Loop Scheduling for OpenMP

Vivek Kale

Supercomputing 2018

November 14th, 2018

Dallas, Texas, USA

Scheduling an OpenMP Loop

- A ***schedule*** in OpenMP is:
  - a specification of how iterations of associated loops are divided into contiguous non-empty subsets (called *chunks), and*
  - how these chunks are distributed to threads of the team.[1]
- The syntax for the *clause* `schedule` according to OpenMP 5.0 specification is:

```
#pragma omp parallel for schedule([modifier [modifier]:]kind[,chunk_size])
```

- There are different ***kinds*** of schedules, each of which are defined by OpenMP implementations:
  - `static`
  - `dynamic`
  - `guided`
  - `auto`
  - `runtime`

Proposal: Add a feature to OpenMP that extends the set of pre-defined scheduling types with loop schedules defined by OpenMP application programmers, or users.

1: OpenMP Technical Report 6. November 2017. http://www.openmp.org/press-release/openmp-tr6/

Intel-specific Loop Schedules

Intel’s (and LLVM’s) OpenMP runtime offer additional scheduling types:

- E.g., static stealing
- Are accessed through `schedule (runtime)` and `OMP_SCHEDULE` environment variable
- Cumbersome to use; very complex to extend (need to modify the RTL code and recompile the RTL code)
- Not portable across OpenMP implementations

Proposed feature to support UDSs will provide a portable and flexible way of extending OpenMP’s loop scheduling types.

Reasons for User-defined Schedules

- Flexibility
  - Given the variety of OpenMP implementations, having a standardized way of defining a user-level strategy provides flexibility to implement scheduling strategies for OpenMP programs easily and effectively.
- Emergence of Threaded Runtime Systems
  - Emergence of threaded libraries such as Argobots and QuickThreads argues in favor of a flexible specification of scheduling strategies also.
- Note that keywords *auto* and *runtime* aren't adequate
  - Specifying auto or runtime schedules isn't sufficient because they don't allow for user-level scheduling.

Proposal for User-defined Schedules in OpenMP

Example: glimpse of how a User-defined Schedule (UDS) might look like

```
void myDynStart(...) {}
void myDynNext(...) {}
void myDynFini(...) {}
#pragma omp declare schedule(myDyn) start(myDynStart) next(myDynNext) fini(myDynFini)
void example() {
    static schedule_data sd;
    int chunkSize = 4;
    #pragma omp parallel for schedule(myDyn, chunkSize:&sd)
    for(int i = 0; i < n; i++)
        c[i] = a[i]*b[i];
}
```

UDS identifier

UDS parameters

- The directive `declare schedule` connects a schedule with a set of functions to initialize the schedule and hand out the next chunk of iterations.
- The syntax of the clause `schedule` is extended to also accept an identifier denoting the UDS.
- Instead of calling into the RTL for loop scheduling, the compiler will invoke the functions of the UDS.
- Visibility and namespaces of these identifiers will be borrowed from User-Defined Reductions in OpenMP 5.0.

An Implementation of the Static/Dynamic Schedule with UDS

Data Structures for the User-defined Scheduler

```
// This is a user-supplied type that the UDS needs to store some information and state.
// This can be as easy as a single variable (e.g., for a dynamic) or something complex such
as historic performance data gathered during past loop executions.
typedef struct {
int lb;
int ub;
int incr;
int counter;
double fs;
} loop_record_t;
```

mysd_start
Scheduler's Loop Start

```
// lb, ub, incr, and chunksz are formal parameters required by the specification. lr is a
user-supplied formal parameter and there could be more if needed.
void mysd_start(int lb, int ub, int incr, int chunksz, loop_record_t * lr) {
// We assume that this function is only called by the master thread. Thus, no
// synchronization will be required. Memorize a few things about the loop schedule.

        lr->lb = lb;
        lr->ub = ub;
        lr->incr = incr;
        lr->chunksz = chunksz;
```

Mysd_next
Scheduler's Loop Next

```
// lower, upper are formal parameters required by the specification.
// lr is a user-supplied formal parameter and there could be more if needed.
// Signature: void X_next(int *, int *, ...)
void mysd_next(int * lower, int * upper, loop_record_t * lr) {
    int start;
 if(lr->counter < (lr->ub - lr->lb)) {
     *lower = lr->fs*(lr->ub - lr->lb)*(tid/numThreads);
     *upper = *lower + lr->fs*(lr->ub - lr->lb)/numthre
 ads;
      lr->counter += (*upper - *lower)/numThreads
 }
 else
 {
 #pragma omp atomic capture
     {
         start = lr->counter;
         lr->counter += lr->chunksz * lr->incr;
     }
     *lower = start;
     *upper = start + lr->chunksz * lr->incr;
}
```

