL2SCALE Exec. time (s)	Speedup
EP (CLASS C)	8.98	14.81	60.61%
IS (CLASS C)	1.31	2.42	54.30%

FT (CLASS C)	13.49	19.61	68.77%
MG (CLASS C)	7.68	13.58	56.57%
CG (CLASS C)	15.22	105.04	14.49%
LU (CLASS C)	42.73	246.08	17.36%

The implementation of NPB in HTA running on SWARM runtime is a proof of concept that demonstrates the feasibility of executing non-trivial HTA programs on codelet runtime systems. Although the results in this report are very preliminary, they show that the performance improves with the number of threads used. We expect that as we start working on identifying and removing the overhead and optimizing the HTA library there will be significant improvements in performance. The Illinois team will work closely with ETI on improving the HTA implementation and PIL's code generation scheme.

Overhead Analysis with Mini-benchmark

As a first step in clarifying the cause of the overhead, we collaborated with the ETI team and analyzed the performance results of a mini-benchmark. The mini-benchmark simulates the behavior of the NAS parallel benchmarks but with a relatively simple algorithm for easier analysis of the overhead in parallel operation invocation. The pseudo-code of the mini-benchmark program is shown in the following code snippet:

```
01  #define EXP (8)                 // for grain size control
02  #define ITER (100000)           // the number of parallel operation invocations
02  void power(HTA *r, HTA* h) {
03      double* data1 = r->leaf.raw, data2 = h->leaf.raw;
04      int num_elem = Tuple_product(&r->flat_size);
05      for(int i = 0; i < num_elem; i++) {
06          double x = data2[i];
07          POW(x, EXP);
08          data1[i] = x;
09      }
10  }
11  int hta_main() {
12      HTA *h = HTA_create(...);
13      HTA *r = HTA_create(...);
14      HTA_map(init, h);
15      for(int i = 0; i < ITER; i++) // Each iteration invokes a parallel operation
```

```
16          HTA_map(power, r, h);     // r = pow(h, exp)
17  }
```

We found the overhead due to the sequential spawning of parallel codelets which requires argument boxing. By using swarm_Locale_scheduleToLeaves() which assigns work for each of the available worker thread without argument boxing, the overhead is reduced significantly. We will keep exploring the chances in reducing the overhead in different levels and our goal is to get the performance HTA-to-SCALE version close to that of OpenMP on shared memory machines.

HTA-to-PIL Interface Design Changes for Supporting SPMD

In Q6, we designed a strategy for expanding HTA programs to create a SPMD version which can then be efficiently mapped onto distributed memory SCALE. HTA programs consist of a single thread with invocations to data parallel operations. In SPMD mode, the sequential parts are replicated. This is done automatically by the PIL compiler which generates SCALE's codelets to let the underlying SWARM runtime system automatically discover and exploit the parallelism hidden, schedule execution dynamically, and move data according to the runtime information or compiler hints. We finished the HTA-to-PIL interface design, and have continued the work in Q7 and Q8 on the SPMD design. We expect to have a working implementation of the SPMD version of HTA at the end of Y2.

Support for Irregular Tiles

In Q7, the UIUC team extended the HTA library to support irregular tiles partitioning. By having the partitioning mechanism, it is now possible for applications to create irregular tiles and thus providing more flexibility in controlling the granularity of parallel computation tasks. The API functions HTA_part() and HTA_rmpart() are added into the HTA library implementation, and we plan to investigate whether any of the NAS benchmark programs can utilize this feature to improve performance.

PNNL Accomplishments

This year PNNL worked on two application codes: 1) the Coupled Cluster Method (CCM) in NWChem, and 2) LULESH. For CCM, we provided sets of tensor equations, a driver program, and input deck that were used to develop and evaluate an automatic Codelet code generator. For LULESH, after developing a CnC diagram of the program's control and data flow, we ported the code from C to C++ to prepare it for execution on the

simulator. We then transformed the refactored code into a CnC program using Intel's CnC framework and Rice's CnCOCR framework. In runtime systems, we continued evaluating the effects of compression algorithms for primitive rescinded data types on dense data structures. We created two methodologies for very fine-grained execution models: 1) Architected Composite Data Types (ACDT), and 2) Group Locality.

