# Milestone 8 Status Report

Award #: **DE-SC0008717**
Recipient: **Intel Federal LLC**
Project Title: **TRALEIKA GLACIER X-STACK**
PI: **Shekhar Borkar**
Report Date: **September 1, 2014**
Period Covered by Report: **June 1, 2014 to August 31, 2014**

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

# Contents

## Executive Summary

Prof. Torrellas and his team evaluated the whole TG system architecture (and the runtime system) on the FSim simulator with insightful findings about performance scalability and energy consumption. The report on the evaluation is presented as a separate document: "System Evaluation of V2.5".

To continue to strengthen the research infrastructure, ETI completed the changes in the FSim simulator, needed to transition the software stack to the new 4.1 ISA. Reservoir Labs Updated LLVM and binutils to support version 4.1.0 (64-bit) of the TG ISA, including a DMA runtime providing C interface to DMA instructions, and support in R-Stream for automatic generation of TG DMA operations through the DMA runtime interface.

Earlier, we had completed the implementation of CnC on OCR for execution on a single shared-memory node. Now we are close to completion of CnC on OCR for both, distributed and FSIM platforms, using a new, event-driven approach for the CnC runtime implementation. We are addressing a more general approach to memory management in CnC to address some concerns about memory usage. Habanero-C++ too now runs on top of OCR, a library approach that uses the features of C++ to increase productivity and allow the programmer to use Habanero constructs as library calls, while presenting an easy to use language-like interface.

The OCR version emulating the TG architecture on x86 now supports multiple "blocks" and properly emulates a TG-like memory hierarchy, which will be the basis for our multi-block version on FSim. The FSim version of OCR is fully functional for a single block, and supports multiple levels of memory hierarchy (such as block shared memory). The MPI version of OCR version is being continuously improved and should be fully merged with the other versions by the application workshop being held at the end of the month.

University of Delaware continued their efforts on a framework (SAFE) to implement self-awareness in the system software. Their work involves adapting using the heat and energy models developed for the TG simulator (FSim) and injecting them back into SAFE. After stabilizing and extending the original self-aware API, they have started adding control (through the "decide" and "act" steps) in the framework.

To evaluate the programming models and the runtime system, we have selected four primary proxy applications, and fifth one will be added soon—namely CoMD, LULESH, HPGMG and SAR. And the kernels are Cholesky, Fibonacci, FFT, Conjugate Gradient (CG), the other NAS Parallel Benchmarks, EP, IS, LU, FT, MG, HPCG, FFT and Stream. UCSD successfully developed the CoMD proxy app with two different algorithms using MPI, MPI+OpenMP, OpenMP, MPI+PThreads, PThread, and OCR.

We are planning an applications workshop (Hack-a-thon) to be held later this month, which will be well represented by our DOE co-design partners. They will get hands-on experience using TG software stack. And we are looking forward to their valuable input and insights.

Now that we are exiting the infrastructure building and move into maturing previously researched technologies, we are not extending our X-Stack relationships with ETI and University of Delaware into the third year of the contract. This will in no way impact our deliverables under the contract and we look forward to a productive third year. We thank ETI and University of Delaware for their invaluable contributions to the TG project.

| # | Due | Milestone | Lead |
|---|---|---|---|
| 1 | 11/30/12 | Architecture V2 spec & preliminary apps kernel 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 |
| 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 |
| 6 | 2/28/14 | Architecture V2.5 spec, system evaluation of V2.0 | Intel, UIUC |
| 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 |
| 9 | 11/26/14 | Arch V3.0 spec, selected apps evaluation with Exec model & PGM system for V2.5 | Intel, UCSD, Rice, Reservoir |
| 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 – Shekhar Borkar

## Introduction

During this quarter, Intel worked in the following areas:

- The OCR specification is being developed collaboratively with another team at Intel.
- Continued progress on targets for OCR: emulated TG on x86 now supports multiple blocks, TG on FSim is fully functional on a single block and the MPI implementation is being improved to bring it in line with the current API
- Identification of scalability bottlenecks in OCR: we are working on evaluating the performance of the applications we have to identify bottlenecks in the runtime. This effort is being conducted on x86.
- Simplifying the process of building and executing OCR applications; we now have a tightly integrated tool-chain that allows for quick application execution on all supported OCR platforms.
- Several bugs were fixed and features added based on requests from our partners.
- Four proxy apps where selected for focus: CoMD, LULESH, HPGMG and SAR.
- We scheduled Application Workshop 3 for September 30 – October 2 and developed the proposed agenda.

## Accomplishments

### OCR

The OCR version emulating the TG architecture on x86 now supports multiple “blocks” and properly emulates a TG-like memory hierarchy. This version will be the basis for our multi-block version on FSim.

The OCR version on FSim is fully functional for a single block. It also supports multiple levels of memory hierarchy (block shared memory for example).

The OCR version on MPI is being continuously improved and should be fully merged with the other versions by the application workshop being held at the end of September.

We improved the development environment to enable developers to be more productive in writing and executing their OCR applications. We now have a unified way to build and run applications on all platforms (including FSim). This required the implementation of simple file I/O for FSim. It is now possible to write a single program for most applications[1] and have it run on x86, TG on x86 or TG on FSim.

We are also continuing our effort to produce a specification for OCR as well as an "example" document that will detail several common application patterns and how to code them in OCR. We intend to have this document ready for the application workshop being held at the end of September.

## Applications

In order to exercise the programming models and the runtime system, we are focusing upon four primary proxy applications and we are participating in the definition of a fifth. In addition we have implemented several computational kernels as vehicles to define the measures of performance and energy efficiency, but they also of use as teaching examples for event driven programming. These examples will also help define a logical path in migrating from legacy programming models to "event driven thinking."

