Milestone 10 Status Report

Award #: **DE-SC0008717**
Recipient: **Intel Federal LLC**
Project Title: **TRALEIKA GLACIER X-STACK**
PI: **Shekhar Borkar**
Report Date: March 2, 2015
Period Covered by Report: **December 1, 2014 to February 28, 2015**

Acknowledgment: This material is based upon work supported by the Department of Energy [Office of Science] under Award Number DE-SC0008717.

Contents

Executive Summary

The team's focus is to mature SW stack and evaluate it with applications on the Traleika Glacier (TG) architecture. We enhanced our collaboration with the application developers to support evolutionary applications (traditional C, MPI, etc.) on the OCR platform, continued development of the tools and simulators for the third version of the architecture, and strive to increase scalability, performance, and quality. PNNL's internal OCR version has become an extremely useful test-bed to research novel ideas with applications since it provides a very competitive environment when compared to other fine grain runtime systems and models.

A preliminary implementation of the unified CnC front end unifies the CnC language across different runtime implementation, such as CnC-HC, CnC-OCR, Intel CnC-C++. This allows the user to run the same application on multiple runtime targets without any changes, hence enhancing user productivity. Reservoir has extended R-Stream optimizations to larger sections of performance-critical kernels in the HPGMG benchmark demonstrating performance and productivity benefits by automatic generation of efficient OCR code for HPGMG kernels. And they are on track to make it capable of scalable automatic on-the-fly creation and management of data-blocks—ready for the upcoming applications workshop.

To demonstrate interoperability, we conducted two experiments using CnC for coarse grain parallelism, with HTAs inside the CnC steps for inner parallelism and locality, targeting Cholesky factorization, and a matrix inversion. Initial results show that if offers slight performance improvement over CnC alone, with slight impact to programmability.

Significant progress to report in the refactorization of proxy applications for extreme-scale architectures. We now have several implementations of CoMD that run with OCR, including two versions refactored directly to the OCR API, and a version that uses CnC to produce OCR code. These are demonstrated on both, the x86 implementation of OCR for larger scalability, and the TG simulator for architectural explorations. Another application, namely HPGMG, too, is ported on OCR and tested using the x86-pthreads target.

Applications Workshop #4 is planned to be held in Hillsboro, OR April 7-8, to test the SW stack with "hands-on" experience with our DOE colleagues on the proxy applications.

1	11/30/12	Architecture V2 spec & preliminary apps kernal identified for evaluation	Intel
2	3/1/13	Simulators V2 functional, tools (C + binutils) in place, IRR V1 identified	ETI, Reservoir
3	5/31/13	Selected kernels evaluated for 0(compute)	Intel, PNNL
4	8/30/13	Basic timing in simulator, intelligent scheduling in Exec model, tools (LLVM, etc)	ETI, Rice, Reservoir
5	11/27/13	Selected kernels evaluated for 0(com), select apps coded with PGM system for IRR	UCSD, PNNL
6	2/28/14	Architecture V2.5 spec, system evaluation of V2.0	Intel, UIUC (Josep)
7	5/30/14	Simulators V2.5 functional, tools for V2.5 released	ETI, Reservoir
8	8/29/14	System evaluation of V2.5	UIUC (Josep)
9	11/26/14	Arch V3.0 spec (ISA 4.1.0), selected apps evaluation with execution model and programming system for V2.5	Intel (with all)
10	2/27/15	Simulators V3.0 functional, tools for V3.0 released	Intel, Reservoir
11	5/29/15	Release OCR (Open Collaboration Runtime) V1.0	Rice
12	8/28/15	Evaluation of all X-Stack technologies and report	Intel

Intel

FSIM and OCR (Romain Cledat)

Introduction

During this quarter, we focused on three main areas: **(a)** increased collaboration with application developers to support evolutionary applications (traditional C, MPI, etc.) on the OCR platform, **(b)** continued development of the tools and simulators for v3 of the architecture, and **(c)** increasing the quality of OCR particularly in the area of performance.

Accomplishments

We have increased collaboration with application developers (see specific section on "Applications") and held two internal face-to-face meetings to determine how to support evolutionary applications on OCR. This effort is motivated by DoE's ask to support today's models in our OCR environment. We are specifically focusing on supporting a traditional C program by porting a libc implementation called *newlib* as well as supporting unmodified MPI code by implementing a lightweight wrapper around OCR to emulate MPI calls. These features are being supported by modifications to the OCR codebase.

We are also working with application developers to develop visualization tools to help in the understanding and debugging of applications written directly to the OCR platform. We currently have a timeline visualizer and initial versions of visualizers that allow the inspection of **(a)** the state of memory, **(b)** the data traffic OCR generates as well as **(c)** the task flow-graph of the application.

We also continued development of the tool-chain and simulators for v3 of the architecture. Unfortunately, we were not yet able to port OCR to it as the environment is not yet stable. The current focus is on revamping the simulator for the control engines (CE), validating the execution engines (XE) using RTL and implementing missing features in the simulator (queue engines, memory engines, power estimates, etc.)

OCR internals have also been improved. Significant features include a new allocator that is better able to handle multiple "levels" of memory, a scheduler infrastructure that will enable us to further explore various scheduling and data placement heuristics. We are also in the process of fixing some of the major bottlenecks identified till date.

As we prepare for a June release of OCR, several automatic regression suites are also in alpha version to allow us to track the evolution of the runtime. We are specifically looking at identifying the evolution of runtime overheads and scaling. The OCR specification is also being finalized with work on the memory model currently at the forefront.

Plans

For the next milestone, we plan to:

- Port OCR to v3 of the architecture
- Continue internal OCR improvements

- Host the 4th Application Workshop in April (pushed back from end of February due to DoE conflicts).

Issues

None.

Inventions

Several inventions were disclosed with this milestone that will be reported under separate cover.

