ECP SOLLVE : Taking OpenMP to Exascale

Super Computing 2017, OpenMP Booth

Martin Kong

Assistant Computational Scientist

Brookhaven National Laboratory

Denver, Colorado, USA
November 2017

EXASCALE COMPUTING PROJECT

Talk Outline

- Overview of The Exascale Computing Project (ECP) and SOLLVE
- Exascale Challenges
- OpenMP and SOLLVE: where are we, and where are we going?
- Conclusion

What is the Exascale Computing Project (ECP)?

- ECP was established to accelerate delivery of a **capable exascale** computing system that integrates hardware and software capability to deliver approximately 50 times more performance than today's petaflop machines.
- A collaborative effort of two US Department of Energy (DOE) organizations:
  - Office of Science (DOE-SC)
  - National Nuclear Security Administration (NNSA)

A **capable** exascale computing system will have a well-balanced ecosystem (software, hardware, applications)

What is a *capable* exascale computing system?

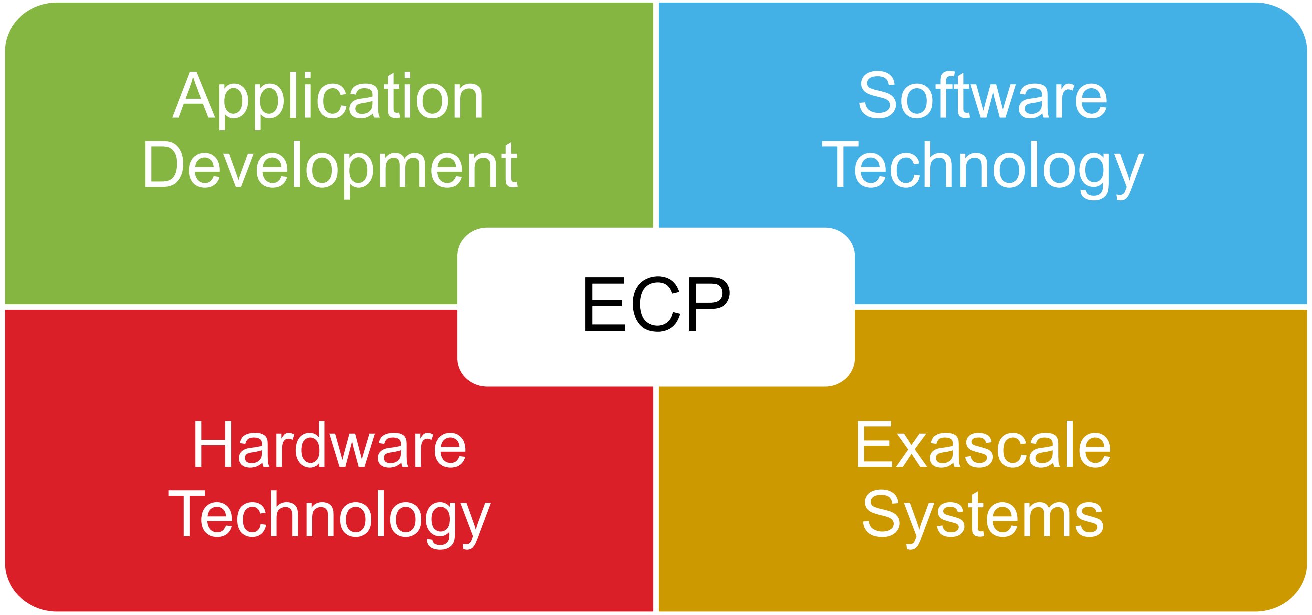
A capable exascale computing system requires an ***entire computational ecosystem*** that:

- Delivers 50x the performance of today's 20 PF systems, supporting applications that deliver high-fidelity solutions in less time and address problems of greater complexity
- Operates in a power envelope of 20–30 MW
- Is sufficiently resilient (average fault rate: ≤1/week)
- Includes a software stack that supports a broad spectrum of applications and workloads

This ecosystem will be developed using a co-design approach to deliver new software, applications, platforms, and computational science capabilities at heretofore unseen scale

ECP EXASCALE COMPUTING PROJECT

Integration and co-design is key



Capable exascale computing requires close coupling and coordination of key development and technology R&D areas