These proxy applications are CoMD, LULESH, HPGMG and SAR. We have begun discussions with John Bell's group at LBNL to define a proxy app for our purposes to be derived from AMR. The kernels are Cholesky, Fibonacci, FFT, Conjugate Gradient (CG), the other NAS Parallel Benchmarks, EP, IS, LU, FT, MG, HPCG, FFT and Stream

Our goal is to have the proxies and kernels supported in several of the higher level notations, CnC, H-C, HTA and Reservoir as well as hand coded versions that run directly upon OCR, thus on the TG architecture. This will allow us to quantitatively assess the performance and energy implications of TG.

During this performance period we have consolidated our efforts upon the first three proxies, CoMD, LULESH and HPGMG and now have versions of CoMD and LULESH running in both OCR-hand coded and CnC modes. Work is progressing on HPGMG and will leverage previous work done upon miniGMG on a CnC version and we are beginning wok upon a hand coded version for comparison.

UCSD is porting the two algorithms to CNC to test CnC->OCR-FSIM to compare to the hand coded version. The CnC version of CoMD for both algorithms is complete and is currently paced by planned additional CnC/OCR/FSIM development.

[1] We do not support the full standard library on FSim and are adding needed features on an as-needed basis.

UCSD has successfully developed the CoMD proxy app with two different algorithms using several different approaches including a CnC version. UCSD also completed a series of performance experiments to analyze the performance features and bottlenecks of the different algorithms and programming environments. Their experiments show that current bulk synchronous programming models are not capable of handling the anticipated dynamic environment of Exascale systems. This data was written up in a paper submitted to Co-HPC a workshop on Co-Design for SC14. The paper was also sent to the ExMatEx team and they have been in email discussion about the results.

Lulesh tiling: In order to improve performance in our CnC version of LULESH, we started to implement a tiling scheme that groups nodes and elements together into super sets. This reduces the overhead involved with creating new CnC data items. We compared the performance of this with varying size tiles and discovered that our result were comparable to and in some cases better than the OpenMP version of LULESH. The results produced are preliminary and we hope to get some final numbers and tests in the future.

Lulesh CnC-OCR: The initial CnC version of LULESH at PNNL used Intel's CnC, while waiting upon the CnC-OCR on **FSIM**. Ellen Porter (PNNL) hopes to have LULESH running on **FSIM** by the CnC 2014 conference in mid-September. She submitted a poster to SC2014 outlining the steps involved in converting LULESH to CnC showing its advantages.

HPGMG: Intel, PNNL and Reservoir have begun reviewing HPGMG code. Intel and PNNL are designing a CnC code diagram. We have a plan in place to enable the use of R-Stream optimized portions of HPGMG as CnC steps. Some progress has to be made on each independently before combining the two approaches. The process of putting HPGMG into CnC form has uncovered a possible improvement. Currently several versions of HPGMG have been identified and a user chooses one of these pre-identified versions. Because of the CnC philosophy of separation of concerns, we might be able to produce one version that is general enough to subsume existing configurations and also a wide range of other similar ones as well. We're currently investigating this possibility.

Implementation of CnC: Rice had previously completed the implementation of CnC on OCR for execution on a single shared-memory node. They are now near completion of CnC on OCR for both distributed and **FSIM** platforms, using a new, event-driven approach for the CnC runtime implementation.

Memory management in CnC: A CnC domain spec indicates the semantic meaning of the application and allows for a wide variety of approaches to tuning (distribution and placement) it also allows for a wide variety of approaches to memory management. Several approaches to memory management have been implemented in Intel and Rice implementations of CnC. None of these are yet implemented in CnC-OCR. We are investigating the existing and new approaches and coming to understand the pros and cons of each.

CnC and HTA: Intel, Rice and UIUC are investigating ways to allow the application programmer access to a combination of CnC and HTA. We've identified several distinct approaches. These approaches share the need for a methodology to convert HTA tiles to CnC data items and CnC data items to HTA tiles. This is currently under investigation.

### Plans
For the next milestone, we plan to:

- Finalize an initial OCR specification (with support from Rice and other teams within Intel) as well as a document detailing some common patterns and how to code them in OCR.
- Propose a resiliency framework
- Prepare for and conduct Application Workshop 3 on September 30 – October 2, 2014.

### Issues
None.

### Inventions
Four subject inventions were disclosed and are being reported separately.

### Publications
None.

## ET International

### Accomplishments
This quarter, ETI has finished the changes needed to transition the software stack to the new 4.1 ISA within FSim.

Our primary accomplishment for the quarter has been in the support and addition of features to FSim for the UIUC evaluation. To that end, we added additional PMU tracing commands, added CE PMU reporting of clock cycles, finalized and tested infrastructure to import NDA restricted energy numbers provided by Intel, and added specific energy accounting for memory. Additional hours were spend ensuring UIUC used of FSim fully understood how to set configuration files appropriately to achieve good performance and also to extract all the information required for their review.

ETI's prototype for the Power API for the FSim framework has initial configuration completed but still lacks runtime based functionality necessary to make it compliant with the x86 implementation. We will be transitioning this work to Intel at the end of the quarter.

### Status
The ISA changes are completed for all instructions currently implemented in FSim. Small amounts of testing was started however the full regression test suite was not ported over to the 4.1 ISA. This work has been documented and transitioned to Intel.

Currently, FSim models detailed dynamic energy consumption of instructions and accounts for static energy per cycle in real-time. There is an energy estimation of network energy outside of the chip when using tracing with the possibility to refine and internally compute the model using all the parameters exposed within the FSim framework. We have implemented energy consumption for DRAM, Chip

Shared Memory, Unit Shared Memory, and Block Shared Memory within the FSim framework which can automatically pull the needed information from the energy file.

### Issues

For the first month of this last quarter, we were still unable to test and verify the ISA due to lack of binutil support. This led to a freeze on the system architecture for evaluation purposes. By mid-July, binutils was available however priority was given to make energy accounting improvements and provide support on the old ISA for the UIUC team to do their evaluation.

Our energy infrastructure was extended however the energy information within the user provided file isn't updated with all the latest information on scaled energy values. This should simply be an update and sanity check from Intel as the UIUC team has tested the functionality extensively in their evaluation.