Publications

None.

Applications (Bill Feiereisen)

Introduction

During the period December 15, 2014 - March 15, 2015 we made significant progress in the refactorization of proxy applications and "teaching" kernels into the revolutionary programming models that are based upon the OCR dynamic runtime. We also built upon the unified build structure of the application and runtime repository that we set up last quarter, by adding a formal regression test system. This system allows for nightly tests of all the proxies and kernels upon the runtime. This will assist in the evaluation of the initial public release of the system V1.0 that is planned for this coming summer. It has already proven its worth in runtime debugging and necessary subtle changes in applications refactoring.

Accomplishments

Proxies and Kernels**:** We now have several implementations of CoMD that run upon OCR, including two versions refactored directly to the OCR API and a version that uses CnC to produce OCR code. These versions run upon both the x86 implementation of OCR and upon the simulator for the Traleika Glacier architecture. Work is underway to understand scaling issues related to the memory hierarchy representation in the simulator.

Work upon HPGMG proceeds on several fronts. We have a refactored version programmed directly to the OCR API and in addition our colleagues at Reservoir Labs and UCSD have used the R Stream compiler to optimize a version for OCR. A CnC version to OCR is in work in concert with our colleagues at PNNL and is expected during the next quarter. Inspired by the visualization tool we have now generated a timeline visualization of HPGMG in order to understand the individual running tasks. HPGMG is the subject of investigation also for the implementation of some communication-avoiding ideas.

Application Repository: This repository structure now allows us to run regular app regression tests upon the Traleika Glacier software and hardware infrastructure, providing a powerful tool for co-development of the architecture, the simulator and the OCR runtime. This regression framework is in operation and we are adding applications regularly. Our goal is to provide this repository fully populated to the DOE for assessment and for use as teaching examples for refactoring codes in the high level programming notations.

Applications Workshop #4 will be held in Hillsboro, OR April 7-8. This workshop is planned to be "hands-on" and mostly devoted to joint coding with our DOE colleagues on the proxy applications. We invite our DOE colleagues to "bring their own codes." Preparations are ready and we remind our DOE colleagues to please consider attending. If you should hear about this workshop through this report and and want to attend, but are not already on our invitation list please contact us immediately.

Plans

During Q2-2015 we will continue to populate the applications repository with the initial versions of all of the proxy applications, including the original versions for comparison. And we will expand the implementations of the proxies beyond the current OCR and CnC→ OCR versions.

Issues

None.

Inventions

None.

Publications

None.

Community Development and Coordination (Wilf Pinfold)

Introduction

We reached agreement between the four large X-Stack programs DEGAS, XPRESS, D-TEC and the smaller DOE X-Stack projects to evaluate developments and collaborate on common candidates for the revolutionary programming system. In particular, Intel plans to leverage the programming model ideas from an evaluation and selection study we are collaboratively conducting. Intel also aims to explore common learnings from research into distributed dynamics runtimes and work toward a common Exascale runtime interface "EOCR.".

Accomplishments

Programming Environments Whitepaper: Completed and distributed to Programming System Workshop attendees for event starting March 9th.

Runtime System Report: All sections filled but considerable editing still to do.

PI meetings: Completed discussions on hardware changes for Exascale and built content into programming systems and runtime documents.

Plans

- Programming Environment and Runtime Systems Workshops: March 9-13, 2015

Issues

None.

Inventions
None.

Publications
None.

Reservoir Labs - Richard Lethin

Introduction
This research memo describes the contributions of the Reservoir Labs X-Stack team during the period of December 15, 2014 through March 15, 2015. A summary of our contributions during this period includes:

- Extending R-Stream optimizations to larger sections of performance-critical kernels in the HPGMG benchmark to demonstrate performance and productivity benefits offered by R-Stream through automatic generation of efficient OCR code for HPGMG kernels
- Making R-Stream capability for scalable automatic on-the-fly creation and management of datablocks robust (It will be made available during Intel's Exascale Applications Workshop in April)
- Supporting users using R-Stream installed in the Intel X-Stack cluster

Accomplishments
This section details our contributions during this reporting period.

Scalable Datablocks Support in R-Stream
We continued our work on improving R-Stream's capability to automatically generate scalable OCR code, especially on improving scalable OCR datablocks support in R-Stream. Previously, R-Stream used to create one large datablock for each array and then create smaller datablocks within EDTs that fit in the local scratchpad (for e.g. XE scratchpad in TG architecture) based on the data accessed within an EDT. The new capability takes in a data partitioning (aka data tiling) specification from the user and creates datablocks according to the specification. R-Stream automatically figures out the datablocks that each EDT needs and creates an input slot for each datablock. Within an EDT, the data needed by the EDT from each of its input datablocks is automatically copied on to temporary local arrays that collectively fit in the local scratchpad attached to the processing element. This capability eliminates the need to create one large datablock for each array and provides a pathway to achieve scalable performance.

R-Stream supports automatic generation of OCR code with on-the-fly scalable creation of EDTs. To compliment this capability, it now supports on-the-fly creation of datablocks. Creating all the datablocks at the beginning of the execution is non-scalable and adds a huge "startup" overhead (similarly, it incurs a huge startup overhead when all EDTs are created at the beginning of the execution). A datablock is not created until it is needed for the first time by an EDT that is ready to execute and uses the datablock for its computation. R-Stream automatically figures out the dependence between different EDTs and the dependence between EDTs and datablocks, and automatically generates code for optimal on-the-fly EDT

and datablock creation. R-Stream has a light-weight runtime layer that operates on top of OCR and handles these capabilities, namely, on- the-fly creation of EDTs and datablocks.