We created the ACDT conceptual framework to exploit opportunities that arise when the runtime is aware of a composite's properties (access patterns, data composition, dynamic range, etc. We evaluated the performance of the framework on two representative processing kernels: 1) Matrix Vector Multiply and 2) Cholesky Decomposition applied to varying degrees of matrix sparsity. For Group Locality, we developed a highly parallel tiling strategy with intratile parallelism, a framework for fine grained parallelism where threads perform under a micro dataflow execution model, and methods for data movement and restructuring of data such that accesses with reuse are preserved in a state that matches the pattern of future access.

Further Details

Further details on the above Accomplishments can be found in the Q5, Q6 and Q7 reports: Brandywine X-Stack Report Q5, Brandywine X-Stack Report Q6 and Brandywine X-Stack Report Q7.

Collaborative Research Accomplishments

The following works were accomplished during this period in close collaboration between Reservoir, PNNL, ETI and University of Delaware:

- A framework for adaptive optimizations based on exascale task schedulers[2]. The framework has a clean separation of concerns between monitoring the runtime environment (which is done by the runtime scheduler) and the subsequent use of this information by dynamic optimizations at any point in the Exascale software stack (compiler-generated, libraries, OS, runtime itself) and possibly hardware. In this task-centric adaptation framework, meta-information is associated with tasks either by the compiler (which is particularly natural to R-Stream since it defines and generates the tasks) or a programmer. Validation was performed using an adaptive sparse compression engine, whose behavior reacts to the run-time communication-to-computation ratio.
- A runtime-based framework to support high-level runtime composite data types,

[2] T. St John, B. Meister, A. Marquez, J. Manzano, G. Gao, X. Li, "ASAFESSS: A Scheduler-driven Adaptive Framework for Extreme Scale Software Stacks." In proceedings of The 4th International Workshop on Adaptive Self-tuning Computing Systems, ADAPT '14, Vienna, Austria, January 2014. Recipient of the best paper award.

validated with runtime sparse compression optimization in massively multithreaded linear algebra computations[3]. A runtime discriminator matches the runtime features of composite data with features of the architecture to define runtime type qualifiers. Runtime type qualifiers can be used by all levels of the Exascale software stack to adapt their behavior with respect to the data. This data-centric adaptation framework is complementary to the task-centric one presented above.

Reservoir has also been supporting the UIUC team's effort to generate code that can be mapped through R-Stream. This is work in progress and hasn't resulted in any publications yet.

UIUC and ETI are working together on tuning the SCALE output from the PIL and HTA tools, toward the goal of demonstrating high-performance, scalable codelet-based implementations of the NAS parallel benchmarks.

The collaboration work between the different team members have generated a number of joint publications, the details of which can be seen in the Publication section below.

Technologies Delivered

Technologies that are delivered can be found on the DynAX page of the X-Stack wiki: https://www.xstackwiki.com/index.php/DynAX#Deliverables

Technologies Delivered (Q5-Q7):

- NWChem TCE module:
  - Standalone application (ETI-Q5/Q6)
  - Function-level parallel OCR version (ETI-Q6)
  - Function-level parallel SWARM version (ETI-Q7)
- Tuned R-Stream SWARM backend - autodecs (Reservoir Q5/Q6)
- Tuned R-Stream SWARM backend - virtual DMA, preliminary (Reservoir Q6)
- NAS Parallel Benchmark Implementation with PIL HTA (for Shared Memory SCALE) (UIUC-Q5)
- HTA-to-PIL-to-SCALE interface design changes for supporting SPMD (UIUC-Q6)
- Support for irregular tiles (UIUC-Q7)
- Coupled Cluster equation, driver program, and input sets (PNNL-Q5)
- C++ version of CnC based LULESH application code (PNNL-Q6)

Technologies anticipated to be delivered (Q8):

- NWChem TCE module:
  - Block-level parallel OCR and SWARM versions (ETI-Q7-Q8)
- Tuned SWARM backend - virtual DMA, advanced (Reservoir-Q8)

[3] A. Marquez, J. Manzano, S. Song, B. Meister, S. Shresta, T. St John, G. Gao, "ACDT: Architected Composite Data Types - Trading in Unfettered Data Access for Improved Execution." Submitted to the the IEEE Cluster 2014 Conference, Madrid, Spain.