### Plans

Our work on the Traleika Glacier X-Stack program concluded with this milestone.

### Publications

None.

### Inventions

None.

## Reservoir Labs - Richard Lethin

### Introduction

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

- Updated LLVM and binutils to support version 4.1.0 (64-bit) of the Traleika Glacier (TG) ISA.
- Created a DMA runtime that provides C interface to DMA instructions for 32-bit and 64-bit TG ISA.
- Implemented support in R-Stream for automatic generation of TG DMA operations through the DMA runtime interface.
- Investigated configurations and identified critical sections in HPGMG benchmark that dominate the overall execution time; these critical sections are the candidates of immediate focus for applying compiler and runtime optimizations.
- Identified and executed the initial steps towards developing a proxy application for AMR that can be used for evaluating the exascale features of TG software stack.

### Accomplishments

This section details our contributions during this reporting period.

## LLVM for 64-bit ISA

The LLVM backend was provided a major update to support version 4.1.0 (or the 64-bit version) of the TG ISA. The TG architecture earlier had employed a 32-bit ISA. The opcode names, number of operands for most instructions have been revised in V4.1.0. The LLVM code-generation backend was modified to encode the new ISA. Furthermore, the new ISA provides certain new capabilities that were not present in the prior instruction set. The enriched ISA has been leveraged to generate target code that is smaller in size and higher performing compared to code that was being generated for the prior ISA.

### *Revised ISA Encoding*

The instruction formats are changed in V4.1.0. The changes are geared towards making instructions amenable to future extensions and also, reducing the number of different forms of an instruction for different operand sizes by adding a field to specify operand size.
A representative revision is shown below. The ISA version 4.0.8 defines the following ***bitwise or*** instruction format.

*or64 r1, r2, imm12*

The above instruction performs or between 64-bit operand in register r2 with the sign-extended 12-bit immediate value imm12 and stores the 64-bit result in register r1. ISA 4.1.0 provides the following ***or*** as a replacement:

*bitop1 r1, r2, imm28, BitOp, SIZE*

It performs the bit operation between a value in register r2 with the sign extended 28-bit immediate value and stores the result in register r1. The actual bit operation to be carried out is indicated in BitOp field (one of them is ***or***), and the size of values in registers r2 and r1 is specified via SIZE operand.
The LLVM backend has been updated to emit assembly code in the new instruction formats.

### *Fewer Comparison Operations*

Certain comparison operators – greater than (>), greater than or equal to (≥), not equal to (≠) have been added to V4.1.0. The LLVM backend has been equipped to make use of the new operators, which reduces the generated assembly code size and increases the performance of the program. Additionally, this allows us to convert *unordered* floating point comparison operations into *ordered* comparison operands. For example, unordered a < b is true if either a or b is undefined or if both are defined, a < b must be the case. Ordered a < b on the other hand, is true only when a and b both are defined and a < b is true.

The TG architecture by default executes ordered instructions. As a consequence, when unordered instructions are encountered, additional comparisons have to be inserted to check if the operands are undefined. Hence, it is desirable to only emit ordered comparisons. Since V4.1.0 ISA provides a full spectrum of comparison operands, the LLVM backend flips the unordered comparison operation (for example, > is flipped to ≤) and the resulting comparison is ordered by construction. It also flips the branch targets to preserve correctness of the generated code.

### *Larger Register File*

The 32-bit ISA encoding could fit a 6-bit register address. Therefore, the number of registers that the TG architecture could support was $2^6 = 64$. In the 4.1.0 ISA, a register address can be 9 bits long, enabling a program to use $2^9 = 512$ registers. The LLVM backend has been modified to emit code making use of all 512 registers.

### *Faster Loading of Immediate Values into Registers*

The 4.0.8 version of the ISA allowed at most 18 bits of an immediate value to be loaded into registers at a time.

*movimms r1, imm18*

This implied that, when there is an immediate value of length greater than 18 bits, multiple movs have to be initiated to fully load the value into a register. The 64-bit instruction encoding in V4.1.0 has room to move up to 42 bits of an immediate value into a register in a single instruction.

*movimm r1, imm42, SIZE*

This means that if we have an operand of size at most 42 bits, it can be transferred into a register in a single ***mov***. Further, the total number of ***movs*** needed for a larger immediate value is also reduced. Consider the following function.

```
long test(void) {
   return 1L << 20;
}
```

V4.0.8 generates the following code that requires two ***movs*** to load a 20-bit immediate value.

```
test:
movimmz r1, 16
movimmshf16 r1, 0
jabs r63
```

V4.1.0 accomplishes the required movement in a single instruction.

```
test:
movimm r1, 1048576, 64
jabs r511
```

## Binutils for 64-bit ISA

A version of binutils has been ported to support version 4.1.0 of the TG ISA (informally, the 64-bit version of the ISA). Binutils is a set open-source programming tools for creating and managing binary programs, object files, libraries, profile data, and assembly source code, including:

| | |
|---|---|
| as | assembler |
| ld | linker |
| addr2line | convert address to file and line |
| ar | create, modify, and extract from archives |
| nm | list symbols in object files |
| objcopy | copy object files, possibly making changes |
| objdump | dump information about object files |
| ranlib | generate indexes for archives |
| readelf | display content of ELF files |
| size | list total and section sizes |
| strings | list printable strings |
| strip | remove symbols from an object file |

and several others. In addition, we were asked to port a separate utility, elf2hex, originally written by engineers at Intel. In addition to porting binutils to the 64-bit ISA, we changed the file format used for object code, executable files, etc. from Elf32 to Elf64. Given the way binutils is organized, we weren't required to individually port each of the above programs; instead, certain core libraries had to be rewritten and the many programs were all generated automatically, in an all-or-none fashion.