Scheduler's Loop Finish

```
// Signature: void(...)
void mysd_fini(loop_record_t * lr) {
    // Do nothing
}
```

User-defined scheduler.

```
#pragma omp declare schedule(mysd) init(mysd_start)
next(mysd_next)
void example() {
    loop_record_t lr;
 #pragma omp parallel for schedule(mysd, &lr)
    for (int i = 0; i < n; i++) {
        a[i] = s * a[i] * b[i];
    }
}
```

Application loop specifying a User-Defined Schedule

Status of Proposal for Adding UDS to OpenMP

<u>To be added to OpenMP 5.1 - vote in Santa Clara this January</u>

→ We aim to have users to test and support the proposal.
→ Work in progress on reference implementation in Intel's version of LLVM.

Oct 2017

Presented proposal at OpenMPCon SUNY Stony Brook

Got Michael Klemm interested in the ideas.

Feb 2018

Presented proposal in Austin

Got feedback from community, reference implementation of UDS started Intel's LLVM OpenMP library by Intel Stanta Clara and Intel Russia

Oct 2018

Got additional support from UBasel and TU-Darmstadt

Gained support during Barcelona Face-to-face meeting.

Jan 2019

Presenting ticket for vote at Santa Clara OpenMP F2F

Will aim to have users test and support proposal around then.

RAJA Framework at LLNL

- RAJA helps scientists at DoE labs write programs with parallelizable loops that are portable across different architectures.
- RAJA has several policies to schedule iterations of the loops of an application to either cores of a CPU or cores of a GPU.
- The miniApp LULESH demonstrates the use of RAJA well.
- LULESH has OpenMP loops with load imbalances.
- **Problem:** RAJA code with loops that can benefit from my loop scheduling strategies → how to do it without disturbing RAJA's goals?
- Worked with David Beckingsale from LLNL to address the problem.

Lightweight Loop Scheduling in RAJA: lws-RAJA

Code through hand transformation or maybe ROSE.

```
#include "vSched.h"
#define FORALL_BEGIN(strat, s,e, start, end, tid, numThds )
loop_start_ ## strat
(s,e ,&start, &end, tid, numThds); do {
#define FORALL_END(strat, start, end, tid) } while(
loop_next_ ## strat (&start, &end, tid));
void* dotProdFunc(void* arg)
{
  int startInd = (probSize*threadNum)/numThreads; int endInd
=  (probSize*(threadNum+1))/numThreads;
  while(iter < numIters) {
   mySum = 0.0; //reset sum to zero at the beginning of the
product if(threadNum == 0) sum = 0.0;
   if(threadNum == 0) setCDY(static_fraction , constraint.
hunk_size);

#pragma omp parallel
FORALL_BEGIN(statdynstaggered , 0, probSize , startInd,endInd
,threadNum, numThreads)
for (i = startInd ; i < endInd; i++) mySum += a[i]*b[i]
 FORALL_END(statdynstaggered , startInd , endInd,threadNum)
pthread_mutex_lock(&myLock);
  sum += mySum;
  pthread_mutex_unlock(&myLock);
  pthread_barrier_wait(&myBarrier);
  if(threadNum == 0) iter++;
  pthread_barrier_wait(&myBarrier); } // end timestep loop
}
```

MPI+OpenMP code explicitly using lightweight scheduling.

RAJA User Code

```
RAJA::ReduceSum<RAJA::seq_reduce, double> seqdot(0.0);
RAJA::forall<RAJA::omp_lws>(RAJA::RangeSegment(0, N), [=]
(int i) {

seqdot += a[i] * b[i]; });

dot = seqdot.get();
std::cout << "\t (a, b) = " << dot << std::endl;
```