SOLLVE Team: Institutions

RICE

Argonne NATIONAL LABORATORY

BROOKHAVEN NATIONAL LABORATORY

OAK RIDGE National Laboratory

Lawrence Livermore National Laboratory

SOLLVE Components

Applications

- Application Requirements
- Verification and Validation
- Lead: ORNL

Project Coordination

- Team coordination
- Software releases
- Compiler research
- Lead: **BNL**

Enhanced OpenMP Runtime

- Light-weight threads
- Tasklets
- Lead: ANL

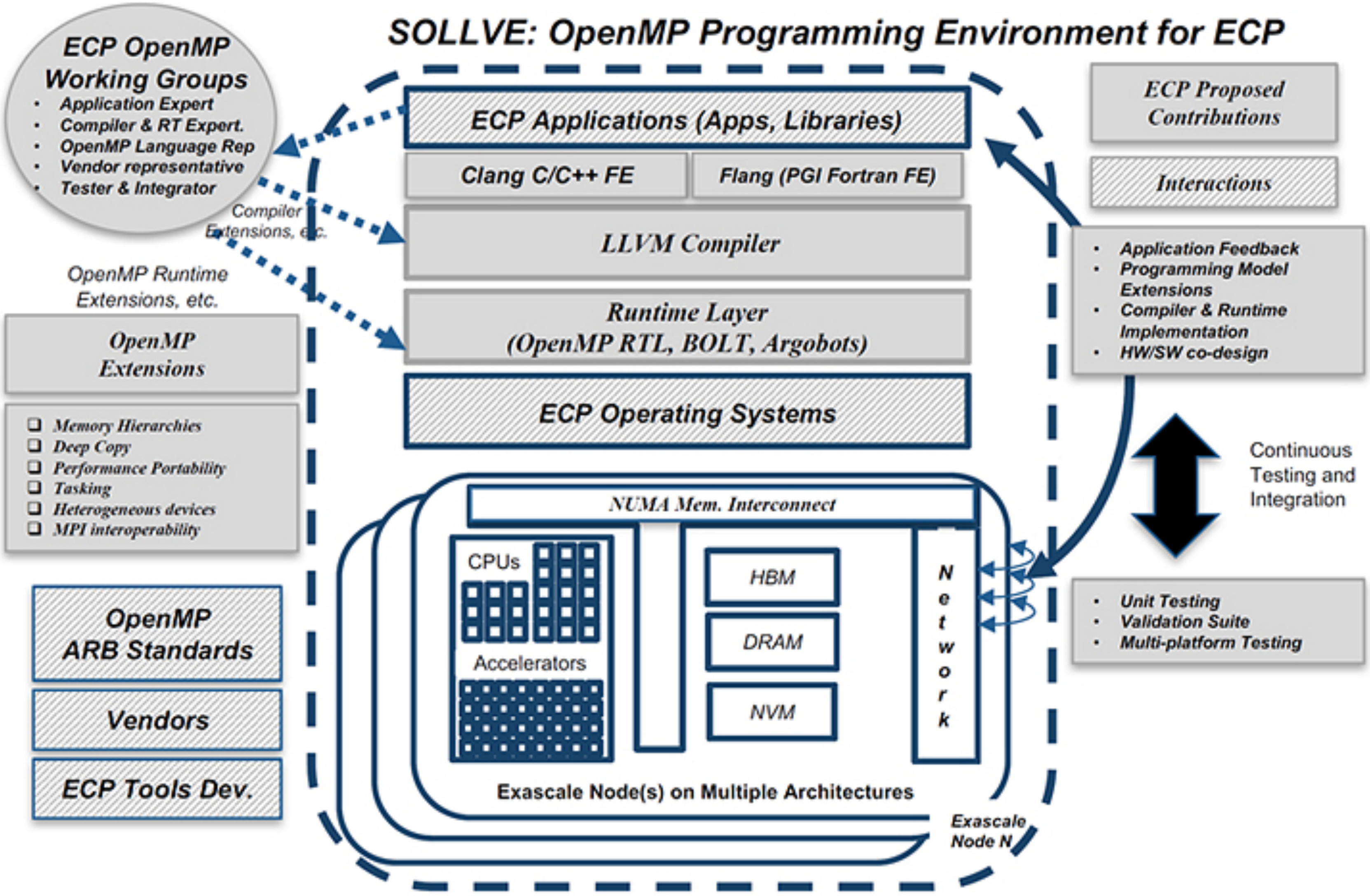
OpenMP Language

- Drive language specification
- Extensions prioritization
- Lead: LLNL

Compiler Optimizations

- Performance portable transformations
- Link-time optimizations
- Lead: Rice, ANL