We began with a version of binutils targeted to a combination of the 32-bit ISA and Elf32. Extensive rewrites were required in four files:

- A header file defining all the operand fields in the various instructions (the semantics and their length and position in the instruction word). In the 64-bit ISA, there are 42 different kinds of field, a relatively large number.

- A C file that is primarily a large table with entries for each instruction, where an entry specifies the instruction's mnemonic, encoding, operand fields, etc.
- A C file that has code to assemble each of the different operand fields.
- A C file that has code to disassemble each of the different operand fields.

In addition, a further 45 files required relatively small edits, some as trivial as rewriting certain constants with a trailing L to indicate that they were part of a 64-bit computation.

### Supporting TG MemISA in R-Stream

A DMA runtime was created for both the 32-bit (V4.0.8) and 64-bit (V4.1.0) TG ISA. This runtime provides a straightforward C interface to DMA instructions previously available only through inline assembly language. The runtime is implemented as strided DMA functions where both source and destination locations may have independent strides. All code was compiled with the LLVM and binutils toolchain for the applicable ISA.

A test suite was implemented, compiled and run successfully on the 32-bit F-Sim code checked into the X-Stack *git* repository. The test suite consists of six subtests that exercise features from simple one-dimensional data transfers to DMA where both input and output arrays have different strides. All tests in the suite pass and the code executes in under one minute.

The runtime is designed to seamlessly integrate with R-Stream OCR code generation facilities. The runtime was given to Kelly Livingston at ETI and he was able to integrate it with an R-Stream generated OCR matrix multiplication kernel in one afternoon.

### AMR Proxy Application

AMR or Adaptive Mesh Refinement is a very important class of algorithms that are used widely as the backbone of large-scale computer simulations. One way to perceive AMR algorithms is to view them as grid-management frameworks, i.e., frameworks that are used to manage an hierarchy of structured grids where each level of the hierarchy represents a refinement level from the coarsest (e.g. level 0) to the finest (e.g. highest level).

Each level of an AMR grid hierarchy consists of areas of interest or in other words sections of the grid that capture interesting physical activity according to predefined physical rules (e.g. pressure gradient thresholds etc.). In the context of AMR these sections are called boxes simply because they are captured by hyper-rectangular boxes of structure grid. As a result, an AMR grid hierarchy can also be viewed as an hierarchy of hyper-rectangular boxes or simply boxes that represent refinement patches of structured grid.

This approach is very useful because it allows selective and adaptive refinement of the resolution of the grid while keeping the PDE solvers simple by focusing on the structured grid patches. The structured nature of the PDE solvers alleviates a huge burden from domain-experts that work on those solvers for the price of a more complex grid-management framework that is harder to optimize in a portable manner for future exascale parallel computer architectures like TG.

Our objective is to mitigate this tradeoff by exploring the steps required towards the development of an AMR proxy application that would essentially be a simpler version of the full-scale AMR application. This proxy application will be the primary vehicle for investigating compiler optimizations that would eventually be directly applicable to the full-scale AMR applications.

### *Steps toward an AMR Proxy Application*

During this quarter, our efforts at Reservoir Labs were concentrated on identifying and executing the initial steps towards the development of an AMR proxy application. We consider these steps to be the following:

a. Identify the mechanism for constructing and maintaining dependences between boxes of different levels (inter-level dependences) as well as boxes within the same level (intra-level dependences).
b. Identify inter-processor communication operations for data exchanges between dependent boxes.
c. Put everything together and form a minimal pseudo-algorithm that encompasses our findings from steps **a** and **b**.

We believe that these three steps are very important because they reveal a much clearer picture of the high-level control-flow structure of AMR. We also believe that the third step – which is the outcome of our work during this quarter – can serve as an excellent starting point for further discussions with domain-experts and also as a blueprint for preliminary implementation efforts.

One of the main tasks involved in executing these three steps was to perform a detailed study of an AMR application in order to identify sections of code relevant to steps **a** and **b** as well as those sections that are not relevant to steps **a** and **b**. The latter are represented by black-boxes in our pseudo-algorithm (i.e., step **c**) and include sections like: initialization, mesh-refinement policy, PDE solvers etc. We believe that frequent interaction with domain-experts is necessary for better understanding of the importance and relevance of these code sections to the development of the proxy application. In other words, we consider the next step forward to be the characterization of these black-box sections of code and ultimately the definition of their role in the AMR proxy application.

## HPGMG Study, Analysis, and Mapping

### *Reference Platform Definitions*

Configuration of the HPGMG benchmark is a non-trivial task. The benchmark consists of two different code bases that can be compiled and run independently. One is a finite element code; the other is a finite volume. Further, users must set a number of options at build time, including multigrid cycle, bottom solver, and smoother. Finally, actual problem size is specified at run time. Initially it was unclear which set(s) of options could be combined to produce a proxy application for both current and near-future multigrid solvers that was capable of being implemented on and optimized for the TG architecture.

Reservoir Labs investigated configurations with one of the HPGMG authors, and found that modern GMG solvers can best be proxied with the finite volume code configured with V-cycles, a biconjugate gradient stabilized (BiCgStab) bottom solver and red-black Gauss-Seidel (GSRB) smoothers operating at each level of the multigrid code. Future GMG solvers are best proxied with finite volume code configured with full multigrid F-cycles, a BiCgStab bottom solver, and Chebyshev smoothers at each level.

### *Identification of Performance Critical Sections*

Examination of HPGMG timing out showed performance critical sections include smoothing, restriction, interpolation, and ghost zone exchange.

### *R-Stream Optimization*

After identifying the smoother as a bottleneck area, the GSRB and Chebyshev smoothers are being modified to enable parallelization and optimization by the R-Stream compiler.

### *CnC Collaboration*

Reservoir has been collaborating with the CnC team at Intel/Rice and the PNL team. A guide to building and running HPGMG was presented and subsequent collaborations have detailed a plan to enable the use of R-Stream optimized portions of HPGMG as CnC steps.