- Preliminary research on leveraging communication layer and asynchronous task graph execution model to target mesh computations (Reservoir Q8).
- PIL HTA implementation for SPMD (UIUC-Q7-Q8)
- NAS Parallel Benchmark support for Distributed Memory SCALE (UIUC-Q8)

Technologies in progress in Year 2 to be completed in Year 3:

- Performance evaluation of the HTA library on both shared memory and distributed machines using the NAS Parallel Benchmarks (UIUC)
- Evaluation of miniMGM (PNNL)
- Initial studies about the effects of additional ACDTs (which focus on resiliency types) on selected architectures under massively multithreaded runtime systems such as SWARM (PNNL)
- Performance and power studies for new enhancements to the initial discriminator framework (PNNL)

Presentations

- Extreme scale technical review on TCE as a proxy app in December 2013 (audience includes Traleika Glacier project teams and a broader audience).
- Extreme scale technical review on the adaptive task-centric runtime framework in February 2014.
- Presentation on distributed scheduling challenges in Cholesky to OCR core team in March 2014.
- Presentation made on Virtual DMA optimization to Extreme scale technical review audience in April 2014.

Publications

A. Marquez, J. Manzano, S. Song, B. Meister, S. Shrestha, T. St John, G. Gao, “ACDT: Architected Composite Data Types. Trading in Unfettered Data Access for Improved Execution,” submitted to the IEEE Cluster 2014 Conference, Madrid, Spain.

B. Meister, M. Baskaran, T. Henretty, R. Lethin, “An Exascale-Ready Event-Driven Runtime Synchronization Construct,” submitted to the International Conference on Supercomputing, ICS’2014.

S. Shrestha, G. Gao, et.al, “Gregarious Tiling: A Novel Methodology for Collaborative Multithreaded Execution”, submitted to 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming in Florida, USA.

B. Meister, N. Vasilache, M. Baskaran, T. Henretty, R. Lethin, “R-Stream experiments with hierarchical iteration tiling”, 16th SIAM Conference on Parallel Processing for Scientific Computing, February 2014, Portland, Oregon, USA.

N. Vasilache, M. Baskaran, T. Henretty, B. Meister, R. Lethin, “A tale of three runtimes,” submitted to 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming in Florida, USA.

T. St John, B. Meister, A. Marquez, J. Manzano, G. Gao, X. Li, “ASAFESSS: A Scheduler-driven Adaptive Framework for Extreme Scale Software Stacks.” In proceedings of The 4th International Workshop on Adaptive Self-tuning Computing Systems, ADAPT ‘14, Vienna, Austria, January 2014. Recipient of the best paper award.

- We are working on publications for the Cholesky, SCF and TCE work.

Year 3

- ETI
  - Applications support reliability via containment domains
  - Migration paths for MPI+OpenMP applications
- Reservoir
  - Unstructured mesh computations (Q8-Q12)
    - Started, w/ Chombo study.
    - Considering data-centric approach
  - In support of Unstructured mesh computations (Q8-Q10)
    - Data/communication layer (with DynAX team)
    - Keep improving aspects of R-Stream as needed
- UIUC
  - Extend HTA design and implementation to include multiple levels of parallelism, irregular parallelism
  - Implement a variety of other benchmarks
    - Mini GMG, AMR, etc
  - Evaluate and tune for performance
  - Evaluate programmability using objective metrics (e.g. number of operations)
- PNNL
  - Potential application of ACDT to OCR.
  - Test different memory mappings on Tilera to orchestrate data movement and avoid memory interference in memory banks and pages.

Websites

The DynAX group website for meetings, project overviews, and deliverables is located at: http://xstackwiki.com.