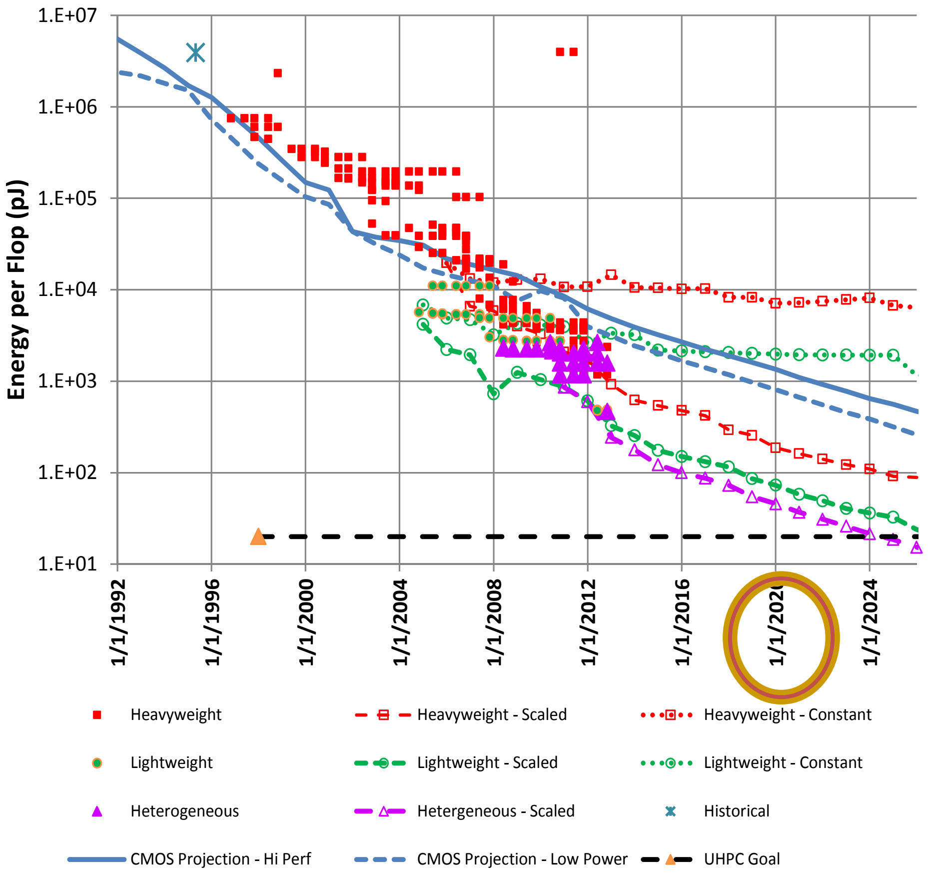
SOLLVE Overview



Exascale Challenges

- "Architectures are moving to support massive degrees of different kinds of parallelism."
- Challenges:
  - Different design choices that affect applications (# threads, heterogeneity)
  - New complex memory hierarchies
    - Extreme NUMA – "islands of NUMA"
    - Different types of memories
  - Current testbeds provide data points for *today's* systems.
  - How do we decide if current programming models are suitable for future architectures (post-Exascale)?
- How to port applications to next generation systems in a performance portable style?

HPC Architectures Evolution



*Reference: Presentation by David Donofrio, LBNL "Exascale Node Architecture Trends".

EXASCALE COMPUTING PROJECT

Areas that we are currently addressing based on Application needs

- Custom Data Maps (Deep Copy) – TR6
- Memory management APIs – TR6
  - Support for NUMA domains – OpenMP 5.0
- Tasking Performance and Accelerator Model – TR6
- Performance portable code generation -- OpenMP 5.0
  - #pragma omp concurrent
- Nested parallelism overheads (implementation via BOLT)

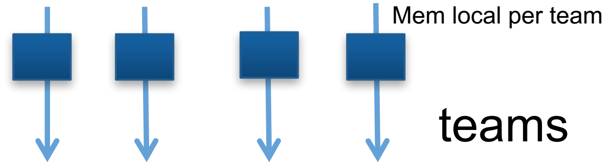
Memory Management API – (Available in TR6)

- APIs to access different types of node memories
- Pre-defined allocators for
  - Large capacity mem.
  - Read-only memory
  - High bandwidth mem.
  - Low latency mem.
  - Local Memory in the same contention group
    - #pragma omp teams -- per local team
  - Threads in the same parallel team
    - #pragma omp parallel

```
!$omp  target teams distribute simd simdlen(512) collapse(3)
!$omp& private(k1,i,j,k,iter,i1,i2,q,ie,addmass,weightssum)
!$omp& private(mass,sumc,x,c)
!$omp& allocate(omp_cgroup_mem_alloc: c,x)

do ie = 1 , nelemd
 do q = 1 , qsize
   do k = 1 , nlev
     min_tmp = minp(k,q,ie)
     max_tmp = maxp(k,q,ie)
     do k1 = 1 , np*np
       c(k1) = sphweights(k1,ie) * dpmass(k1,k,ie)
       x(k1) = ptens(k1,k,q,ie) / dpmass(k1,k,ie)
     enddo
```

Use of memory on the same contention group (e.g. GPU shared memory)
6x speedup on K20x (Titan) a kernel from ACME.

