ECP SOLLVE : Taking OpenMP to Exascale

Super Computing 2017, OpenMP Booth

Martin Kong
Assistant Computational Scientist
Brookhaven National Laboratory

Denver, Colorado, USA
November 2017
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)
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.
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

OpenMP Language
- Drive language specification
- Extensions prioritization
- Lead: LLNL

Project Coordination
- Team coordination
- Software releases
- Compiler research
- Lead: BNL

Enhanced OpenMP Runtime
- Light-weight threads
- Tasklets
- Lead: ANL

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

*Reference: Presentation by David Donofrio, LBNL “Exascale Node Architecture Trends”.*
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
  – 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 simdf(len(512)) collapse(3)
$omp& private(k1,i,j,k,iter,il,i2,q,ie,addmass,weightssum)
$omp& allocate(omp_cgroup_mem_alloc: c, x)

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
#pragma omp parallel
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

```c
#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 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](http://www.bolt-omp.org)
\(^2\) [http://www.argobots.org](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 Pthreads
  - Oversubscription becomes much less expensive
    - Low context-switch overhead of lightweight threads
    - Cooperative scheduling for efficient resource usage

- Executions 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

```c
#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

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

```c
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

```c
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];
```

```c
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;
```
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/
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
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.