RAJA library implementation with policy `omp_lws`

```
#include "vSched.h"
#define FORALL_BEGIN(strat, s,e, start, &end, tid, numThds) do {
#define FORALL_END(strat, start,tid)); start, end, tid, numThds ) loop_start_
## strat (s,e ,&end, tid) } while( loop_next_ ## strat (&start, &end)
template <typename Iterable , typename Func >
RAJA_INLINE void forall_impl(const omp_lws<&, Iterable&& iter, Func&&
loop_body) {
RAJA_EXTRACT_BED_IT(iter);
int startInd , endInd;
int threadNum = omp_get_thread_num();
int numThreads = omp_get_num_threads();
FORALL_BEGIN(statdynstaggered , 0, distance_it , startInd , endInd , threadNum
, numThreads) for (decltype(distance_it) i = startInd; i < endInd; ++i) {
loop_body(begin_it[i]); }
FORALL_END(statdynstaggered , startInd , endInd , threadNum)
}
```

→ Significantly reduces lines of code for application programmer to use strategy: **<u>easy-to-use</u> strategies.**
→ Improves ***<u>portability</u> of loop scheduling strategies***.

Early Results for lws-RAJA

- **<u>Experimentation with Jacobi example code</u>**
  - Implementation overhead: 4%
  - Using lws-RAJA instead of explicit lws reduces lines of code, by 58%
  - Using lws-RAJA in place of RAJA adds 0 lines of code and requires one change to the policy.

- **<u>Experimentation with LULESH</u>**
  - Only one place in RAJA Lulesh code to change the policy for *each* loop.
  - Performance is still being evaluated.

Plug: Synergistic Load Balancing and Loop Scheduling

- My prior work focused on within node loop scheduling for MPI+OpenMP programs.
- Need to combine with across-node balancing.
- Charm++ supports across-node load balancing.
- We extended Charm++'s loop scheduling mechanisms with my scheduling strategies.
- Work with Harshitha Menon, Karthik Senthil → *SC17 Best Poster Award Candidate*
- Later work w/ Harshitha Menon along with Mathias Diener and Kavitha Chandrasekar.

Possible Outcomes of lws-RAJA

- **More users:** Solicit OpenMP users for RAJA at conferences.
- **More policies**: Add more tasking / scheduling strategies as policies in RAJA.
- **lws-RAJA→ uds-RAJA:** If succeeds, can also have RAJA support User-Defined Schedules as a new policy, rather than just my experimental scheduler library.
  - *Cleaner:* since uses a standard interface as proposed for OpenMP.
  - *Faster:* The scheduler implementations will be tuned by CS community.
- **Autotuning:** Use LLNL's Apollo to auto-tune value of scheduling strategy parameters in lws-RAJA.

Appendix

Contents

- Summary of OpenMP worksharing construct for a parallel loop
- Explanation of the need for novel loop scheduling schemes in OpenMP
- Utility of other novel loop scheduling strategies
- Basis of proposal for feature in OpenMP that facilitates for User-defined Schedules
- Issues to consider for addition of feature to support User-defined Schedules

OpenMP Loops Redux

- OpenMP provides a loop worksharing construct that specifies
  - How the logical iteration space of a loop is cut into chunks
  - How the resulting chunks are assigned to the work threads of the parallel region.
  - Syntax of worksharing construct:

```
#pragma omp for [clause[ [,] clause] ... ]  for (int i=0; i<100; i++){}
```

- Loop needs to be in canonical form, that is, adhere to certain properties
- The *clause* can be one or more of the following: `private(…)`, `firstprivate(…)`, `lastprivate(…)`, `linear(…)`, `reduction(…)`, **`schedule(…)`**, `collapse(...)`, `ordered[…]`, `nowait`, `allocate(…)`
- We focus on the clause ***`schedule(…)`*** in this presentation.

1: OpenMP Technical Report 6. November 2017. http://www.openmp.org/press-release/openmp-tr6/

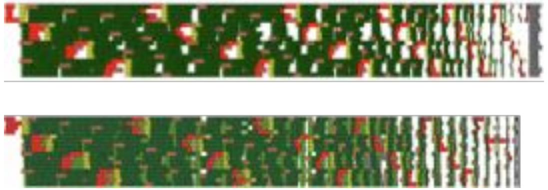
Need Novel Loop Scheduling Schemes in OpenMP

- Supercomputer architectures and applications are changing.
  - Large number of cores per node.
  - Speed variability across cores.
  - Complex dynamic behavior in applications themselves.
- So, we need new methods of distributing an application's parallelized loop's iterations to cores.
- Such methods need to
  1. ensure data locality, reduce synchronization overhead and maintain load balance [1,2];
  2. be aware of internode parallelism handled by libraries such as MPICH[3];
  3. suitable for the needs of a particular loops and machine characteristics; and
  4. adapt during an application's execution.
- Some customer demand for the SSG-DRD EMEA HPC team.