## Plans

For the next milestone, we currently have key ongoing tasks:

- Support LLVM and binutils for 64-bit TG ISA.
- Make progress on mapping various performance critical sections of HPGMG through R-Stream to produce scalable OCR code on x86 and F-Sim.
- Continue discussions with John Bell and his group at LBL along with other TG team members to concretely define a proxy application for AMR that can be used for TG software stack evaluation.
- Work on the final implementations of R-Stream optimizations for producing efficient and scalable OCR code for TG architecture.

## Issues

None.

## Inventions

None.

## Publications

None.

## Conclusion

During this quarter, we completed a major update to the LLVM and binutils tools to support the 64-bit TG ISA. We made progress in extending high-level compiler optimizations for TG through R-Stream. Specifically, 1) we created a DMA runtime that provides a C interface to DMA instructions for 32-bit and 64-bit ISA and 2) we completed implementing the support for automatic generation of TG DMA routines in R-Stream through the DMA runtime interface. We took initiatives within the TG team to focus on the HPGMG benchmark for evaluating the TG software stack, especially, the compiler and runtime components. We made progress in using R-Stream to produce scalable OCR code for the benchmark, specifically optimizing the critical sections of the benchmark that dominate the execution time. We also initiated efforts to develop a proxy application for AMR that will serve as an important application benchmark for TG evaluation.

# Rice University - Vivek Sarkar

## Accomplishments

We have implemented distributed OCR for clusters of nodes, using MPI as the communication layer. We also have an implementation of the distributed OCR using GasNet as the communication layer.

We have designed several API improvements to OCR to support user- and compiler-generated hints that communicate properties of OCR objects to the runtime, such as spatial locality, temporal locality, repeating patterns, amount of parallelism generated, computation/bandwidth requirements.

We have made several performance and scalability improvements to the OCR implementation in response to concerns raised by the application collaborators.

We have ported the OCR implementation of Conjugate Gradient to distributed OCR and demonstrated its functionality and scalability on several cluster nodes.

Earlier, we had completed the implementation of CnC on OCR for execution on a single shared-memory node. We are now near completion of CnC on OCR for both distributed and FSIM platforms, using a new, event-driven approach for the CnC runtime implementation. We are designing a more general approach to memory management in CnC to address some concerns about memory usage in CnC.

This approach will implement run-time checking that will allow expert programmers to reuse CnC item memory while still respecting (at a logical level) the dynamic single assignment property of CnC.

We are currently adding support for exclusive-write access to Data Blocks in distributed OCR. This feature is critical for completing the CnC on OCR (distributed and FSim) implementations.

We have implemented Habanero-C++ on top of OCR, a library approach that uses the features of C++ to increase productivity and allow the programmer to use Habanero constructs as library calls, while presenting an easy to use language-like interface to the programmer. We are currently implementing a novel and scalable distributed finish using OCR as the runtime on a shared-memory node and UPC as the cross-node communication layer.

We have installed, tested and documented the OCR on FSIM implementation at Rice, as part of the plan for Rice to take ownership of the OCR on FSIM.

## Plans

Milestone 9 is on track. We are designing different heuristics for adaptation in the OCR runtime that will perform code and data movement to optimize for user-specified policies (performance, power, energy).

## Issues

None.

## Inventions

None.

### Publications

"Bounded memory scheduling of dynamic task graphs”**.** Presented at The 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT 2014), August 2014 Dragos Sbirlea, Zoran Budimlic, Vivek Sarkar

"Dynamic Declarative Tuning for Locality”**.** Sanjay Chatterjee, Zoran Budimlic, Kathleen Knobe and Vivek Sarkar Draft. Will be submitted for IPDPS 2015

## UCSD - Laura Carrington

### Proposed milestone

Work on experiments with CoMD on OCR-FSIM to explore tradeoffs with the two algorithms and the TG architecture (Hardware-software co-Design). Work on porting the two algorithms to CNC to test CnC->OCR-FSIM to compare to the hand coded version.

Interpretation: 1) Describe preliminary trade-offs of two algorithms: propose metrics, apply metrics, show quantitative results of metrics, and provide a qualitative statement about relative effort of making CoMD work in both forms (quantitative where applicable). 2) Work with Bill F and DOE to identify Proxy App for next Q milestone by end of this Q; provide a written statement of Proxy App to be used in next Q efforts.

### Issues and Limitations Encountered

We completed development and testing of multiple versions of CoMD two different algorithms each programed using MPI, MPI+OpenMP, OpenMP, MPI+PThreads, PThread, and OCR. Upon evaluation of the performance of the OCR version relative to all other versions performance and scalability issues in OCR were identified. The Rice team has the smaller CG kernel code, which exhibits the same performance and scalability issues. They are using this to test various solutions to the OCR issues. Vincent Cave was working on this issue from Rice but his priorities have been shifted, He is expected to begin working on the issue again sometime in Sept.

The CnC version of CoMD for both algorithms is complete and we are currently waiting on Nick Vivrlo to get a CnC->OCR-FSim working. Current priorities are on distributed CnC-OCR and the CnC->OCR-FSim is not currently being worked on. Without that version no experiments on the simulator can be performed.

### Progress

We successfully developed the CoMD proxy app with two different algorithms using MPI, MPI+OpenMP, OpenMP, MPI+PThreads, PThread, and OCR. We implemented several variants of the algorithms that differ in the way they deal with concurrent updates in the force calculation, and in how the work is decomposed into tasks. We also completed the CnC version of CoMD using both algorithms. We completed a series of performance experiments to analyze the performance features and bottlenecks of the different algorithms and programming environments. In addition we completed experiments to

highlight how current bulk synchronous programming models are not capable of handling the anticipated dynamic environment of Exascale systems. This data was written up in a paper submitted to Co-HPC a workshop on Co-Design for SC14. The paper was also sent to the ExMatEx team and we have been in email discussion about the results.