#pragma omp target teams



omp_cgroup_mem_alloc allocates memory local to a contention group (#pragma omp teams)

Deep Copy Support – Complex Data Structures

- Support for moving complex data structures from one memory location to other (e.g. host to accelerator)
- Multiple applications have requested this:
  - Fusion codes (GTC/GENE)
  - QMCPack
  - Etc
- Current OpenMP API supports this via low level APIs and manual pointer attachments

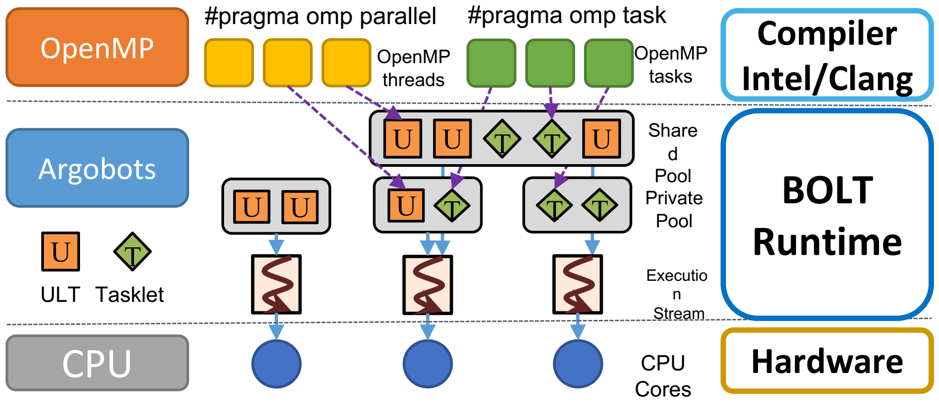
```
#pragma omp parallel
vec_th[omp_get_thread_num()].resize(len);

int **restrict shadows_ptr=shadow.data();
for(size_t tid=0; tid<shadow.size(); tid++)
{
  int *restrict vec_ptr=vec_th[tid].data();
  #pragma omp target
  {
    shadows_ptr[tid]=vec_ptr;
  }
}
```

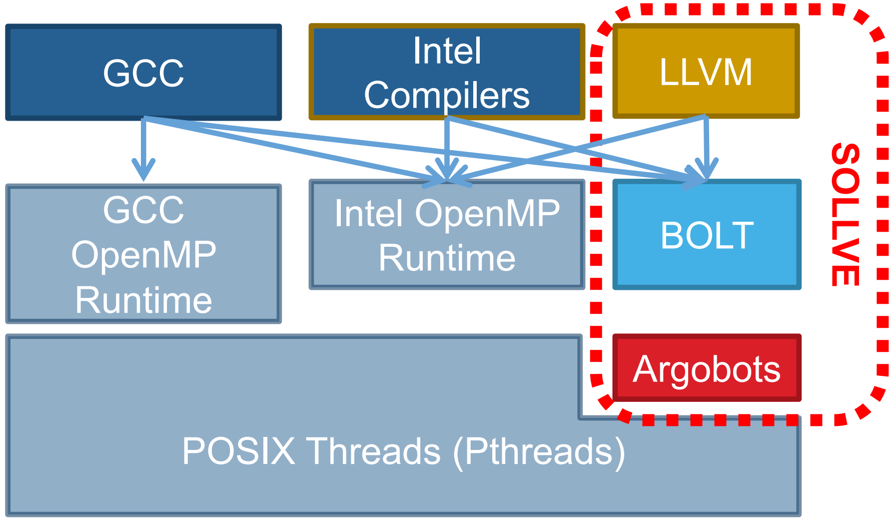
QMCPack is attaching data structures manually in the target regions to link complex data structures in the the accelerator). We need better ways to do this.

Overview of BOLT: OpenMP Over Lightweight Threads

- Key aspects of **BOLT**[1]
  - Compiler simply generates runtime API calls
    - The runtime creates and manages lightweight threads
  - Leverages **Argobots**[2], a highly optimized threading and tasking framework, underneath

- Development
  - Runtime
    - Based on Intel OpenMP Runtime API
    - Generates Argobots work units from OpenMP pragmas
    - Can generate optimized work units depending on code characteristics
  - Compiler
    - Clang/LLVM
    - Passes characteristics of parallel region or task (e.g., existence of blocking calls) to the runtime
    - Extends pragmas with the option "nonblocking"