HPGMG Mapping

R-Stream Optimization

In the previous quarters, we identified HPGMG smoothers as main bottleneck areas, and mapped the GSRB and Chebyshev smoothers to enable automatic parallelization and optimization, and OCR code generation by the R-Stream compiler. As mentioned in the previous quarterly report, we mapped the finest grain parallel region and produced OCR code for it. Initial results showed that OCR performance is better than or comparable to the OpenMP performance.

During this quarter, we worked on optimizing coarser regions of the smoother kernels to exploit the opportunities in executing these regions in a more asynchronous fashion in an EDT-based runtime such as OCR. The smoothers are good candidates for optimized execution in an EDT-based runtime. However it is error-prone and extremely difficult and challenging to hand-code them in OCR or other EDT-based runtime models. R-Stream seems to be the right fit to provide performance and productivity benefits in optimizing these codes, through its automatic OCR code generation and optimizations to achieve scalable performance. Specifically, these smoother codes offer the following opportunities – 1) on-the-fly scalable creation of EDTs performing the smoother computations (as opposed to non-scalable creation of all EDTs at the beginning of the execution) and 2) on-the-fly scalable creation and management of datablocks needed by the EDTs (as opposed to non-scalable creation of all datablocks at the beginning of the execution). R-Stream exploits both these opportunities, i.e. on-the-fly EDT creation and datablock creation and management, and generates an OCR code that is scalable and efficient.

CnC Collaboration

Reservoir Labs has been collaborating with the CnC team at Intel/Rice and the PNNL team to define software interfaces between CnC steps and R-Stream optimized components of HPGMG including smoothers, restriction, residual computation, and interpolation operations.

SDSC Collaboration

We have been meeting with Laura Carrington's group at the San Diego Supercomputing Center to discuss our individual HPGMG progress and plan of action to plug-in R-Stream optimized components in their hand-written OCR code.

Intel Collaboration

Reservoir Labs has been providing guidance to Gabriele Jost from Intel on HPGMG optimizations (e.g. communication-avoidance optimizations) and usage of R-Stream for simple kernels and HPGMG.

Plans

For the next milestone, we currently have the following plan of action:

- Extend the scope of R-Stream optimizations for HPGMG, specifically, enable broader cross kernel fusion and map more coarse-grained parallel regions in different levels of the multigrid solve.
- Enhance performance-scalable optimizations in R-Stream-OCR code generation.
- Demonstrate R-Stream technology to TG project members and DOE users at Intel's Exascale Applications Workshop at Hillsboro in April (Note that R-Stream is made available to all interested users – TG members and DOE users – through an installation in the Intel X-Stack cluster).
- Demonstrate the performance and productivity benefits of R-Stream through automatic OCR code generation for HPGMG benchmark, at Intel's Exascale Applications Workshop at Hillsboro in April (Note that the R-Stream optimized HPGMG code is available in the Intel X-Stack git repository).

Issues
None.

Inventions
None.

Publications
None.

Conclusion
During this quarter, we extended the capability of R-Stream to automatically generate efficient OCR code for larger performance critical sections in the HPGMG benchmark. R-Stream takes few lines of (HPGMG) C code as input and produces efficient OCR code that gives comparable or better performance than OpenMP; this highlights the performance and productivity benefits that R-Stream offers to the exascale software stack. R-Stream generated OCR code for HPGMG kernels is made available to all TG project members through the Intel X-stack git repository. Further, we made progress in extending the high-level compiler optimizations for TG through R-Stream. Specifically, we made the capability in R-Stream for scalable automatic on-the-fly creation and management of datablocks robust. The new R-Stream capabilities will be made available during Intel's Exascale Applications Workshop in April.

Rice University - Vivek Sarkar

Accomplishments
- We have made further performance improvements and committed several bug fixes to the OCR implementations
- We have resolved memory issues in CnC-OCR-FSIM and are now have an implementation of CoMD in CnC running on CnC-OCR-FSIM
- We have an initial implementation of the Unified CnC front end implementation that unifies the CnC language across different CnC runtime implementation (CnC-HC, CnC-OCR, Intel CnC-C++ e.t.c.). This allows the user to run the same application on multiple runtime targets without any changes. The process of integrating this code into the XStack repositories is under way.

- We have written a draft paper on Tuning for CnC with automated generation of tuning annotations using a Unified CnC translator. Target for submission is Supercomputing 2015
- (with UIUC) Working on a prototype to demonstrate interoperability between CnC and HTA (Hierarchical Tiled Arrays). Code within CnC steps can invoke operations implemented in HTA. CnC manages the coordination among coarse-grain asynchronous task-based computation steps. HTA is responsible for optimizing the fine-grain data parallel computation within each coarse-grain step. The initial working example, Cholesky, the coarser-grain CnC invoke a finer-grain data parallel matrix-vector multiplication operation implemented in HTA. We will look at a more serious application and begin performance studies.
- (With PNNL and LBL) Communication-avoiding algorithms in CnC. Use the separation of concerns between domain specification and tuning to make the process of avoiding communication on different targets more productive. CnC implementation of the smoother phase of HPGMG is under way. This phase is the key for the communication avoiding optimizations in HPGMG.

Plans

- Build a prototype of a system that uses Rice HPC toolkit to collect information at the CnC level. This will be used to optimize CnC programs. We'll show the automatically collected data and a by-hand transformation based on the collected data.
- (in collaboration with Purdue) Automatic transformation of CnC program to improve performance/communication/power consumption using CnC hierarchy.

Issues

None.

Inventions

None.

Publications

None.

UCSD - Laura Carrington

Proposed milestone

Work on experiments with CoMD on OCR-FSIM. Complete the port of HPGMG on OCR –FSIM, and work on the performance related issues.

Issues and Limitations Encountered

Some issues with the CnC version of CoMD have been solved and we prepared he code for release in git. Some issues with FSIM prevented scaling to realistic CoMD problems; new versions of the memory hierarchy simulations were recently made available but we haven't successfully scaled the problem size (cause still to be investigated).