We updated the CG/OCR code and implemented a tiled version. Following suggestions from Rice, we also refactored CG and CoMD to avoid dependencies in favor of parameters, although only in CG (at the finest granularity) the optimizations improved the performance.

### Plans

We are currently working on porting the OCR CoMD code to the TG-FSim simulator. We have also identified HPGMG as our next application to port to OCR and have begun work on that process.

### Inventions

None.

### Publications

None.

## University of Delaware - Guang Gao

### Progress on Self-Awareness

UD has been working on a single main aspect during milestone 8: self-awareness.

Continuing the efforts started in March 2014, UD has been working on a framework, called Self-Aware FramEwork (SAFE). It models a Traleika Glacier chip from a hierarchical standpoint: it features a single chip, divided in 16 units; each unit is composed of 16 blocks; each block is composed of 8 execution engines (XEs), and 1 control engine (CE) for a total of 256 Control Engines and 2048 cores. The goal of SAFE is to implement the self-aware API proposed in milestone 6, and provide an abstract environment to test its efficiency when applying the control theory based observe-decide-act loop over a (logical) hierarchy of hardware components. Blocks adapt locally to a set of constraints (*e.g.*, a power goal, a temperature threshold, *etc.*).

In the TG architecture model, each block is under the control of its CE for task scheduling, I/O handling, as well as local self-awareness, *i.e.* introspection and self-adaptation. For each other higher level in the hardware hierarchy, a CE must be selected. It is done arbitrarily in SAFE, but ideally it should be the result of an election algorithm to avoid having CEs cumulate too many roles, and to account for potential resiliency issues. Hence for each hardware level in the hierarchy (block, unit, chip, or even rack), there must be a designated “super-CE” in charge of applying self-awareness for the group of blocks belonging to that level. In SAFE, there are thus 256 block-level CEs, 16 unit-level super-CEs, and 1 chip-level super-CE. When a CE is also assuming the role of super-CE, it does not relieve it of its responsibilities w.r.t. local self-awareness. In SAFE, a block is identified through a tuple <$GlobalBlock_{ID}$, $Unit_{ID}$, $Block_{ID}$>. $GlobalBlock_{ID}$ is the “absolute” ID of a block within the chip, regardless of which unit it

belongs to: it takes a value from 0 to 255. $Unit_{ID}$ is a number going from 0 to 15. Finally, $Block_{ID}$ is the "local" ID of a given block within a unit; its value goes from 0 to 15. Currently, SAFE gives the job of chip-level "super-CE" to the block which ID is <0,0,0>, and a block is a unit "super-CE" if its $Block_{ID}$ (the "local" ID within a unit) is 0; the block ID thus follows the pattern: <$GlobalBlock_{ID}$, $Unit_{ID}$, 0>.

Temperature and power consumption are aggregated from the lower control engines to their parents, where a global decision is made. For example, a unit-level control engine may lower power and temperature thresholds of its subordinate CEs to force them to adapt (the API was modified to allow this behavior). It could also simply order them to perform dynamic voltage and frequency scaling (DVFS) "right now."

In Milestone 7, UD reinforced its framework infrastructure, by adapting the heat and energy models developed for the TG simulator (FSim) and injecting them back into SAFE. After stabilizing and extending a bit the original self-aware API, UD has started adding control (through the "decide" and "act" steps) in the framework. Another part of the infrastructure effort was to reduce the amount of global synchronization that blocks had to go through: instead of synchronizing at every simulated cycle, blocks (*i.e.*, threads) only synchronize through a barrier construct every million (simulated) cycles. Several synthetic workloads have also been devised, some purely random, to stress the system, others more "structured" to reflect specific workloads, such as traditional linear algebra kernels (e.g., BLAS2: matrix-vector multiplication). UD will provide a technical report to summarize the findings w.r.t. those experiments.

Special care has been provided by UD to document the code, and facilitate the transition from UD to Intel, thus hopefully helping Intel's engineers to take over and adapt UD's code back into FSim to add self-awareness to the OCR port on FSim. The self-aware API has aspects which are directly related to the OCR data structures—in particular, the global unique identifiers (GUIDs) used in OCR were already used in the previous runtime system executing on FSim (IRR), to cache meta-data pertaining to the hardware usage of OCR event-driven tasks (EDTs, an implementation of the codelet model proposed by UD). Such meta-data reflects the use of several components: floating-point operations, integer operations, "near" and "far" memory operations, *etc.* These are not yet implemented back into SAFE or in the OCR/FSim runtime system, but the code located in IRR should be easily ported back to either framework.

## Issues

Stabilizing the framework took slightly more time to perform, thus reducing the time UD had to start experimenting with user goals/policies, and the decision-action steps of the ODA loop. However, this does not hamper UD's writing of a technical report related to UD's findings w.r.t. self-awareness.

## Achievements w.r.t. Milestone 8's goals

Milestone 8's initial goals were the following:

1. To test several sequences and distributions of "synthetic tasks" on the framework, to evaluate the scalability of the protocol as well as its ability to adapt to different workloads.
2. To add new kinds of decisions outside of DVFS (such as clock-gating parts of the system)
3. To add new kinds of goals, *i.e.*, user-driven goals (*e.g.*, power envelope to respect).

4. To summarize findings in a technical report, and study the impact of the overall codelet-based model.

Goal 1 was reached: UD ran various synthetic tasks using our framework, and measured instantaneous and average power consumption, as well as temperature of each execution. Such tasks include "fully random" sequence of instructions, as well as "sampled" instructions that reflect more realistic workloads (*e.g.*, 30% memory operations, 30% integer arithmetic operations, and 40% floating-point operations). By changing the frequency at which control engines were performing the ODA operations, we have been able to lower the overhead related to applying resource management in SAFE. While this is not fully representative of what will happen in the real TG chip or even FSim, this is already a good indicator.

Goal 2 was partially reached: UD has added energy consumption aspects to account for clock-gating. However, actual control policies which use clock-gating or power-gating are still in the making.