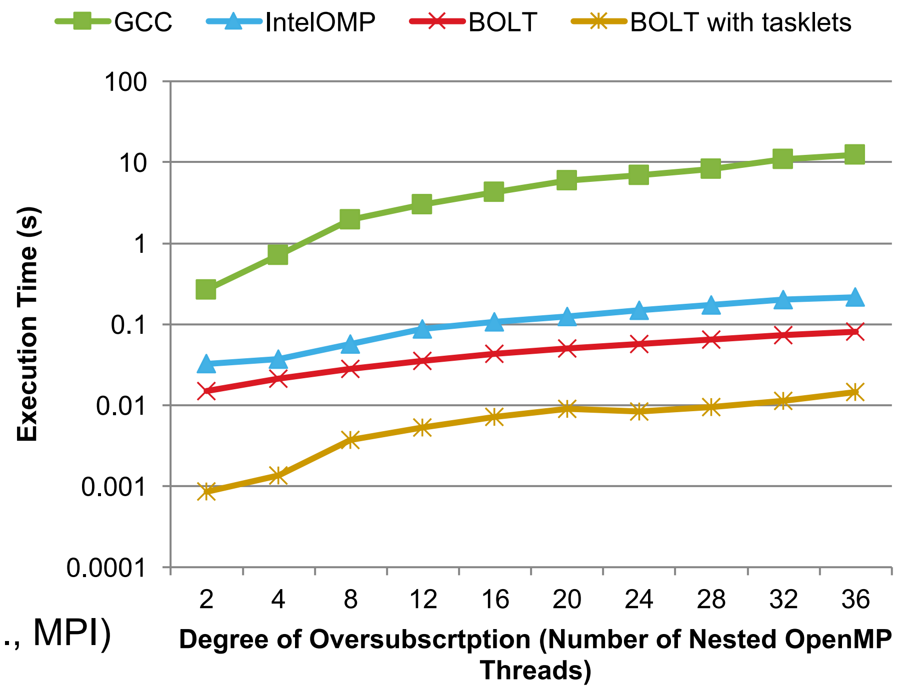




[1] http://www.bolt-omp.org
[2] http://www.argobots.org

BOLT: Better Support for Fine-grained Parallelism

- Increasing importance of fine-grained parallelism with OpenMP
  - Nested parallel regions to expose parallelism
  - Emerging asynchronous task-based applications
- Pthread-based OpenMP implementations handle fine-grained parallelism poorly
  - Managing Pthreads is a heavy task
  - Context-switching between Pthreads is expensive
  - Time-sharing compute resources in HPC is unacceptable
    - Oversubscribing Pthreads is a performance killer
  - Inefficient interoperation between OpenMP and other programming systems (e.g., MPI)
- Advantages of BOLT over existing OpenMP runtimes
  - lightweight low-cost thread management
    - Leverages lightweight threads and tasks instead of Phtreads
  - Oversubscription becomes much less expensive
    - Low context-switch overhead of lightweight threads
    - Cooperative scheduling for efficient resource usage



- Nested parallel loops on a 36-core Haswell machine
- GCC OpenMP (GOMP) does not reuse idle threads in nested parallel regions, all the teams of threads need to be created in each iteration
- Intel OpenMP reuses threads, but those are heavy Pthreads, with high management costs
- BOLT with tasks reduces further thread management costs

Main Compiler R&D Directions

Memory on Accelerators

- Unified Memory (UVM) enhancements
- Memory management (shared mem, HBM, etc)
- Deep copy
- Mapper clauses
- Data-layout transformations

Parallelism

- Leverage upcoming LLVM IR for exposing and exploiting parallelism
- Automatic parallelization
- Analysis and restructuring for new OpenMP ***concurrent*** clause
- Compile-time automatic task placement and scheduling

Code Generation

- Target specific code generation: SIMD, concurrent, GPU, power 8-9
- Automatic offloading
- Auto-generation of new OpenMP pragmas

LLVM + OpenMP Runtime

Work on Unified Memory

- Hierarchical data structure mapping
  - Map the current instances and all indirectly referenced data
  - Programmers' burden
  - Time consuming, error prone
- TR6 will introduce custom mapper
  - Alleviate deep copy
- Unified memory solves deep copy (perfectly?)
  - Indirectly referred data are moved on demand
  - Techniques for prefetching and pinning memory regions

```
#pragma omp target data map(to: A, B) map(from: C)
{
#pragma omp target teams distribute
for (int i = 0; i < N; i++) {
    #pragma omp parallel for
    for (int j = 0; j < M; j++)
        C[i][j] = A[i][j] + B[i][j];
}
}
```

Data Allocation

```
A = omp_target_alloc(N*M*sizeof(int));
B = omp_target_alloc(N*M*sizeof(int));
C = omp_target_alloc(N*M*sizeof(int));
#pragma omp target data is_device_ptr(A, B, C)
{
#pragma omp target teams distribute
for (int i = 0; i < N; i++) {
    #pragma omp parallel for
    for (int j = 0; j < M; j++)
        C[i][j] = A[i][j] + B[i][j];
}
}
```