Progress

We were able to run some experiments with CoMD on FSIM and we are now experimenting with the new memory hierarchy modules recently released. We renamed and organized the directories according to the new format proposed for applications that are maintained in the source tree. In addition, now that CoMD/CnC/OCR works we are adding that too to the repository.

We completed the development of HPGMG on OCR and tested the code on x86-pthreads-x86 target. The code is ready to be checked into the repository. We are now investigating its performance differences when compared to the reference implementation.

The plots below show the performance of HPGMG with respect to the reference MPI implementation.

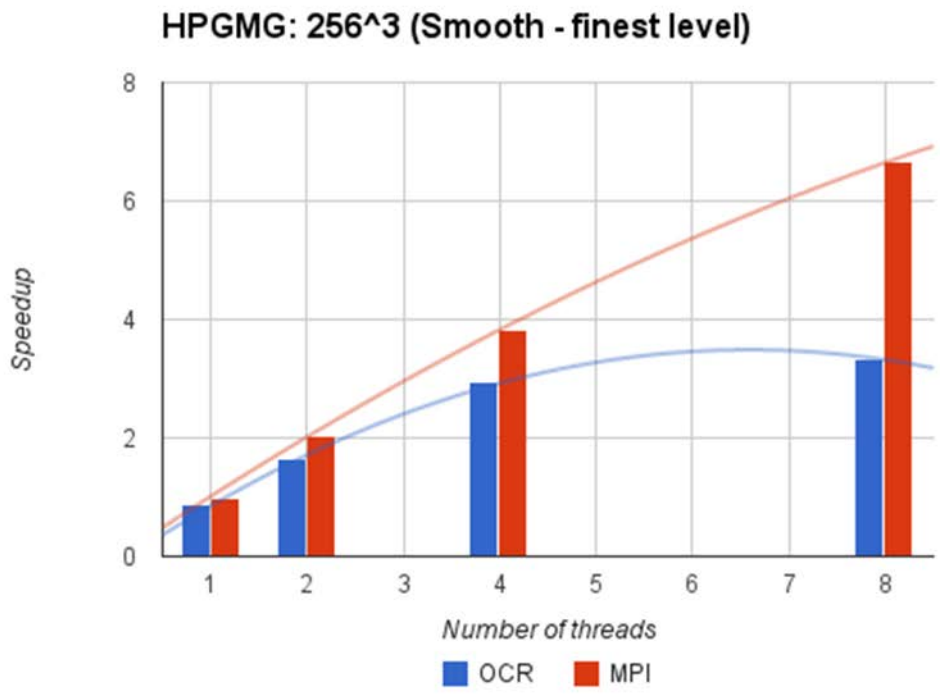

Figure 1

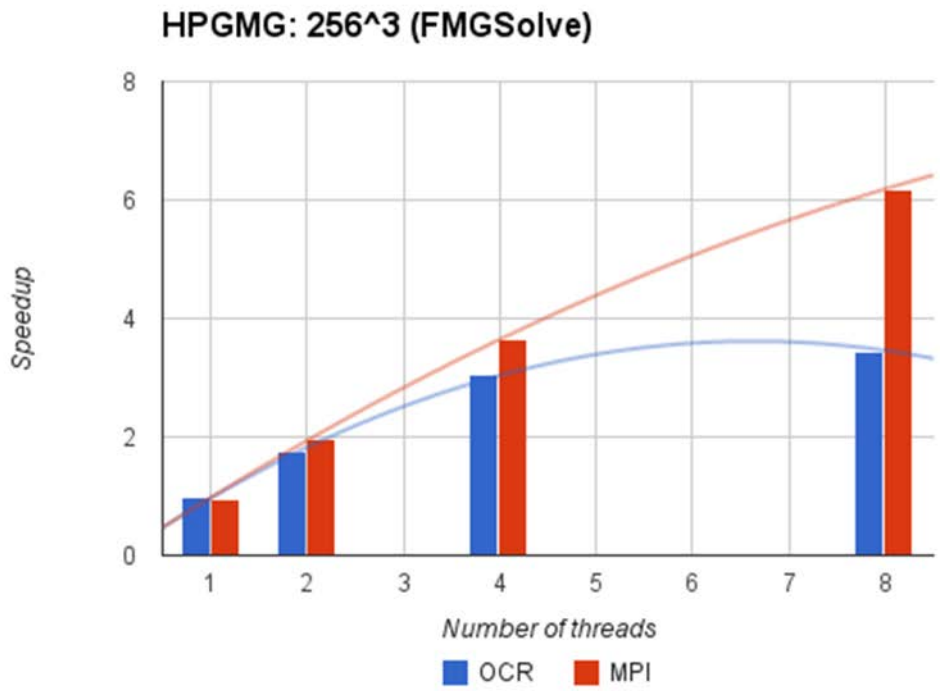

Figure 2

In agreement with the ExMatEx team, we are currently working on a new CoMD/OCR version that will have fewer synchronization points and that will be more scalable. In the previous version we had global synchronization due to the fork/join points delimiting the computation phases (e.g. force computation). In this new version, we maintain parallel "lanes" of EDTs, one per cell, and use data exchanges and dependencies to enforce local synchronizations between EDTs operating on neighboring cells. There are some subtleties that need to be taken care of, such as how to create EDTs and notify EDTs in other lanes about their guid, which greatly complicate the coding (fork/join much easier to write and understand); but overall, we believe that the performance and scalability will improve.