Goal 3 was achieved: to the initial temperature thresholds, UD has added power goals. Contrary to temperature thresholds, which could (arguably) be used homogeneously over all blocks in a TG chip, power goals must be "divided" among individual units, then divided again among blocks: the power budget of an individual block is a fraction of the power budget of a unit, etc. The power budget is defined using the messages of the API and can be changed at runtime according to the control policies. Likewise, since instantaneous power readings can vary widely from one sample to the other (due to the different nature of the computation phases), the threshold cannot be "hard:" some latitude must be had for the block to avoid unnecessary cycling from full voltage/frequency to near-threshold voltage modes.

Goal 4 will be fulfilled by September 1, 2014. The technical report UD is preparing describes:

- The architecture of the self-aware framework
- The updated self-aware API
- The types of workloads that have been fed to the framework, including how to use the scripts to generate them (and others)
- Our first results using the ODA loop, including a comparison between full-voltage full-frequency, full-NTV, and control-based executions of synthetic tasks, w.r.t. power and temperature, on various numbers of blocks of the TG chip architecture.

## Plans

Our work on the Traleika Glacier X-Stack program concluded with this milestone.

## Inventions

None.

## Publications

Stéphane Zuckerman, Aaron Landwehr, Kelly Livingston, and Guang R. Gao. "Toward a Self-Aware Codelet Execution Model" in *PACT 2014 Workshops: Dataflow Execution Models for Extreme Scale*

*Computing (DFM 2014),* Edmonton, AB, Canada, August 24, 2014. P*resented at the workshop.To be published.*

# University of Illinois Urbana Campus – David Padua

## Accomplishments

We have been developing the Hierarchically Tiled Array (HTA) library as an abstraction of the event driven task (EDT) execution model, specifically with the Open Community Runtime (OCR). HTAs allow a concise representation to control locality and parallelism on tiled array data structures.

### Performance Improvements

The focus of this quarter was on the evaluation of HTAs to improve the performance of our system at two levels. The first level is the implementation of HTAs on top of PIL. The second level is the generation of OCR code from PIL. We have identified one promising optimization at both levels to reuse small allocated arrays. This optimization is expected to have a dramatic effect on algorithms that have many iterations of parallel loops by significantly reducing the number of calls to allocate new arrays.

### SPMD Implementation of HTAs in OCR

In the vein of performance improvements, we have been striving toward a SPMD implementation of HTAs on top of OCR. We believe that this implementation will allow us to remove some barriers, and thus overhead, from the system. We will achieve this by only using point-to-point synchronization when necessary, instead of global barriers for synchronization.

We have had a long term goal of implementing task parallelism in PIL. We made the decision this quarter to focus on the implementation of the task parallel design, and slightly postpone the implementation of SPMD. We believe that in the long run this will be better, since the SPMD implementation can be done easily using the work from implementation of task parallelism. We have updated our milestones already to reflect this delay.

## Issues

### R-Stream Integration

We continued the work on integration of the R-Stream compiler with our work on HTAs as discussed in previous quarterly reports. This work is proving to be more difficult than previously expected. We uncovered a new bug and have been working with Reservoir to resolve the issue. However, progress has been stalled due to legal reasons on releasing proprietary information on Reservoir's side. They are currently working with their legal team to resolve this issue.

### Targeting OCR on Distributed Multi-Node x86 Systems

Moving forward we will need to shift our focus to targeting OCR on a shared memory x86 machines. We had originally planned to target OCR on distributed x86, however the implementation of OCR on multi-node systems has taken too long, and we will not have time to use it and still meet our future milestones.

## Goals for Next Quarter

### SPMD HTA on OCR

As mentioned above, we plan to complete the SPMD implementation of HTAs on OCR in Quarter 9.

### Performance Improvements

We plan to continue our effort to optimize the implementation of HTAs on OCR on two fronts. We will continue to seek out and remove overheads in the implementation of HTAs as well as in the OCR code generated.

### Graph Operations in HTA

We want to expand the HTA API to include graph operations and optimizations on graph computations. In Quarter 9, we will evaluate opportunities to incorporate these graph notations into the HTA API.

## Inventions

None.

## Publications

This quarter we completed our first paper on the work in this project. In the paper, we proposed using Hierarchically Tiled Arrays as a high-level abstraction for writing programs to execute on runtime systems based on the codelet model. It presents a strategy for mapping HTA programs to codelet execution in a general sense, not limiting to a specific codelet runtime system implementation. The approach we described in the paper does not require any global barriers between HTA operations and allows minimal synchronization among codelets. The mapping strategy also aims to respect the distribution of tiles explicitly programmed by the software developers in order to exploit the locality of the algorithms. We believe this is a good first step toward using high-level abstractions to improve the programmability and we expect to delve more into the performance optimizations that can be used to enhance this design in the near future. The citation for the paper is below and a copy of the paper can be found here:

http://www.cs.ucy.ac.cy/dfmworkshop/wp-content/uploads/2014/08/DFM2014-10-Hierarchically-Tiled-Array-as-a-High-Level-Abstraction-for-Codelets.pdf

Chih-Chieh Yang, Juan C. Pichel, Adam R. Smith, David A. Padua. *Hierarchically Tiled Array as a High-Level Abstraction for Codelets.* In the Fourth Workshop on Data-Flow Execution Models for Extreme Scale Computing, 2014

# University of Illinois Urbana Campus - Josep Torrellas

## Accomplishments

In this quarter, we accomplished two main things:

1) We have evaluated the whole TG system (architecture and runtime system) on the FSim simulator. The report on the evaluation is presented as a separate document: "System Evaluation of V2.5".
2) We evaluated the Software-Managed Caches API integrated with a compiler. Recall that we had designed an API to manage the memories in a TG chip in software as incoherent caches, using instructions for invalidate (INV) and writeback (WB). In previous quarters, we had integrated the API with the Polyhedral compiler of Prof. Sadayappan. In this quarter, we evaluated the enhanced compiler running the PolyBench benchmark suite on a SESC-based simulator. We cannot yet use FSim because FSim does not include all the necessary instructions. Our algorithms have two parts. For affine computations, the algorithms precisely identify data dependences, using the Polyhedral-model machinery. For irregular computations, the algorithms use inspector-executor techniques. The results of the simulations have shown that the compiler techniques developed are competitive with hardware coherence schemes in terms of their efficacy to preserve cache locality. We submitted a paper to SC14 on "Compiler Support for Software Cache Coherence" but was rejected.