Let driver make date movement decisions

Proposal for OpenMP Mappers

- Allow to manipulate complex data structures across different targets
- User provides routines to pack, unpack, compute size, serialize, copy, etc.
- Mappers declared via OpenMP runtime
- Runtime invokes user-provided routine
- Mappers are staged and composable
- Any arbitrary data can be mapped

```
class List {
  List* next;
  List* previous;
  plain_data_t data;
  size_t count_nodes() {
    List* cur = next;
    size_t cnt = 0;
    while (cur && cur != this) { cnt++; cur = cur-
>next; }
    return cnt;
  }
  void pack(List* buf, size_t n_nodes) {
    buf[0].next = (List*)(size_t)n_nodes; // stash
length for unpack
    int i = 1;
    List* cur = next;
    while (cur && cur != this) {
      buf[i].data = cur->data;
      cur = cur->next;
      i++;
    }
  }
};

#pragma omp declare packer_to mapper(clist)\
    size(l.count_nodes() * sizeof(List))\
    expr(l.pack(omp_buf, omp_size / sizeof(List)))
```

Get dynamic size

Serialize list into flat array

Integration of Data-Layout Transformations

- Data layout transformations
  - Impact on spatial data locality on program variables by:
    - Permutation of multidimensional arrays
    - Conversion between array-of-structs (AoS) and struct-of-arrays (SoA)
    - Data tiling (combined with iteration space tiling)
  - Need be synergistically optimized with loop transformations
    - Mutually complementary relations between loop and layout transformations
    - Optimal solutions depend on target systems, e.g., CPUs vs. GPUs

```
double C[N][N], A[N][N], B[N][N];
…
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      C[i][j] += A[i][k] * B[k][j];
```

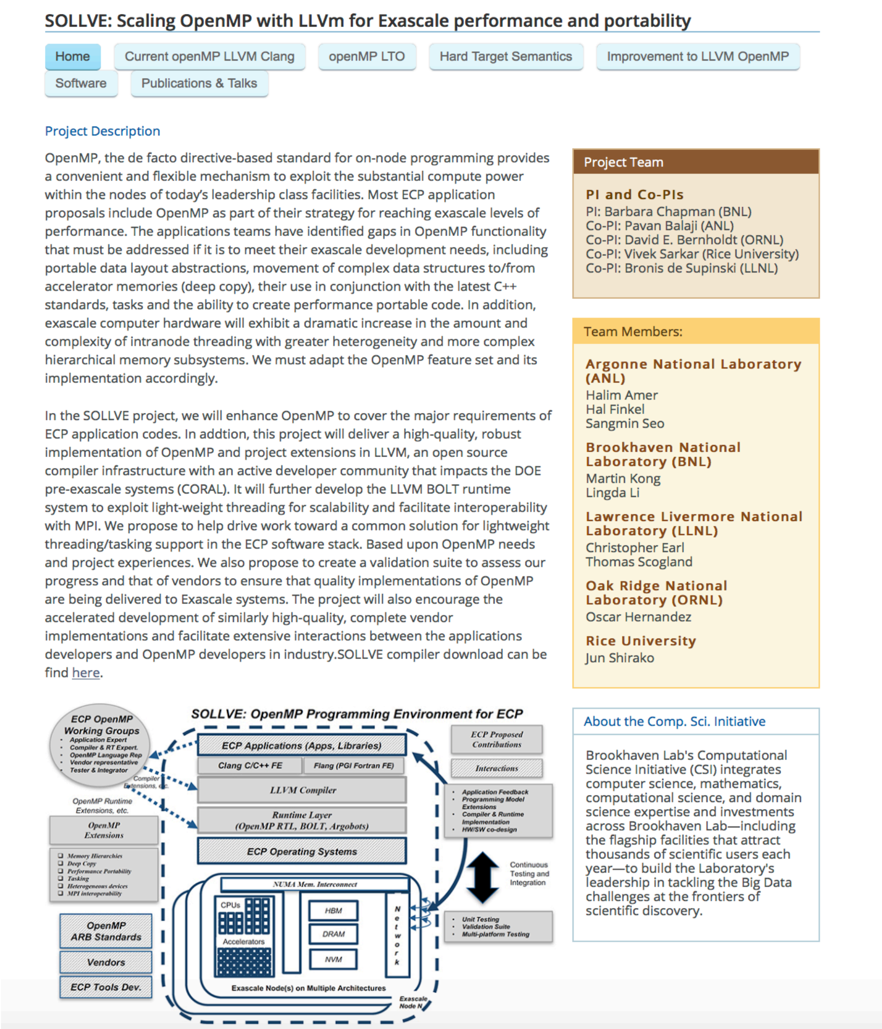
Input kernel

```
double C[N][N];
struct { double A; double B; } S[N][N];
…
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      C[i][j] += S[k][i].A * S[k][j].B;
```

Transformed kernel
- layout permutation (array A)
- change to AoS (A and B)