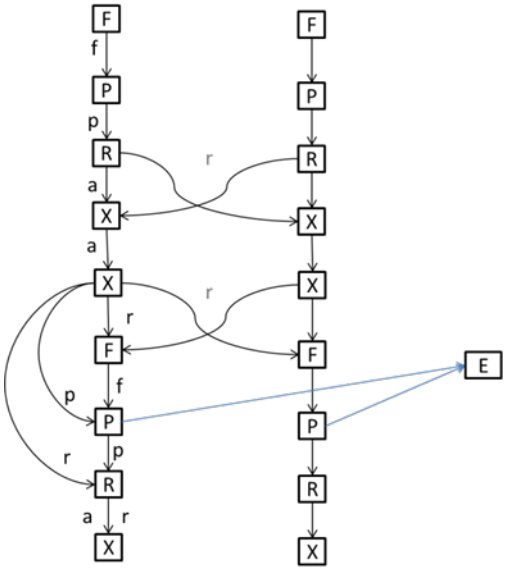

Figure 3: Asynchronous CoMD/OCR. the computatin phases F(force), P(momentum), R(position), X(atoms exchange) are performed in parallel on single cells without global synchronizations.

Next steps

We will continue experimenting with CoMD on FSIM and compare different variants for their performance and energy efficiency. We will complete the CoMD/OCR version described.

We will test our OCR HPGMG code on FSIM target (build and run) and evaluate performance and energy efficiency. We will continue our analysis on performance characteristics of HPGMG code on x86-pthreads-x86 target.

By the end of the next milestone all the code will be in the repository and integrate for regression testing.

University of Illinois Urbana Campus – David Padua

Accomplishments

HTA Implementation Improvements

Last quarter we began the implementation of SPMD parallelism in HTAs. This will allow us to leverage asynchrony in OCR through the removal of global synchronization (barriers) in the HTA library. This quarter we completed the implementation of the SPMD parallelism in the HTA library. In the new asynchronous SPMD implementation, all synchronization is pointtopoint. Additionally, this synchronization only happens *when needed* . Two tasks only synchronize if they *need* to exchange data.

Furthermore, we have adapted the NAS benchmarks to leverage this new execution model in the HTA library. We are currently using these updated benchmarks to evaluate the performance of the library, and add any performance improvements we can. We have observed some high overheads that appear to be coming from the OCR runtime that are discussed further in the Issues section below.

HTA and CnC Integration

We have completed two initial experiments using CnC for coarse grain parallelism, with HTAs inside of the CnC steps for inner parallelism and locality. These experiments were a Cholesky factorization, and a matrix inversion. Initial experiments show that we are able to gain only slight performance improvements over CnC alone, with a slight increase in programmability. These experiments are promising for future improvements.

Plans

Graph Extensions for HTAs Implementation

The implementation of the graph extensions for HTAs is taking longer than expected. We will continue to work on this in Quarter 11.

Graph Extensions for HTAs Evaluation

Once the implementation of the graph extension is complete, we plan to provide an evaluation of the implementation with an Single Source Shortest Path (SSSP) algorithm, as outlined in the Quarter 9 report.

Evaluation of HTA Implementation

We will provide an evaluation of our performance improvements via six of the NAS benchmarks. These benchmarks will be EP, IS, MG, CG, FT, and LU. We will provide an evaluation using our synchronous fork/join implementation of HTAs as well as the new asynchronous SPMD implementation of HTAs that leverages pointtopoint synchronization.

Continuation EDT

We discovered this quarter that the implementation of the continuation EDTs that we need for efficient execution has been delayed beyond the end of this project. Without the continuation EDTs, we will have to rely on the antiquated ocrWait() call that has very high overhead. We will do our best with what we have.

Issues

None.

Inventions

None.

Publications

None.

University of Illinois Urbana Campus - Josep Torrellas

Accomplishments

In this quarter, Wooil Kim and I continued the evaluation of FSIM under OCR 4.0.8. The system is not yet totally debugged for performance. There are a few bottlenecks and inefficiencies that are currently being fixed. In particular:

- The memory allocator is still not performing well. As a result, running an application with more blocks results in slowdowns.
- The inactive XEs are left spinning rather than being power gated. As a result, they continue executing instructions and consuming energy.
- The energy numbers are still not available. We expect these statistics to be available in the next month.

We present here some data from the experiments we performed running Cholesky.

Cholesky Program

Cholesky is a decomposition method of a positive, definite matrix into a product of a lower triangular matrix and its conjugate transpose. The algorithm is iterative, and values produced in the previous iteration are used in the current iteration. The algorithm has O(N^3) complexity because it iterates N (the matrix size in one dimension) times for matrix elements (N x N). The simplified sequential algorithm is written as follows:

```
for (k = 0; k < N; k++) {
  A[k][k] = sqrt(A[k][k]);
   for (j = k+1; j < N; j++) {
      A[j][k] = A[j][k] / A[k][k];
      for (i = k+1; i < N; i++) {
         A[i][j] = A[i][j] - A[i][k] * A[j][k];
      }
   }
}
```

The parallel version of Cholesky divides an entire input matrix into t x t tiles, where t (the number of tiles in one dimension) is N (the matrix size in one dimension) divided by T (the tile size in one dimension). An external program transforms an original matrix into a tiled lower triangular form, and the Cholesky program starts from it. The breaking of the program into different tasks (called EDTs or Event-Driven Tasks) was described in the previous milestone report.

All EDTs are created early on, but they wait until their dependences are resolved. The degree of parallelism in Cholesky is affected by two main factors. The first one is the number of tiles. Given a fixed problem size, more tiles with smaller tile size enable more parallelism. However, increasing the number of tiles also increases the overhead. The second factor is related to the runtime. When there are many EDTs that are ready to execute, the runtime picks one among them and assigns it to an idle XE. If the runtime chooses an EDT with more dependent EDTs, the completion of the EDT increases the number of executable EDTs. Overall, Cholesky shows a highly-variable degree of parallelism.

Performance as We Change the Number of Blocks

Figure 1 shows the execution time (in cycles) of Cholesky with matrix size 500 for different tile sizes and different number of blocks. We have four tile sizes: 25, 50, 100, and 250 elements. For each, we run the program on 1 block (8XEs), 2 blocks (16XEs), or 4 blocks (32XEs).