   For more irregular applications, we have a semi-manual pass where the synchronization library inserts the INV and WB instructions. Specifically, when the barrier library is called, it first issues a WB for all the dirty cache lines, then performs the barrier, and then issues an INV for all the cached data. A similar operation is done inside the lock library. However, the programmer can specify the subset of data structures to write back and to invalidate, through special directives. This saves a lot of traffic and improves performance. Our simulation results have shown that the execution of irregular applications on incoherent cache hierarchies can deliver reasonable performance. For execution within a cluster of processors, the performance is comparable to simple support for hardware coherence. For execution in multiple clusters, the performance is lower, but it scales with the processor count. We have submitted a paper for publication.

## Issues

We need to work with the rest of the team members to fully insert the API for software managed caches in FSim. Then, we need to evaluate it.

## Plans

Continue to evaluate the TG architecture on FSim, and take the software managed cache API, insert it on FSim and evaluate it.

## Inventions

None.

## Publications

Submitted for publication:

1) Title: Compiler Support for Software Cache Coherence
   Authors: Sanket Tavarageri, Wooil Kim, Josep Torrellas, and P Sadayappan

2) Architecting and Programming an Incoherent Multiprocessor Cache Hierarchy
Authors: Wooil Kim and Josep Torrellas

# Pacific Northwest National Laboratory – John Feo

## Accomplishments

### Lulesh tiling

In order to improve performance in our CnC version of LULESH, we started implementing a tiling scheme that groups nodes and elements together into super sets. This reduces the overhead involved with creating new CnC data items. We compared the performance of this with varying size tiles and discovered that our result were comparable to and in some cases better than the OpenMP version of LULESH. The results produced are preliminary and we hope to get some final numbers and tests in the future.

### Lulesh CnC-OCR

When we set out to write a CnC version on Lulesh, our initial goal was to run the code on FSIM using CnC-OCR. This wasn't quite ready to go when we were so our initial CnC version of LULESH uses Intel's CnC. Since then CnC-OCR has been tested on FSIM and is working, so we have picked up that effort again. The steps involved include; create a text based CnC graph, run script that converts the text based graph to CnC-OCR code, move existing CnC step code into CnC-OCR template, test and debug on x86 architecture, test on FSIM. This is currently in progress, and we hope to have LULESH running on FSIM by the CnC 2014 conference in mid-September.

### HPGMG

We began reviewing HPGMG code and designing a CnC code diagram

### ACDT

We are concentrating our efforts on porting ACDT to OCR, starting with the balanced and unbalanced ACDTs types. This effort will involve the porting of a couple of compression algorithms, an enhanced compression format and parts of the ACDT runtime.

### SC2014 Poster

We worked on and finished a poster outlining the steps involved in converting LULESH to CnC. This was submitted to SC2014.

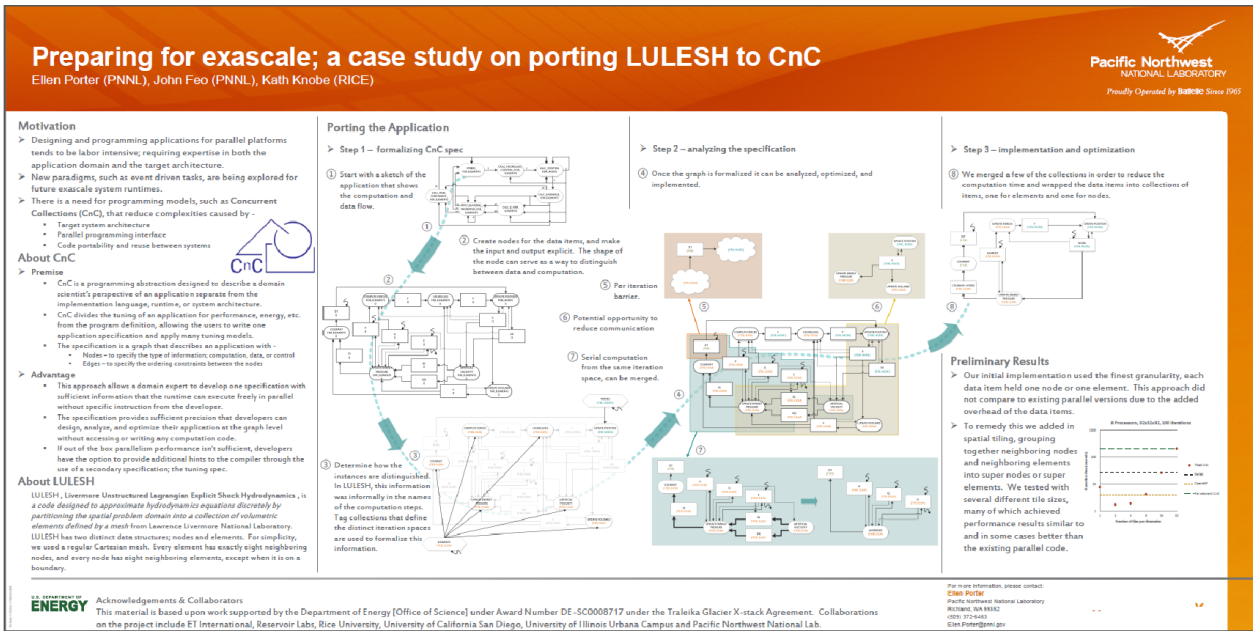


**Figure 1 - PNNL SC2014 Poster**