SOLLVE Website

- Overall project information
- Plans, milestones, roadmap
- Affiliations, teams and organization
- Software releases
- Please visit and/or drop us an email: https://www.bnl.gov/compsci/projects/SOLLVE/



SOLLVE: Scaling OpenMP with LLVm for Exascale performance and portability

Home | Current openMP LLVM Clang | openMP LTO | Hard Target Semantics | Improvement to LLVM OpenMP | Software | Publications & Talks

Project Description

OpenMP, the de facto directive-based standard for on-node programming provides a convenient and flexible mechanism to exploit the substantial compute power within the nodes of today's leadership class facilities. Most ECP application proposals include OpenMP as part of their strategy for reaching exascale levels of performance. The applications teams have identified gaps in OpenMP functionality that must be addressed if it is to meet their exascale development needs, including portable data layout abstractions, movement of complex data structures to/from accelerator memories (deep copy), their use in conjunction with the latest C++ standards, tasks and the ability to create performance portable code. In addition, exascale computer hardware will exhibit a dramatic increase in the amount and complexity of intranode threading with greater heterogeneity and more complex hierarchical memory subsystems. We must adapt the OpenMP feature set and its implementation accordingly.

In the SOLLVE project, we will enhance OpenMP to cover the major requirements of ECP application codes. In addtion, this project will deliver a high-quality, robust implementation of OpenMP and project extensions in LLVM, an open source compiler infrastructure with an active developer community that impacts the DOE pre-exascale systems (CORAL). It will further develop the LLVM BOLT runtime system to exploit light-weight threading for scalability and facilitate interoperability with MPI. We propose to help drive work toward a common solution for lightweight threading/tasking support in the ECP software stack. Based upon OpenMP needs and project experiences. We also propose to create a validation suite to assess our progress and that of vendors to ensure that quality implementations of OpenMP are being delivered to Exascale systems. The project will also encourage the accelerated development of similarly high-quality, complete vendor implementations and facilitate extensive interactions between the applications developers and OpenMP developers in industry.SOLLVE compiler download can be find here.

Project Team

PI and Co-PIs
PI: Barbara Chapman (BNL)
Co-PI: Pavan Balaji (ANL)
Co-PI: David E. Bernholdt (ORNL)
Co-PI: Vivek Sarkar (Rice University)
Co-PI: Bronis de Supinski (LLNL)

Team Members:

Argonne National Laboratory (ANL)
Halim Amer
Hal Finkel
Sangmin Seo

Brookhaven National Laboratory (BNL)
Martin Kong
Lingda Li

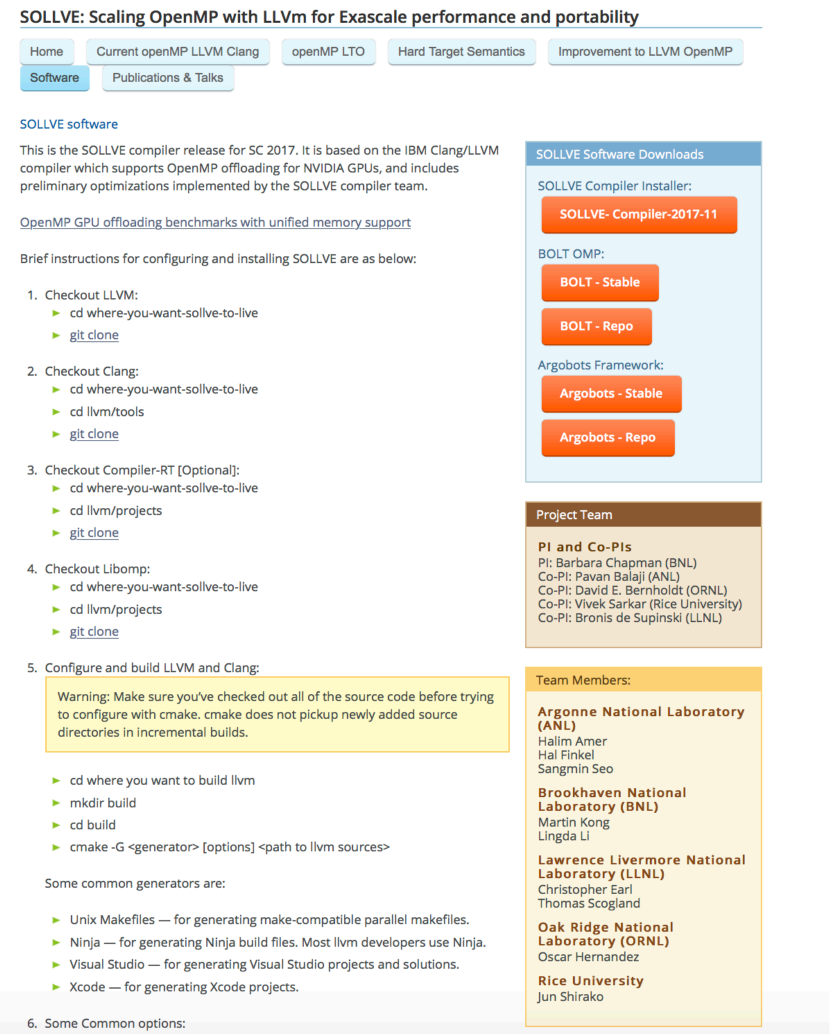
Lawrence Livermore National Laboratory (LLNL)
Christopher Earl
Thomas Scogland