We see that, by reducing the tile size, the execution time decreases. This is because of the higher parallelism between cores. However, we also see that, for a given tile size, as we add more blocks, there is no (or little) benefit. The execution time remains approximately the same. This is because of the suboptimal implementation of the memory allocator in FSIM.

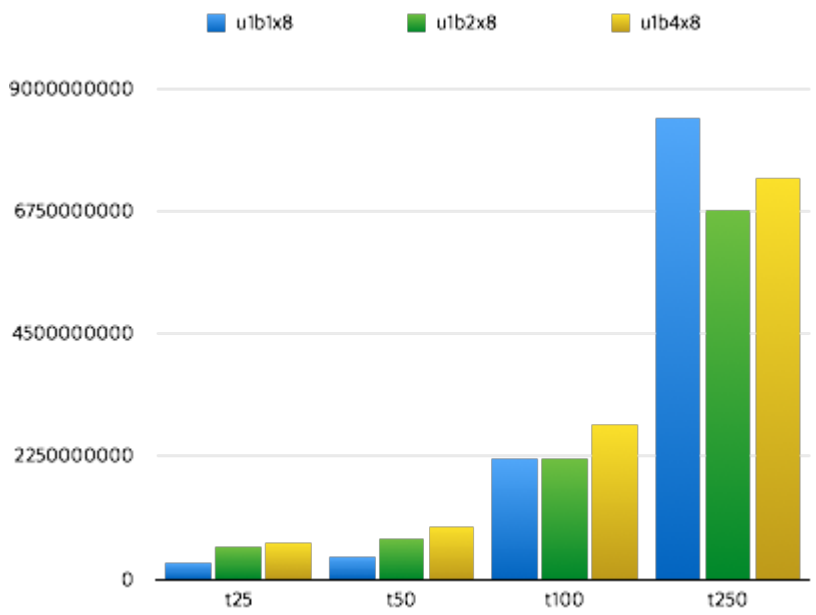

Figure 4: Execution time (in cycles) of Cholesky for matrix 500, varying tile size and varying number of blocks used (one, two or four blocks, of 8 XEs each).

We find that all remote memory references go to DRAM, rather than obtaining the data from another block's memory. To understand the effects and debug the software, we need more statistics on how the EDTs are scheduled and executed.

Plans

Continue to evaluate the architecture on FSIM, focusing on having OCR fully implemented and evaluated.

Issues

We need to work with the rest of the team members so that FSIM fully supports OCR for different numbers of processors and allocates the data in the block memories. We also need to have the support for incoherent caches fully debugged in FSIM.

Inventions

None.

Publications

None.

Pacific Northwest National Laboratory – Andres Marquez

Introduction:

During this quarter, PNNL has been working on developing the HPGMG proxy application in CnC together with Intel's assistance. Moreover, we conducted a survey and selected two mini apps to showcase the capabilities of our group locality framework. Finally, we are in the final steps to complete the OCR testbed and integrate the single node ACDT framework into it. More details to follow.

Accomplishments

Current Work:

- Development of the HPGMG application in CnC [Extended to Q11]
- Survey and documentation for proxy applications under the Group locality framework [Finalizing report]
- Single node ACDT framework and shared memory OCR testbed [Completed, Under final review]

Future Work:

- Distributed OCR testbed and ACDT framework [Q11]

Applications:

HPGMG:

Last quarter, we were looking into generalizing the control of HPGMG. We have since implemented a version of the control of HPGMG using generic steps and tags. We hope to reuse this in our final integrated version of HPGMG in CnC-OCR as soon as the code is far enough along to support multiple cycle types and different smoothers.

Regarding the fully integrated version, we have been working with Intel to create a .cnc file depicting HPGMG. Once this file is finalized, we will be integrating the individual step code for HPGMG. This is ongoing work, but our current goal is to have a simple version running by the next applications workshop in early April.

In order to provide sufficient support to the team at Intel, we have brought in a new staff member that will help us make quicker progress porting HPGMG to CnC.

Tiled Lulesh (Intel CnC):

Over the past quarter a lot of progress has been made using the Intel-CnC version of LULESH to implement tiling and explore automated tiling in CnC. We are looking at porting the efforts made here to the CnC-OCR version of LULESH which would give us a much better performance than the current per element repository version that we finished last quarter. We have also been discussing the possibility of implementing a tiling scheme, such as red black tiling to improve CnC.

CnC-OCR Lulesh:

We have started hooking the per element CnC-OCR version of LULESH into the regression test suite in the repository. This is the first non-kernel app being hooked in and will provide the OCR team valuable results moving forward.

PNNL's version of OCR has successfully run the per element CnC-OCR version of LULESH (c.f. Section 3.3), and has helped to point out areas to improve the runtime. The code identified bottlenecks in memory allocation, scaling for large events, and performance of events.

Group Locality (GL):

The report identified three examples from the Mantevo Suite of mini apps [https://mantevo.org] that showcases the advantages and short comings of the Group Locality framework. Group Locality is the concept in which threads collaborate at a very fine grain level. Such collaboration can be influence from the compute and/or memory perspectives. The current status of the Group Locality framework provides a highly parallel tiling strategy with intra-tile parallelism and a very fine grain runtime system based on micro data flow and a restructuring data space for the group of threads working together.

The provided report talks about the parallel tiling strategy plus the fine grain runtime system when they are applied to the three selected mini apps. The first mini app is called ***MiniSMAC-2D*** and it is used to solve finite-difference 2D incompressible Navier-Strokes equations. Group Locality could take advantage of the symmetric Gauss-Seidel relaxation kernel. However upon further analysis, we found that the iteration space cannot be tiled under our framework. This example showcases in which cases our tiling strategy cannot be applied.