1: R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. Journal of ACM 46(5):720–748, 1999.
2: S. Donfack, L. Grigori, W. D. Gropp, and V. Kale. Hybrid Static/Dynamic Scheduling for Already Optimized Dense Matrix Factorizations. In IEEE International Parallel and Distributed Processing Symposium, IPDPS 2012, Shanghai, China, 2012.
3: E. Lusk, N. Doss, and A. Skjellum. A High-performance, Portable Implementation of the Message Passing Interface Standard. Parallel Computing, 22:789–828, 1996.

Utility of Novel Strategies Shown

- Utility of novel strategies is demonstrated in published work by V. Kale et al [1,2] and others.
- For example, mixed static-dynamic scheduling strategy with an adjustable static fraction.
  - To limit the overhead of dynamic scheduling, while handling imbalances, such as those due to noise.



CALU using static scheduling (top) and f_d = 0.1 (bottom) with 2-level block layout run on AMD Opteron 16 core node.

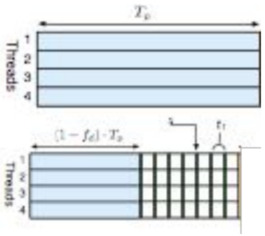
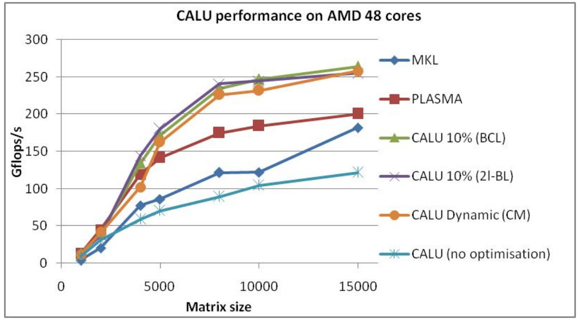
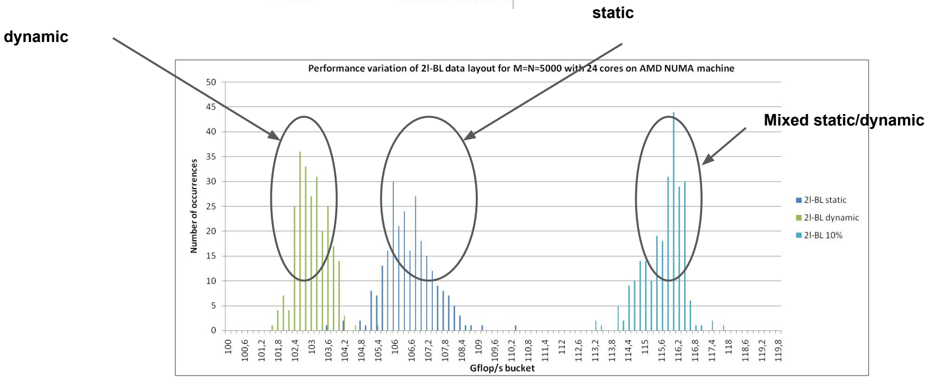


Diagram of static (top) and mixed static/dynamic scheduling (bottom) where f_d is the dynamic fraction.





1: S. Donfack, L. Grigori, W. D. Gropp, and V. Kale. Hybrid Static/Dynamic Scheduling to Improve Performance of Already Optimized Dense Matrix Factorizations, IPDPS 2012.
2: V. Kale, S. Donfack, L. Grigori, and W. D. Gropp. Lightweight Scheduling for Balancing the Tradeoff Between Load Balance and Locality 2014.

Proposal for a User-defined Scheduling Scheme

1. We aim to specify a user-defined scheduling scheme within the OpenMP specification[1] .
2. The scheme should accommodate an arbitrary user-defined scheduler.
3. These are the elements required to define a scheduler.
   a. Scheduler-specific data structures.
   b. History record: adapt the loop schedule based on previous loop invocations and/or user-specified carry parameters.
   c. Specification of scheduling behavior of threads.

Issues to Consider

1. **Issue:** How do we handle loop having indices that are non-monotonic?
    - *One proposed resolution*: We restrict users to to use monotonic loops for the initial version of UDS.

2. **Issue**: How do schedules guarantee correct execution when a global variables are used?
    - *One proposed resolution*: TBD. A proposed solution needs to be discussed.

3. **Issue:** How can UDS be compatible with clause concurrent?
    - *One proposed resolution*: can enforce to users that concurrent not be used with user-defined schedules.