Oak Ridge National Laboratory (ORNL)
Oscar Hernandez

Rice University
Jun Shirako

About the Comp. Sci. Initiative

Brookhaven Lab's Computational Science Initiative (CSI) integrates computer science, mathematics, computational science, and domain science expertise and investments across Brookhaven Lab—including the flagship facilities that attract thousands of scientific users each year—to build the Laboratory's leadership in tackling the Big Data challenges at the frontiers of scientific discovery.



SOLLVE: Scaling OpenMP with LLVm for Exascale performance and portability

Home | Current openMP LLVM Clang | openMP LTO | Hard Target Semantics | Improvement to LLVM OpenMP | Software | Publications & Talks

SOLLVE software

This is the SOLLVE compiler release for SC 2017. It is based on the IBM Clang/LLVM compiler which supports OpenMP offloading for NVIDIA GPUs, and includes preliminary optimizations implemented by the SOLLVE compiler team.

OpenMP GPU offloading benchmarks with unified memory support

Brief instructions for configuring and installing SOLLVE are as below:

1. Checkout LLVM:
   - cd where-you-want-sollve-to-live
   - git clone
2. Checkout Clang:
   - cd where-you-want-sollve-to-live
   - cd llvm/tools
   - git clone
3. Checkout Compiler-RT [Optional]:
   - cd where-you-want-sollve-to-live
   - cd llvm/projects
   - git clone
4. Checkout Libomp:
   - cd where-you-want-sollve-to-live
   - cd llvm/projects
   - git clone
5. Configure and build LLVM and Clang:

   Warning: Make sure you've checked out all of the source code before trying to configure with cmake. cmake does not pickup newly added source directories in incremental builds.

   - cd where you want to build llvm
   - mkdir build
   - cd build
   - cmake -G <generator> [options] <path to llvm sources>

   Some common generators are:

   - Unix Makefiles — for generating make-compatible parallel makefiles.
   - Ninja — for generating Ninja build files. Most llvm developers use Ninja.
   - Visual Studio — for generating Visual Studio projects and solutions.
   - Xcode — for generating Xcode projects.
6. Some Common options:

SOLLVE Software Downloads

SOLLVE Compiler Installer:
SOLLVE- Compiler-2017-11

BOLT OMP:
BOLT - Stable
BOLT - Repo

Argobots Framework:
Argobots - Stable
Argobots - Repo

Project Team

PI and Co-PIs
PI: Barbara Chapman (BNL)
Co-PI: Pavan Balaji (ANL)
Co-PI: David E. Bernholdt (ORNL)
Co-PI: Vivek Sarkar (Rice University)
Co-PI: Bronis de Supinski (LLNL)

Team Members:

Argonne National Laboratory (ANL)
Halim Amer
Hal Finkel
Sangmin Seo

Brookhaven National Laboratory (BNL)
Martin Kong
Lingda Li

Lawrence Livermore National Laboratory (LLNL)
Christopher Earl
Thomas Scogland

Oak Ridge National Laboratory (ORNL)
Oscar Hernandez

Rice University
Jun Shirako

EXASCALE COMPUTING PROJECT

Conclusion

- Exascale is just around the corner.
- Grand challenges to meet exascale requirements (complex and heterogeneous ecosystems, massive amounts of parallelism, power caps, high-productivity).
- SOLLVE’s goal: enhance OpenMP for Exascale with focus on portable high-performance and user-productivity.
- Several efforts at the specification, runtime, application and compiler level.

Acknowledgements

Several thanks to all the SOLLVE team members:

- Argonne National Laboratory: Halim Amer, Pavan Balaji, Hal Finkel
- Brookhaven National Laboratory: Barbara Chapman, Martin Kong, Lingda Li
- Lawrence Livermore National Laboratory: Bronis Desupinski, Tom Scogland, Christopher Earl
- Oak Ridge National Laboratory: David Bernholdt, Oscar Hernandez
- Rice University: Vivek Sarkar, Jun Shirako

EXASCALE COMPUTING PROJECT

Acknowledgements

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations - the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative.

EXASCALE COMPUTING PROJECT