The next mini app is called ***TeaLeaf*** and it is used to solve a heat conduction equation. It uses a 5 point stencil solver that our framework can exploit. Using both our runtime and jagged tiling framework, we achieved an increase of 30% over State-of-the-art polyhedral generated code for an Intel Phi Architecture.

The final kernel is called ***Mini AMR***, and it does an adaptive mesh refinement. It is composed of an computational heavy kernel. After applying our jagged tiling strategy, we found out that there was a degradation compared to the state-of-the-art code of around 11%. The memory profile of the application shows that the tiling methodology should have won in these aspects (i.e. lower memory trips, lower cache misses, etc). Thus, this kernel requires more study.

The report will be made available shortly after the end of February (after a more thorough analysis of the last kernel has taken place).

Architected Composite Data Types (ACDT):

As mentioned in previous reports, we developed a dynamic self-aware light-weight system, the Architected Composite Data Types Framework (ACDTF), to integrate with the Intel's Open Community Runtime (OCR)'s interfaces. The framework asynchronously samples and compresses data while an OCR program runs and is able to achieve a maximum performance improvement of 18X for a sparse Cholesky kernel. The previous framework was designed to run on top of OCR and worked on shared memory

systems. However due to the limitations of the then current OCR framework, the system could not share state information across distributed nodes.

Thus, the decision was made to develop an in-house OCR and to provide several runtime hooks that would make the integration of ACDTF possible. The objectives were to allow PNNL to improve OCR's fine-grain performance on shared memory systems, to enhance OCR with ACDTF, and to develop a scalable distributed version of OCR. These objectives are well under way and have produced very promising results.

Over Q10, a shared memory version of OCR was developed at PNNL, extended to support distributed systems via TCP/IP, and enhanced with ACDT. One caveat is that the distributed OCR does not support finish event driven tasks and exclusive write data blocks at the moment. This will be part of our Q11 work plan.

Our in-house OCR system, required for the ACDTF, managed to substantially reduce the overhead vs. public OCR, see Figure 1. Overhead is especially problematic for very fine-grain tasks. To achieve these performance improvements, PNNL used custom designed synchronization algorithms and data structures, amortized the cost of growing structures, aligned data to the cache lines, attempted to reduce false sharing where possible, and developed a custom thread-caching malloc to reduce hits on the standard allocation table locks for data under 4 kilobytes.

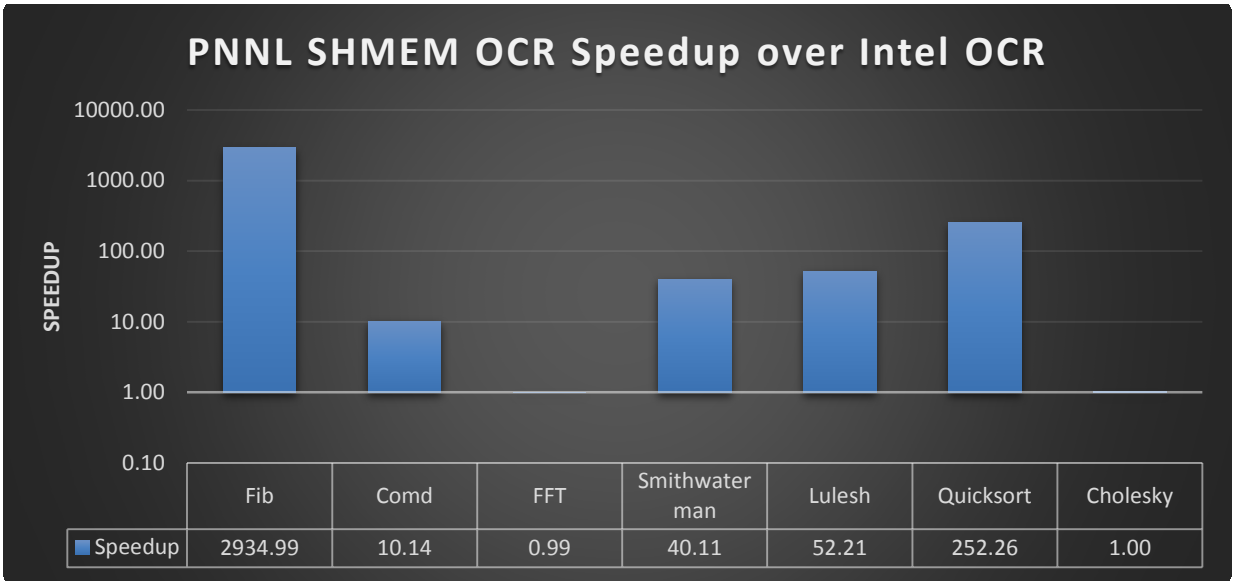

Figure 5: PNNL's SHMEM OCR performance vs Intel's OCR, running on 16 cores (gcc 4.8.3 –O2 –mtune=native, Intel Xeon CPU E5-2690) . Fib: 30, Comd: 12^3, FFT: 2^20, LULESH Domain: 4, Quicksort: 2000 elements, Cholesky: 16K^2, Smithwaterman: 800

For distributed systems, in an attempt to achieve performance within a quick development time frame, our initial design focused on developing a TCP work stealing/sharing algorithm for distributing the work across nodes. OCR is unique because data as well as work must be stolen/shared. So for the initial design, PNNL implemented a multi-threaded system that could steal chunks of work and data (known as

mugging). Furthermore, it intelligently caches data blocks that had been copied across memory to reduce network traffic. Figure 2 shows the current performance for strong scaling using work stealing. The speedup shows that there are some aspects that need to be addressed to improve the scalability of our approach. These will be targeted during Q11.

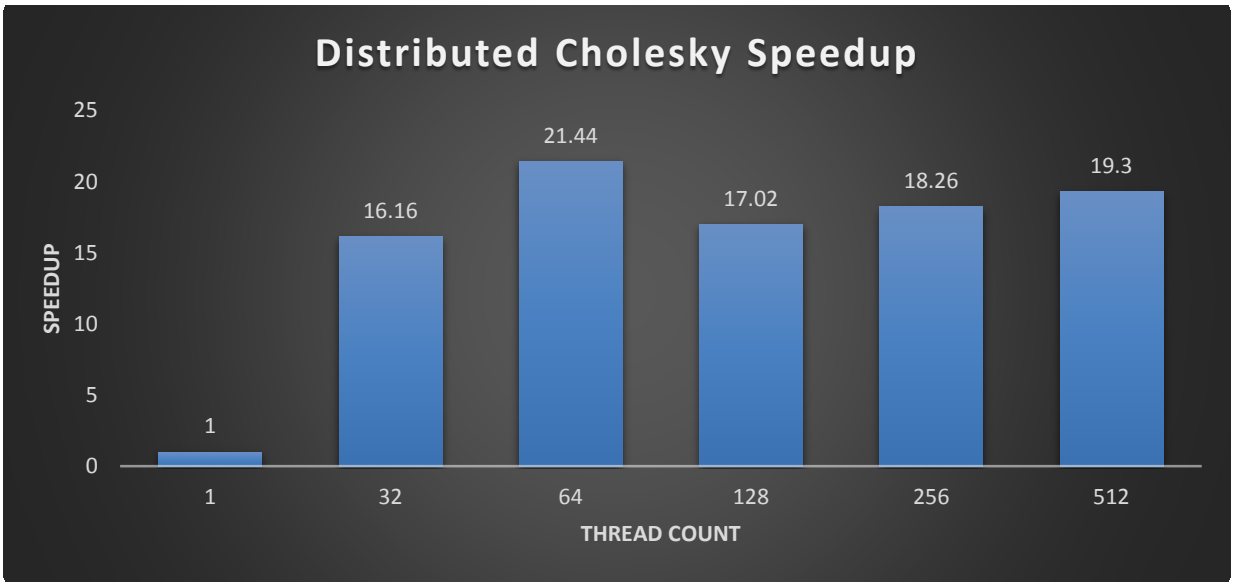

Figure 6: Speedup relative to 1 thread. Strong scaling of a 16K^2 matrix across 16 nodes. Each node has 32 cores that share 16 FPUS, thus 16X per node is the maximum performance we expect. It is nice to see we achieve additional performance moving across nodes.

For ACDTF, integrating it into OCR allowed many of the overheads to be removed because they were implemented in a manner redundant to the underlying OCR. Additionally, a new design could be done using underlying structures implemented for OCR. This included storing the global unique identifiers (GUIDs) of data in a concurrent hash map to represent the ACDT state. This reduced the redundant compression requests and state significantly. Figure 3 shows the current performance of ACDTF on 1 node. The performance is 11.77X faster than the ACDT running on top of the public OCR.

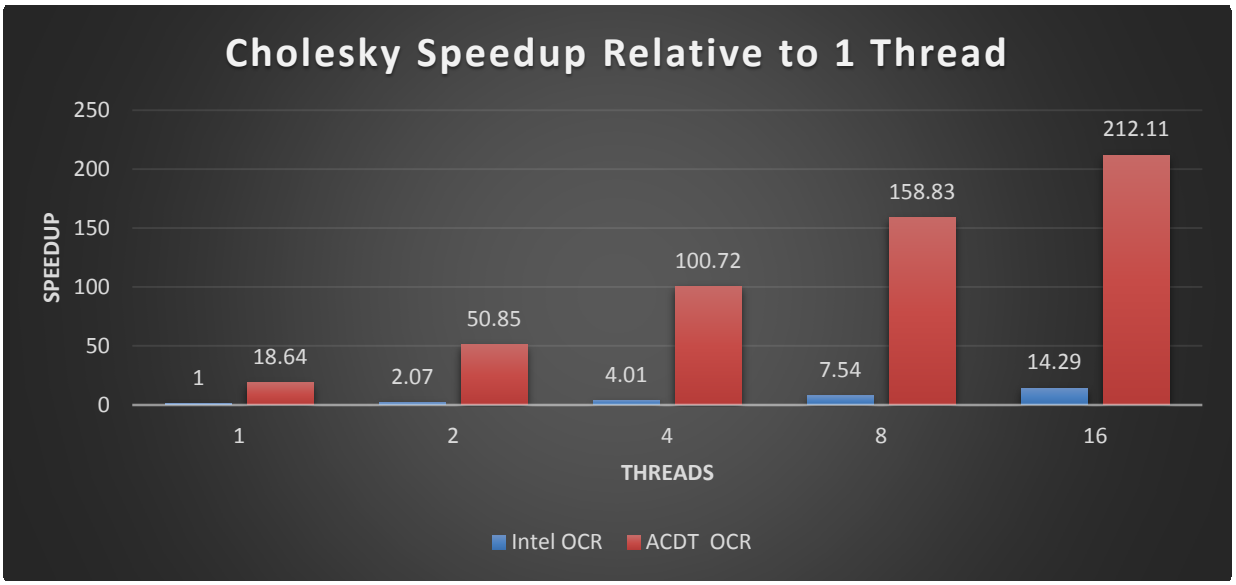

Figure 7: ACDTF samples the data of each data block asynchronously and compresses the block of 16^2 matrix. Performance is relative to 1 Intel OCR Thread.

In conclusion, the internal OCR testbed is shaping up to be an extremely useful and high performance vehicle to test our research ideas. Moreover, it provides a very competitive environment when compared to other fine grain runtime systems and models. Current work focuses on adding hierarchical scheduling to distributed OCR, finishing distributed ACDT, adding finish event driven tasks and exclusive writes to distributed OCR, and performance improvements. Future work will include extending ACDTF to support new features like resiliency.